2017

Agent-based Crowd Simulation Modelling for a Gaming Environment

Songqiao Sun
University of Windsor

Agent-based Crowd Simulation Modelling for a Gaming Environment

By

Songqiao Sun

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2017

Agent-based Crowd Simulation Modelling for a Gaming Environment

by

Songqiao Sun

APPROVED BY:

M. Khalid
Department of Electrical and Computer Engineering

D. Wu
School of Computer Science

I. Ahmad, Advisor
School of Computer Science

October 12, 2017

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyones copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Crowd simulation study has become a favourite subject in the computer graphics community in the past three decades. It usually is a sub-function within many applications such as video games, films, and public security. This thesis proposes an independent crowd simulation model that is capable of running an Agent-based method through a gaming environment. It can simulate realistic human crowds with user-controllable features to provide a gaming-like experience. Our approach features an enhanced rendering system based on Distinguishable Agents Generating Method (DAGM). This method can generate distinguishable and scalable 3D human models in real-time. We also introduce our Multi-layer Collision System (MCS), which features a collision-message collection system and an evaluation processing system. We also introduce Building & City-planning Generating System (BCGS) for the purpose of setting up obstacles for the crowd during an evacuation simulation. Moreover, in this thesis, we also extend the study to other aspects such as crisis training and human animations to provide a complete agent-based crowd simulation model.

Key Words: Agent-based Model, Crowd Simulation, 3D Visualization, Crisis Training, Gaming Environment, Visual arts

DEDICATION

To my mom, my grandma and the one I love.

AKNOWLEDGEMENTS

First of all, I would like to thank my supervisor Dr.Imran Ahmad, for his inspirational thoughts and the willingness of sharing his wisdom to me. This thesis cannot be accomplished without my supervisor's insightful feedback and guidance. It is an honour to be one of his students.

Second, I would like to appreciate my thesis committee members Dr.Dan Wu and Dr.Mohammed Khalid for kindly offering their time, encouragement, and valuable comments to me.

Meanwhile, I would like to express my special thank to my girlfriend Yaxin Li who supported me through this unforgettable period.

Last, I would like to express my best gratitude to my mom, grandma, and the rest family for the love, support, and sacrifice. You give me the chance to become the person I love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS/SYMBOLS

3D Three-dimensional

ACS Agent-based Crowd Simulation

BCGS Building & City-planning Generating System

BDI belief, desire, intention

CPU central processing units

DAGM Distinguishable Agents Generating Method

FPS frames per second

GPU graphics processing units

IDE Integrated development environment

k-NN k-Nearest Neighbours

LOD level of details

MACES Multi-Agent Communication for Evacuation Simulation

MCS Multi-layer Collision System

PC personal computer

PMFserv Performance Moderator Functions server

SCMR Separating Components Modelling Rule

UI user interface

CHAPTER 1

Introduction

Crowds are common and unavoidable in our daily lives. However, simulating this general phenomenon using computer graphics is still a challenging work. The need for crowd simulation long existed in film industry [24] and video games [4] for entertainment purposes, in safety projects like crisis training [31], in city architectural planning, and even in the education field. The natural crowd is usually gathered because people or animals share a 'common purpose' or a 'mutual destination' [11]. However, those subconscious decisions are complex to computers and a simulation usually cost massive computational resources at the early stage. Due to the limitations of computer performance, some early models tend to focus on the dynamics method of a crowd as a whole rather than its individual movements, which are simplified methods to deal with the complicated crowd movements.

Examples of early models are the Flow-based model [38, 6] and the Entity-based model [45, 23]. Hence, the crowd models proposed in early studies normally feel 'unrealistic' without individual behaviours [42, 29], and lack a genuine physical collision system [3, 1] to support the simulation.

However, computer performances have been improved significantly in recent years. Current computers have the ability to process much more sophisticated crowd simulation programs than before. More functions [12, 22] and more detailed arrangements can be applied to many recent simulation models; such as using high-quality $3D$ modelling for both agents and environments [14] and individual interactions [19]. More importantly, current computers' performance can support the simulation operating in realtime [43, 39]. The agent-based crowd simulation (ACS) [27, 18, 5], a more

advanced method, was firstly introduced in helbing's model [13]. The principal idea of ACS is controlling the dynamic of crowds that are constructed autonomously. ACS model is considered as the best method [13] for the purpose of simulating crowds with realistic behaviours, since it provides the movement by controlling every agent individually. However, the cost of the simulation is also the highest. Thus, decreasing the computational cost while increasing the simulation performances like the scalability, flexibility, and realism have remained in high demand in the ACS study topics.

1.1 Research Motivation

The ACS studies usually focus on optimizing the decision-making ability and navigation efficiency for autonomous crowds [8, 21, 10] to achieve realistic simulation outcomes. However, in this thesis, we meant to improve the visual-performance and realtime interactions between users and the machine for a humanoid crowd simulation model instead of improving the locomotion method or decision-making approaches. The efficiency of moving algorithms applied to crowds has been a popular in many ACS [30, 28, 35] studies before, but many movement methods are similar to each other as the basic idea is mainly related to tackling the angular and tangential velocity autonomously when agents meet obstacles. Another fact is that in computer architectures, central processing units (CPU) are capable of processing more sophisticated algorithms than what we need in crowd simulation. In contrast, $3D$ computer graphics still consume considerable power from graphics processing units (GPU). Therefore, we attempt to create the $3D$ agent with relatively low vertices, while the render quality is not sacrificed. We propose a dynamic collision-evaluation system to analyze agents' surrounding information while they are moving for the realtime monitoring study. Our method plan is to monitor crowd movement in definable city maps which are more complex than a plane. Thus our autonomous crowd can be examined in different situations. We introduce the specific functions in later paragraphs.

1.1.1 Evacuation Simulation Needs

Our thesis proposes an urban evacuation system featuring a simulation of a human stampede that takes place in a high-density crowd. Crisis training system [32, 44] is a practical topic for Crowd Simulation studies, but few studies have focused on stampede probability [30]. The stampede is expected to happen when a large number of people or animals are fleeing in fear [11, 25]. In the worst case, a stampede happens under the condition of extreme dense situation, the death toll can be high. Although historically crushes have been a relatively rare occurrence, the casualty number of such events cannot be ignored. This is the original motivation for us to simulate this man-made disaster. Using this simulation, we could make predictions about and protect against stampedes in the future. Two features in our thesis method are essential for our stampede study, first, our simulation can adjust crowd density and size to achieve the appropriate density-level and crowd size that are enough for the stampede situation; second, every agent has collision system and event-trigger system to decide the surrounding condition and the falling possibility.

Figure.1 shows a case of dense crowds leading to a stampede. On December 31, 2014, a severe stampede happened in Shanghai, near Chen Yi Square, where nearly 300,000 people gathered in one relatively small area for the New Year celebration. This tragedy, unfortunately, led to 36 fatalities and 49 injuries.

1.2 Research Methods

Visual rendering is the very direct way to show a crowd simulation. However, it has not received much of the attention in recent research. It commonly remains at the basic level in many crowd simulation studies while other aspects like collision system and navigating system have been well documented. However, improving on visual rendering quality does not only provides a plausible model for observation need, but also helps in achieving authentic results. In this thesis, we have devised three approaches based on the motivations we just mentioned.

Fig. 1: 2014 Shanghai stampede in New Years eve

1. We use the Distinguishable Agents Generating Method (DAGM) to model agents that can build up the crowd. This method can generate distinguishable-appearances agents with lower polygon mesh volume. A set of bionic human animations corresponding to the agent model are also deployed for lifelike motion requirements.

2. We have implemented the Multi-layer Collision System (MCS) for the crisis training study. Our MCS can provide synchronized feedback of the density level around one agent. Based on the MCS, we also successfully simulated a city evacuation scenario with the potential for human crashing.

3. The Building & City planning Generate System (BCGS) is designed to support the environment and surrounding settings. In order to create a virtual system consisting of more than just the human crowd; buildings and city roads are also necessary. This system can place obstacles and walkable paths for each agent in the crowd where autonomous agents move based on the priority rules.

(a) (b) (c)

Fig. 2: Several simulation scenes of our crowd simulation frameworks

Figure.2 shows some general sketches from the simulation application. Figure.2-(a) shows a group of male agents has gathered around the destination on the map while some are still in the navigation process. Figure.2-(b) shows the interface of agent control panels with information receiver at the top. Also the crowd and buildings were generated by the control panel. Figure.2-(c) shows the initiation of the navigation process on female characters to the desired destination and the city control panel on the right of the screen.

1.3 Thesis Objectives

This thesis intends to study on three topics among the Agent-based crowd simulation and with the different objectives in the selected topics as explained below.

Our first objective is to improve the modelling method for agent-rendering by providing the realistic graphics with fewer pre-load resources. In previous research, the common method for agent-rendering in crowd simulation are mainly divided by graphical types, $2D$ shapes or $3D$ polygons. When the model is applied with $2D$ shapes like dots or boxes to represent human agents, the $2D$ model has the advantage of less GPU power consumption. It is practical when summing up the dynamic information such as the moving flows and patterns, but the non-realistic feeling is the crucial drawback. Some studies [28, 26, 30] have implemented $3D$ graphics in their models. However, their rendering model either uses several polygon shapes to assemble the low-quality agents with the same appearances or their model have to

load a massive volume of pre-modelled $3D$ resources to achieve the variety of agent rendering types. In comparison, we achieved a high-level $3D$ graphic model with massive agent appearances changes without the need of pre-loading model resources.

Next, we want to design a dynamic system for agents that can receive and analyze information at realtime. We implemented the priority rule as the basic navigation method for our agents to avoid the possible collision with one or more walkers while the navigating is in process. Our MCS is the system we designed to help the surrounding evaluation with a linear time algorithm complexity, but it can efficiently perform the realtime collision feedback. This system is achieved with the help of collision-detection function. On the other hand, our MCS can also assist our crisis training study, especially, for the stampede simulation. In comparison, very few researches have provided a collision system with realtime collision feedback. Also, the research that has focused on evacuation simulation as its topic still rarely simulated stampede events by its collision systems.

The third objective of our thesis is to present a method that can generate a city scheme with blocking function to present environmental obstacles. Some approaches have proposed similar functions to generate surroundings or a village-like architecture model for realism purposes [37]. However, the interactions between a city environment and human groups are usually weak. Buildings are typically used as representative modelling instead of real obstacles, which can block agents when the emergency happened. Thus, in our model, structures and city-schemes could be assigned during the simulation with random and controllable factors. Hence, the surroundings are different with every time it is initiated.

1.4 Contributions

Our contributions are therefore the following: We developed a system for the ACS study using the *Unity* game development engine. We created the DAGM for our ACS's agent generating system, featuring the ability of generating high-quality rendered and body-types individualized agents in the simulation. By applying our DAGM,

we can automatically create different types of agents without the need of manually remodelling on the specific parts to make changes.

Second, we established a new dynamic scheme with two components involved. Our MCS can evaluate individual density status at the realtime and supports our crisis training purposes. Also, the priority-rule based navigation system can provide a method to solve the 'deadlock' problem that commonly existed in many decision-making models.

Third, on the surrounding system aspect, our BCGS can create buildings that cohesively interact with moving crowds. The major advantage of BCGS is that buildings can be randomly placed in a pre-defined area within a few operating steps, but it is flexible for some specific modifications such as the number of buildings and redefining the 'construction area' to satisfy various city looks and patterns.

Lastly, our study can contribute to crisis training studies using the evacuation results we received from the simulation system. Our model can also be used in gaming environments with autonomous crowd needs. Because we designed our model with a high accessibility, it is also suitable for educational purposes.

1.5 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we introduce and review several typical agent-based crowd simulation models. In Chapter 3, we demonstrate and discuss our proposed methodologies: the DAGM, MCS and BCGS. The details of experiments and the evaluation of our proposed methods are presented in Chapter 4. Finally, we present our conclusions in Chapter 5.

CHAPTER 2

Related Works

In this section, we provide an overview of related works. However, crowd simulation has many different aspects to investigate, such as dynamic methods, graphics models, and global settings, etc., which a single model cannot cover comprehensively. For example, an ACS approach may study the dynamic methods for acquiring a non-collision navigation model, which improves on moving algorithms. Also, they could study on the agent's characteristics to provide a realistic behaviour system. Thus, we review both the rendering and dynamics methods in previous studies and compare them with ours. Additionally, other study topics like surrounding systems and crisis training studies are also the focal points of this thesis. So we also provided some reviews on other works related to this two topics.

2.1 Agent-based Dynamics Method

Several authors proposed their models for the Agent-based crowd simulation. Helbling et al.'s *Social Forced Model* [13] settle the fundamental idea about how to apply the agent-based approach to a crowd simulation model. In this model, authors apply the repulsion and tangential forces on each agent; therefore, the steering agent can avoid obstacles and other moving agents. However, the major drawback of this approach is that every agent shares the same attributes without individual identity; also, agents are navigated under the same moving speed; unlike a real human crowd, which has individual differences regarding mobility. Besides, when Helbling's moving method is applied on a $3D$ model, the agents tends to '*shake*' and '*vibrate*' unpredictably

with increasing density and non-priority rule applied. Overall, Helbing's model has introduced the fundamental conception of the ACS study and his research inspired us enormously.

Jan Ondrej et al. [28] proposed a *Synthetic-Vision Based Steering* approach for crowd simulation, where they drive agents according to 'visual perception'. In their approach, agents use the visual field to receive information about their surroundings. They analyzed the reason for a collision occurrence and what time it about to happen. Then based on the information above, they proposed their collision-free locomotion model. The primary principle is that each agent will adjust its angular and tangential velocity based on the bearing angle with the obstacle in the current view and its time-derivative. From the experimental results, this collision-free dynamic approach shows the moving crowds similar to lines of disciplined soldiers passing through each other. Figure.3 shows an example of crowd moving toward and across each other with the collision-free approach applied. In comparison, our thesis is focusing on simulating an ordinary crowd of citizens without any training experience. With this condition, there is no reason to present a collision-free model because our citizens will flee in a chaotic and panicked manner in the scenario. Although Jan Ondrej's and our simulation offer two different types of crowd motions, our study is still greatly influenced by their work.

Dutra et al. proposed a *gradient-based steering* [7] algorithm which is derived from the vision-based steering method. In their algorithm, each agent considers both the risks of future collision and the desired destination. After computing the partial derivatives, the agent moves by following the gradient on the map. This model has a better range of perceiving information for each agent to make decisions of moving.

Classification and regression algorithms also have some contributions on the ACS simulation process. Vermeulen et al. [40] proposed a comparative method based on the k-Nearest Neighbour (k-NN) for solving the collision avoidance problem in their crowd simulation program.

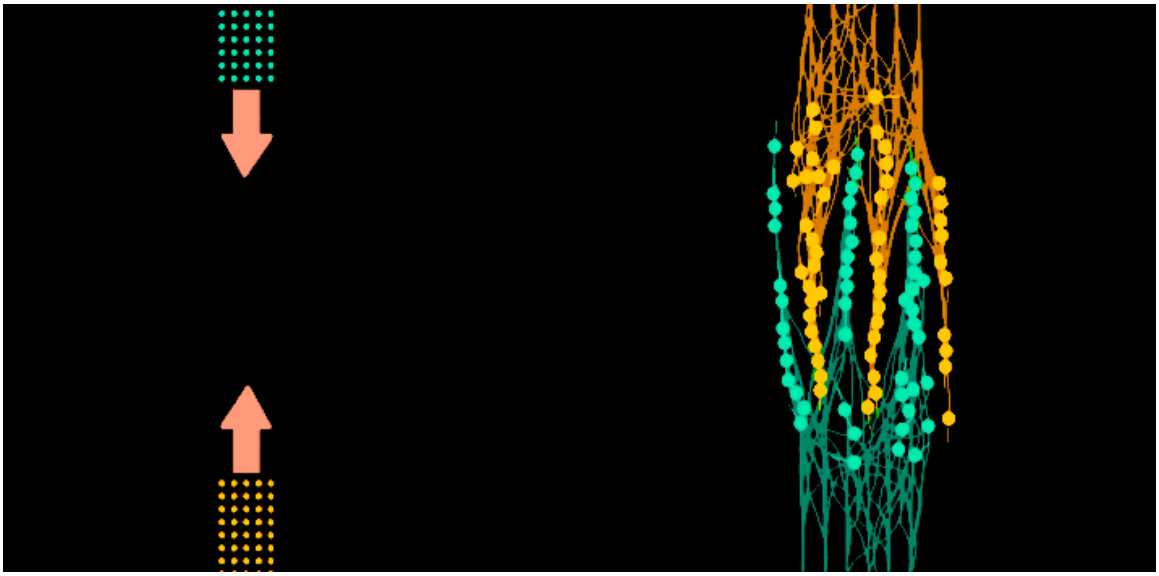


Fig. 3: Example of crowds moving under the Synthetic-Vision model steering

2.2 Crowd-Rendering Method

For the rendering aspect, some of the studies successfully applied the $3D$ graphics [15, 25] into their simulation model. The necessity of $3D$ models in a crowd simulation environment is not merely for the visual performance but also for the advanced physical interactions in a $3D$ system [7], which is the most similar computer model to the real world environment. Furthermore, all agents can have one more dimension to support the collision system, where the $2D$ model [41] commonly does not include the actual physical sizes of the agents' bodies. That is the reason for some $2D$ particle system methods that [9, 20, 2] can achieve extremely dense crowds by sharing one spot for two or more agents. It means the positions is overlapped, but a real human crowd is unlikely to reach the same density level as in a particle system. Thus we decided to apply a human body ratios $3D$ characters in our model to enhance the realism.

Nuria et al. [30]; the authors proposed a *HiDAC* system with cellular-automata models. Although this model only uses several basic polygon shapes to assemble a humanoid character in low-quality, it solved the common problem of overlapping agents in particle systems. Also, his model especially focused on 'pushing' behaviour

and 'squeezing' between people. However, to the best of my knowledge, there are no bionic animations attached to the human crowd.

Rahul et al. proposed their *Aggregate Dynamics for Dense Crowd Simulation* [26]. This approach implements a standard human ratio model controlled by fluid dynamics. It features scalable sizes [36] and adjustable densities. The agent is also rendered in high-level graphics. Rahuls model can manage to simulate a large-scale crowd moving around in one designated area. However, in his approach, agents are produced by one $3D$ male prototype with fewer individual differences in the simulation process.

In contrast, our method includes agents with gender differences and individual body types. Those multi-factorial agents are produced by the DAGM. It can raise the diversity of crowd composition in the rendering aspect. The benefit from it is that we can automatically create a crowd with non-similar agents. With this function we can have more individualized modifications on every agent's identifications. In addition, we could use the individuality for the idea of creating a multi-factorial decision-making model that takes the individual attributes as mobility factors. For example, an athlete usually is stronger than an average person after professional training. Also, age could be another factor affecting strength-level; young people usually have more power than elders. Those could be decisive factors influencing an individuals chances of escape from an emergency scene. Thus all those physical attributes will combine to create an integrated system to decide agent's moving behaviour. Based on that, agents in our model could present more individualized decisions when moving and making other choices.

2.3 Environment Generating Method & Summary

Besides the dynamic method and rendering method in a crowd simulation model, some other aspects that related to our model are also worth to mention. In our cases, urban planning and surrounding system are designed to serve emergency crowd evacuation agenda [17]. A visualizing method [37] has generated a village environment at the

real-time. However, as the best we know from the authors' paper, the village is pre-modelled and not transformable. In contrast to our model, we can control the building number and generating area at realtime. Hence, our model could satisfy multiple environment requirements; making changes at the simulation time is also possible. We also designed some functions, which do not typically exist in other crowd simulation models, for example, a control system for run-time manipulations and the stampede module for the crisis training.

In summary, there are plenty of topics inside the agent-based crowd simulation subject, for instance, the reachable density-level, the capability of population size, the applied dynamic method, and the visualization model type. Thus, most studies generally focused to make improvements on the particular one or two topics. In our case, we attempt to improve the visual-variety of $3D$ rendered crowd and the extensibility of simulation system.

CHAPTER 3

Proposed Methodologies

In this chapter, we present our thesis methodologies for the approaches that we used to achieve the objectives in Chapter 1.

Our thesis involves the design and study of an operational real-time Agent-based crowd simulation model. Comparing our model with previous agent-based crowd simulation models, the major contributions are the diversity of creating crowd and environments, and the operability of simulation processing. The actual methods to achieve those advantages are explained in later paragraphs. However, a comprehensive workflow can help to understand our thesis' idea better.

Figure.4 illustrates the complete workflow of our model. Each sub-box in this workflow is our original work and the software we labeled in the workflow only provides the working environment. We divide the working section into three main parts based on the developing platforms that we use. First, the preparation works section, this part includes character modelling for the DAGM requirements, rigging the skeleton for characters and creating animations based on the skeleton joints transformation. All of that work is done in Autodesk Maya, a commercial modelling software. After that, we import the finished $3D$ models to the *Unity* game engine as part of development assets. The work inside the *Unity* software is closely related to the programming in the *MonoDevelop*. The *MonoDevelop* is an Integrated Development Environment (IDE) that supports the scripting in the *Unity*. It provides the foundational environment for us to implement many simulation functions, hence, our thesis' methods can be primarily built through programming. Our thesis ultimately presents an independent software that can run on any personal computer (PC) with Microsoft Windows or

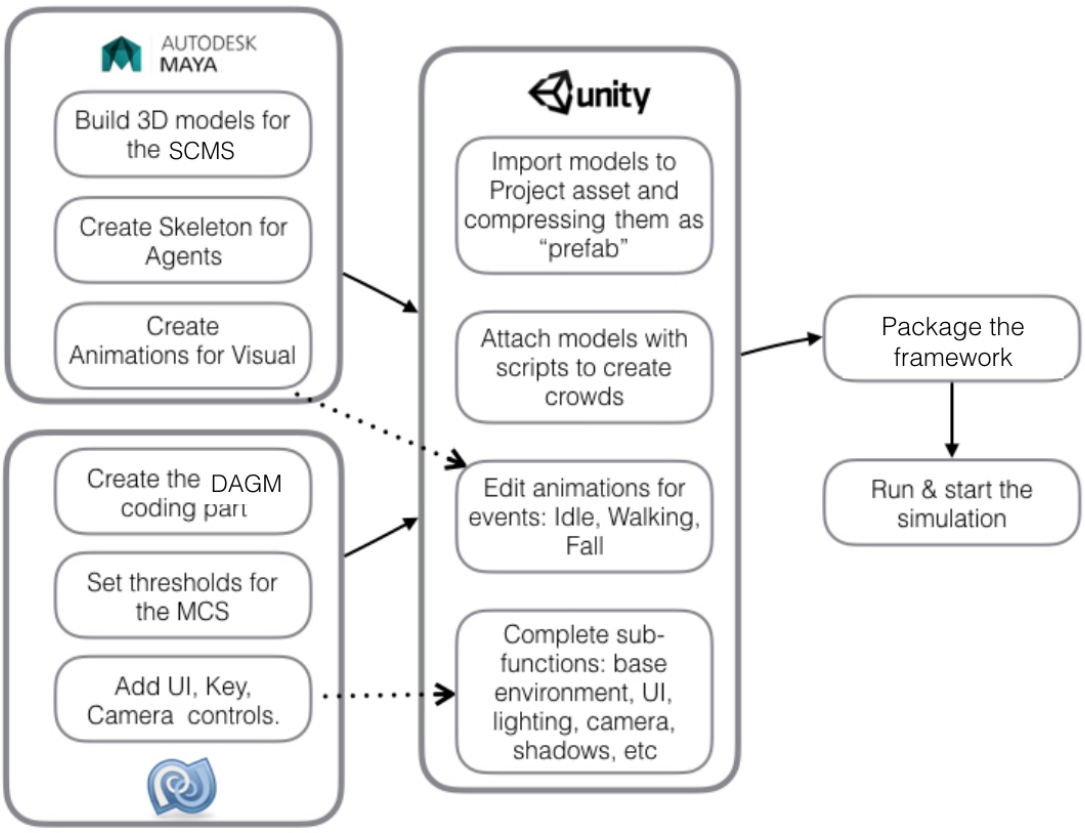


Fig. 4: Our Agent-based crowd simulation model workflow

macOS operating systems, but it also can be ported to other operating platforms.

3.1 Distinguishable Agents Generating Method

3.1.1 Method Overview

Our DAGM is a multi-functional crowd generating system. This system is designed to generate a number of agents at once with individual differences, which means all agents have several distinguishable appearance-factors that provide us a chance to make modifications. The generating function can be recalled unlimited times until the program reaches the computational limits. It means the crowd size could be expanded during the simulation process. Furthermore, all agents are supported by realistic physical interactions between agents themselves and also with surroundings. We have developed several sub-functions to assemble the entire DAGM as an integrated

method. First, we design and implement a new modelling method to create the male and female characters; it features generating agents of different body types and visual uniforms. Then we arrange it with our colour rendering method to give random body, clothing and hair colours. Also, our DAGM has achieved several crowd attribute control functions as:

1. Population control: takes an input value as number of people in the crowd.

2. Agent density control: capable of generating crowds with low-density or high-density.

3. Agent gender ratio: male and female agents ratio in one crowd.

3.1.2 Separating Components Modelling Rule

We define the Separating Components Modelling Rule (SCMR) as a rule to create a modifiable $3D$ human model without manual adjustment. The character that was created under our SCMR can use fewer polygon meshes compared with normal modelling methods, which means less computing time required. We created a group of male and female model prototypes under the SCMR. Each character contains five disconnected body parts: hair, head, torso, legs and arms, and two clothes, shirt and trousers for male, skirt for female. The reason of using SCMR is due to the two points below:

1. Detail Modifications: each agent's parts can be re-scaled and painted individually, because of this feature our agents can have random colour and size differences. We introduce this part in the next two sections.

2. Saving computational power: we want to decrease the 'PolyMesh' numbers on each agent to reduce the graphics power needed. It can improve the overall simulation performance. Our models' meshes are in a separated form without the connection part, thus, the agent has fewer polygon count than a standard modeled human agent.

Those separated model parts are used to assemble agents in the later process, as the ultimate expectation on our DAGM is to auto-generate a crowd with most

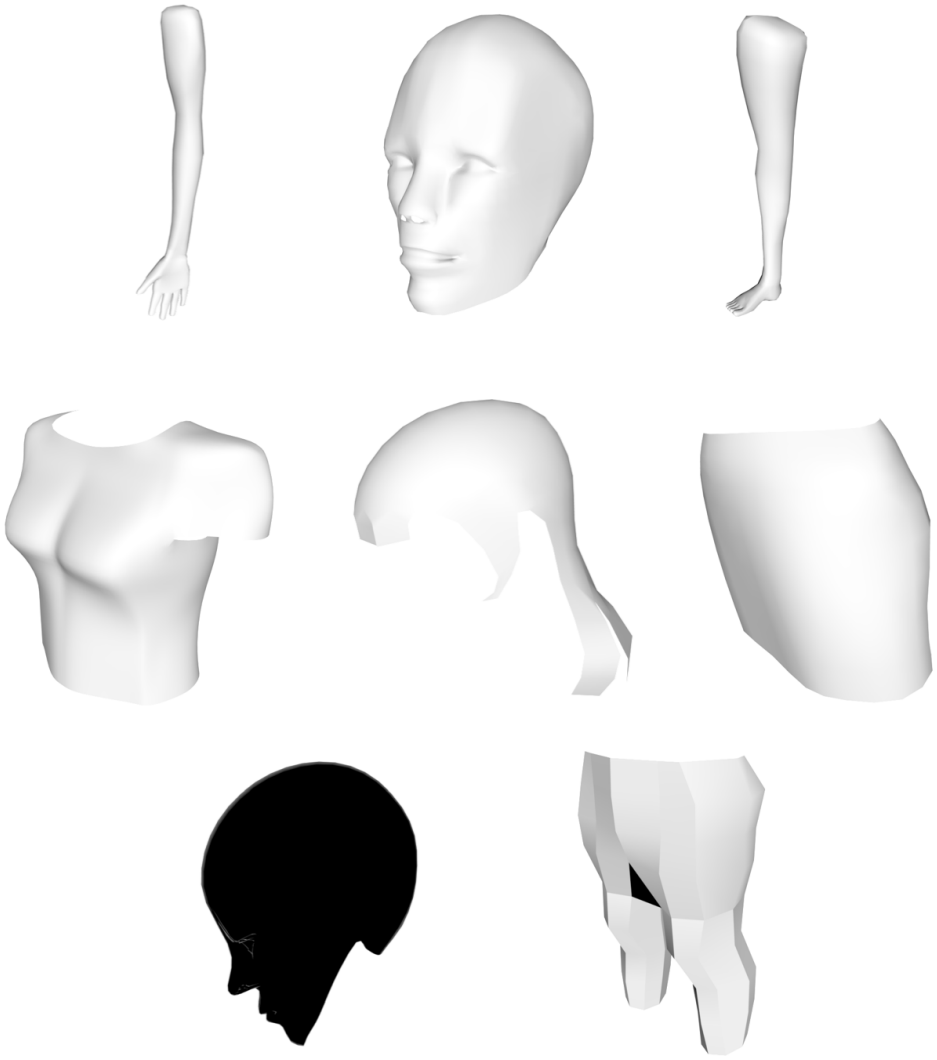


Fig. 5: Agent modelling samples under SCMR

visually distinguishable agents. Thus, each agent in a crowd has dissimilar body styles without the need of loading different resources.

Figure.5 shows a set of character modelling samples under the SCMR, which are in a preview-mode before importing to the developing environment. From top left to right: female arm, female head and female leg, from middle left to right: female shirt, female hair and female skirt, from bottom left to right: male head and male trousers. In this preview mode, we can notice that most of our modelling resources kept in a good and smooth quality. However, some male models have several exporting issues. First, the male head mesh is not textured correctly and the reason for this is due

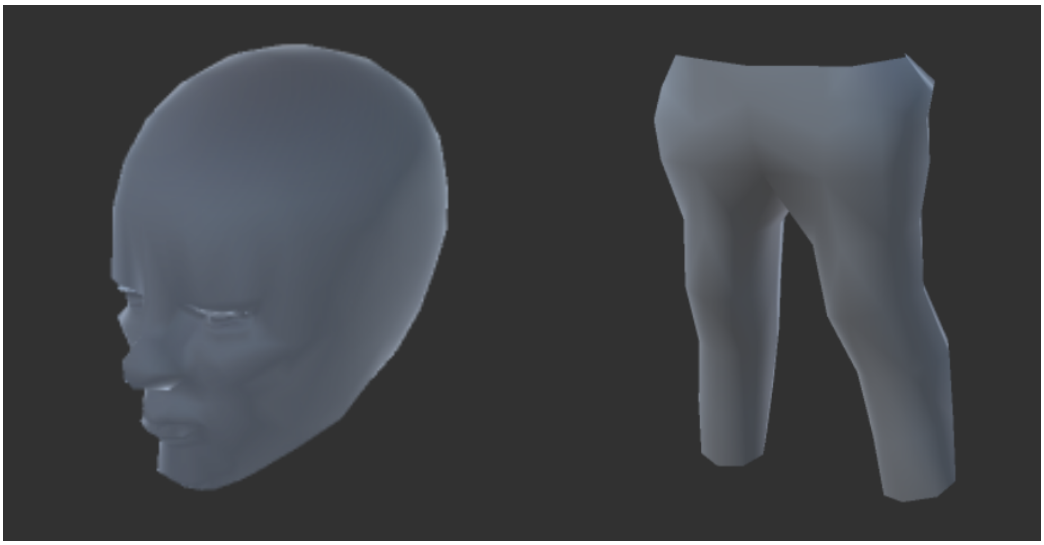


Fig. 6: The result of the adjusted male head with proper texture assigned and smoothed male pants model

to the reversed mesh's normals. Second, the trousers for the male model are in an un-smooth condition and not qualified for the high-quality rendering. Although those issues commonly existed when transplanting models from one software to another; it still affects the overall quality of the visual experiment. However, we solved those problems by correcting the mesh normals of the male head and adding more smooth angles for the trousers model. The fixed models are shown in Figure.6.

3.1.3 Random Colour Rendering System

To give the diversity on clothing variation and body appearances, we divide each agent into four major sections: skin colour, hair colour and two clothing colours. A normal method of giving $3D$ models a random colour texture could be done by altering the numeric representations of RGB triplet value, by providing random numeric value to the specific red, green or blue colour. However, our method is more straightforward; we created three colour gradient controllers, and each of them contains a different range of hue degrees. The colour range is re-definable in the developing environment. The main advantage of this hue system is that we can select one colour by adjusting a single number instead of three in the RGB system. Thus, with the application of random-number generator, our agent can have different colours that are automatically

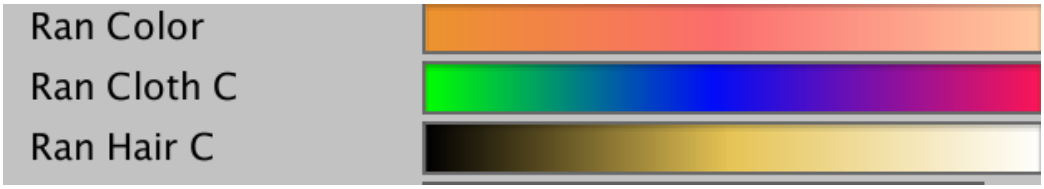


Fig. 7: Three colour gradient controllers in our simulation system

selected. Figure.7 is an illustration of our hue colour controllers. The 'Ran Color' is for the skin colour section. The 'Ran Cloth C' is to provide the colour range for each clothing colour on one agent. Last, the 'Ran Hair C' can assign a random hair colour to our agents.

In order to render realistic and distinguishable agents, we selected the hue range for each colour controller according to the principle of avoiding similar colour tones. It is hard to visually identify the clothing and skin on one agent if the both colours are similar.

We assigned a random selection function to pick colours within the spectrum in each gradient controller. To realistically render our agents, different body sections should have preferred colour choices based on the reality, for instance, skin colour should have the spectrum from white to yellow and black for race diversity. This function helps our agents look close to a real human in terms of appearance. With the wide colour range support, our simulation can avoid the issue of agents rendered in similar colours without identifiable features. As in our observation, we can quickly identify a single agent in a crowd without difficulty.

Figure.8 shows two male characters that were auto-generated under our random colour system. The left man is rendered in a lighter skin colour tone and wearing a violet shirt with blue trousers and comparing with the right male has darker skin colour and clothing colour is different. Figure.9 is a same type of example of two female characters colour comparison.

Fig. 8: Two automatically generated male agents with different colour combinations

Fig. 9: Two automatically generated female agents with different colour combinations

3.1.4 Random Character Size System

Another objective of our DAGM is to provide body size variation on one agent. To expand the diversity, we want to control one agent with several controllable body-part sections and they are head size, arm size, torso size and leg size and we have attached the clothing size on the torso size because we consider the cloth should generally cover the body parts to make it sensible. By assembling body-parts with individual variations, each agent can automatically have a wild range of changes on body parameters in our simulation. Thus, a crowd can have an unique composition of various agents without any manual settings.

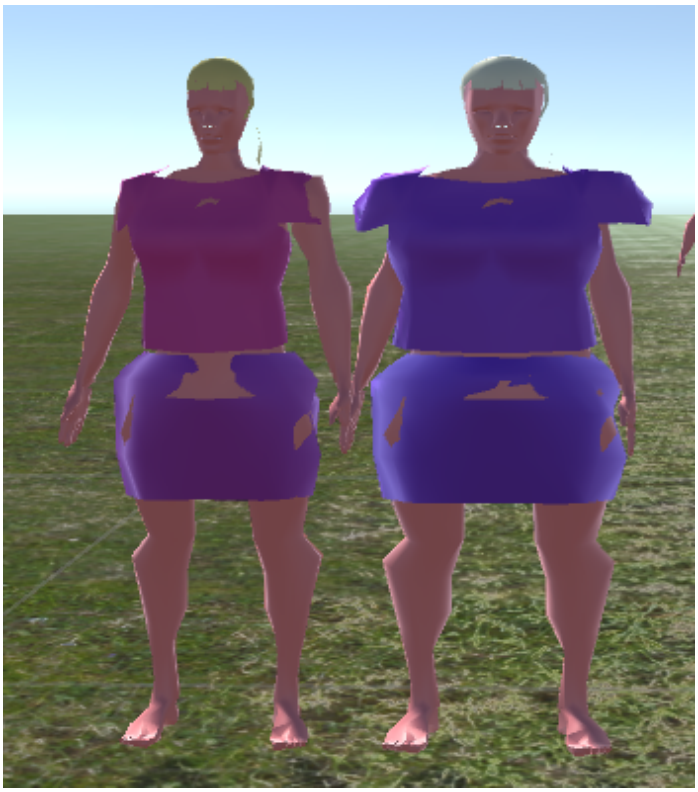


Fig. 10: Two female characters generated in our simulation with size differences

Figure.10 is an example of our random body-size system implemented on two female characters. We can directly observe that left female is slim and right one has an obviously larger body size.

3.1.5 Crowd Spread-level & Gender Ratio Control

Our spread control function and gender ratio function are designed for supplying various simulation needs on the crowd control stage. The crowd spread control adjusts the spread-level for one pending to be generated crowd. The gender ratio adjusts the proportion of male and female characters in one crowd after the total population is specified.

The way to control the spread-level of one crowd in our approach is to enlarge or narrow the possible position that every agent could be generated on. As we built our model in a 3D environment, that commonly use x, y and z as the coordinate system parameters, So we want to have control on the $x \,\&\, z$ value for the agent position because the y represents the height and it should equivalent to the agent height to make the agent feet on the ground. Therefore, the spread function is as follows:

$$a = random(1, \frac{S}{D}) \tag{1}$$

$$x = (a - a') \tag{2}$$

$$z = (a - a") \tag{3}$$

Where a is a random value for us to control the agent positions, the a' and a" are two duplications of a, but the random value is singly assigned when creating. The random range of a is from 1 to $\frac{S}{D}$ where S is the space size, and it is an input value assigned by the user. D is the spread-level parameter and its assigned value from 1 to 5; the higher numerical value, the higher density-level on the crowd. Thus, we can decide the possible positions for the agent by giving a range of values to the spread control parameter. In our case, we can have a crowd in the constant space size but with maximum five times changes on spread-level. After the random parameter is calculated, the position parameter x and z could thus be calculated as in Equation.2 and Equation.3.

Figure.11 shows two groups of twenty (the entire generated amount, some may not capture by the view-camera) agents generated under same space size (100ft) but

with the different spread-level settings. The lower group has spread out more than the upper group from the visual results.

Fig. 11: The spread-level changes while gender ratio, space size and population remain the same

The second operating method is to control the character gender proportion; the control function is defined as:

$$M = \lfloor \frac{A}{R} \rfloor \tag{4}$$

$$F = A - M \tag{5}$$

$$C = M + F \tag{6}$$

Where A is an integer number as the user input and it represents the desired crowd-size by the user. M is the desired male agent number, and F is the desired female agent number. C is the total agent number that sums male and female agents' number after the gender ratio control. R is the ratio value, which is assigned to influence the crowd ratio. We set the value range from $0.01f$ to $1.0f$, f means our function can receive the fractional part of one real number in here. In our function, we positively control the male character number by dividing the total agent number by the ratio value and flooring the result. Then we assign this value as the male character number in the simulation, and we can calculate the rest value as the female character number.

Figure.12 shows the gender ratio controlling result in two simulations, such that upper scene contains more female characters than male, conversely, the other one has male numbers that outnumber than females. With the gender ratio controlling panel displayed at the bottom right in each case.

Fig. 12: The gender ratio changes while the spread-level, space size and population remain the same

Those two functions that we just discussed are built in *Monodevlop* and using the $C^{\#}$ programming language.

3.1.6 Summary

In this section, we have demonstrated our DAGM and all sub-functions that included to support this method. Overall, our DAGM builds the foundation of our crowd system both on generating and manipulating. The DAGM features several unique functions of a crowd generating method. On the individual level, it can create random size and colour agents where we made the base model resources under the separating components modelling rule. On the crowd level, the generating method could be called multiple times to enlarge to the crowd size. We can use this feature to alter on crowd's specifications at each enlarging time, for example, assigning new crowd size, density-level and gender ratio to differentiate the simulation results.

3.2 Multi-layer Collision System

Our Multi-layer Collision System has combined both collision-detection function and physical-collision reaction system together. It can present a comprehensive system that is qualified to support a modern designed agent-based crowd simulation model on realistic dynamic interactions. Furthermore, we have implemented an evaluation function into MCS to estimate the information of both predictable collisions and

actual collisions. Thus, we can show the statistics of the risk-level around one agent at realtime by summarizing the information that processed by the evaluation function, whereas the principle of our MCS is explained in the following paragraph.

In the beginning, we first inspected on the real human anatomy because we want to know that the scale of the visual field a person in a crowded circumstance. A normal persons visual field has 70 degrees to 80 [16] in horizontal meridian. However, we can gain a much wider field of view by twisting our neck and waist which extended the visual field to nearly 360 degrees. Thus we can presume that people have a full circle-perception of the environment in the standard situation. Moreover, the distance between viewer and obstacles decides the warning-level in the viewer's 'mind'. For example, an object that is far away from the viewer does not cause much attention to the viewer, although the viewer maybe aware of the object existence. However, an obstacle that is close to the viewer, may lead the viewer to make an immediate decision to avoid it. This theory is the basis of our principle idea of the MCS.

Figure.13 is a sample implementation of MCS inside the *Unity* environment. The agent positioning at the center has three layers to detect and react any collisions:

1. The outside 'Collider' which has a square shape and has the widest range to detect the objects appearing.

2. The inner 'Collider' has a sphere shape and works for a narrower range to detect the short-range objects with potential of crushing.

3. The core 'Collider' is a capsule shape collision detector that just cover the $3D$ agent body to react the physic interactions with obstacles.

The details of our multi-layer function is discussed in the next section.

3.2.1 Model Overview

Our collision system is primarily based on the vision approach by Ondrej et al. [28] where they presented a model of agents deciding their moves by a triangle shape of the vision field. However, we consider that limiting the humans vision to a triangle

Fig. 13: An example of the MSC implementation on one agent

is over restricted. Moreover, walkers can judge the surrounding obstacles by the distance and this is the basic idea of our multi-layer collision system. As we show in Figure.14, one agent in our model has three layers to detect the imminent collision and auto-evaluate the risky level about current surroundings:

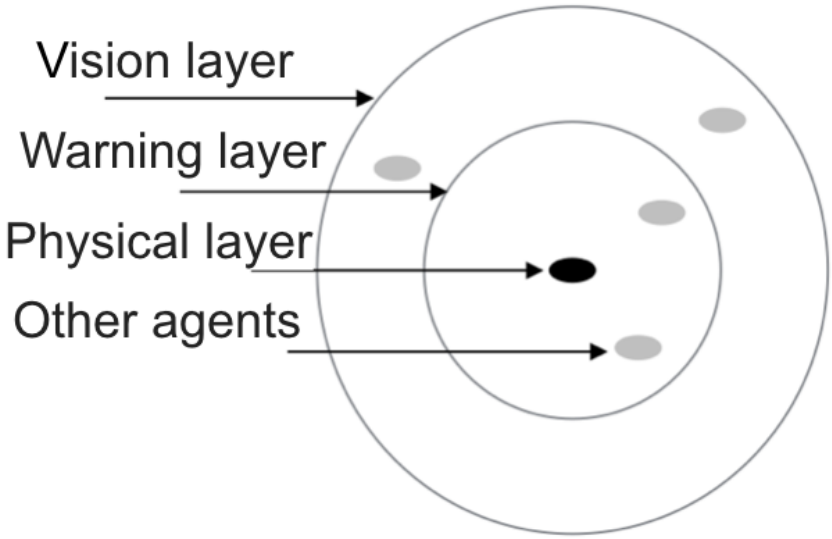


Fig. 14: The Mutli Layer Collision System

1. The vision layer: it represents the agent's maximum visual distance in our model. Obstacles or other agents appeared in this layer are recorded but it is inferior than the information in other layers.

2. The warning layer: this layer gives the information about obstacles that are going to collide with the agent; if there is no detour or slow down actions that are taken by the agent. It requires the agent pays more attentions to avoid.

3. The physical layer: this layer represents the actual physical body of one character. Any information retrieved from this layer means that a collision has already happened. In the case of multiple agents crashing happened, there is a possibility of falling. Agents fall due to multiple 'competing forces' which exceed the individual durability on outside forces; and this durability is decided by the evaluation function and the falling decision algorithm.

Table 1: Density evaluation base on MCS

Threshold	Density
$E \leq 10$	Low-density
$1<E \leq 100$	Medium-density
$10<E \leq 1000$	Dense
$1000<E$	Overdense

3.2.2 Collision Evaluation Function

In our model, once a agent received the 'visual' information of surrounding, the MCS can synchronize all those information into a data set which is updated by each frame while the program running. The evaluation function can dynamically detect the enter and exit of other agents from each layer. The method we transfer this 'visual' information to a numerical data set is by Equation.7 below:

$$E = \sum_{i=0}^{n} Vi + \sum_{i=0}^{n} Wi + \sum_{i=0}^{n} Pi \tag{7}$$

This Equation.7 sums all 'visual' information that received from one agent's MCS and process it as a numerical feedback information. We compute the evaluation value E from the summation of source layers. We make Vision, Warning and Physical as three parameters to divide the information according which layer it comes from; and they contain the weight of each three layers: the vision, the warning and the physical where $V < W << P$. i as the obstacle amount indicator inside each layer, in this case, the number of other agents inside of each layer. Thus we could calculate the emergency level of one's surrounding based on this weighted value.

After the value processed from Equation.7, we set several thresholds to evaluate the current density-level of one agent's surrounding and the results are shown in Table.1. All data in Table.1 is based on the parameters' value that are preset in our program as $V = 1, W = 10, P = 100$. Note that the value of all parameter is just for dividing the classified results, therefore they do not have the actual meaning.

However, we can adjust the parameter value to achieve different dynamic results where those parameters become factors for the stampede simulation study in next section.

3.2.3 Stampede Thresholds

We could formulate our stampede scenario with the benefit from our MCS's evaluation system which introduced in the last section. An individual tends to fall if there are several forces 'pushing' him [her] from one or several directions. This case in our model is that any contact from the physical layer is a pushing force to one specific agent. This agent could endure the forces with a certain amount of durability. Besides, the durability varies on the individual attributes. Thus, those principles make the foundation of our stampede simulation scheme. We first made an assumption that each agent has some amount of durability, and the amount is decided by gender, body size and current velocity. The durability can be considered as the thresholds of falling. When the pushing force over the durability, the agent has the possibility to fall. By the conditions above, we can form the equation for falling simulation as:

$$\mathcal{F} = \mathcal{D} - \sum_{i=0}^{n} Pi \tag{8}$$

The interpretation of Equation.8 is as follows. The collapse indicator $\mathcal{F}$ is determined by the threshold value $\mathcal{D}$, which each individual has their own calculated value in the simulation as the characteristic attributes. When the total forces $\sum_{i=0}^{n} Pi$ is over the threshold $\mathcal{D}$, then the equation returns a negative value on $\mathcal{F}$, which means the agent is in a possible-to-fall condition. However, it is not a certainty condition that once $\mathcal{F}$ meets the threshold, and the agent should fall. We consider that $\mathcal{F}$ represents the 'chance' of falling instead of the 'switch' to fall or not. That means after $\mathcal{F}$ meets the condition, the agent's status to fall or sustain that force are decided by our Falling Decision Algorithm as follows:

Algorithm 1 is the method we determine the agent's status of falling or enduring, when the agent current surrounding density is over the density threshold from the

Algorithm 1 Falling Decision Algorithm

Require: Falling chance $\mathcal{F}$, Random Parameter $\mathcal{N}$
  initialize and update $\mathcal{F}$ from equation.8
  and a loop counter $i = 0$
  while Agent condition is not in $Fall$ and the $velocity$>0 **do**
    if $\mathcal{F} \leq 0$ **then**
      $\mathcal{F} \leftarrow |\mathcal{F}|$
      Using pseudorandom number generator to generating a number as the value of $\mathcal{N}$
      if $\mathcal{F} + i > \mathcal{N}$ **then**
        Return agent condition as $Fall$
      else
        $i++$
        return to while
      end if
    else
      return to while
    end if
  end while

evaluation function. As more forces the agent suffering, the calculation result $\mathcal{F}$ also increases; the higher risk he[her] falls. Furthermore, we applied a pseudo-random number generator to produce the limitation trigger. This trigger gives a nonidentical individual endurability on fallen event simulation in our model. The $\mathcal{N}$'s value has the scale from 1 to 10 and we compare $\mathcal{F}$ with $\mathcal{N}$. If $\mathcal{F}$ is greater than $\mathcal{N}$ in one loop, the agent falls. This algorithm gives the principal function to decide the falling condition for any walker in the crowd. After the agents reached the 'pushing forces limit, they has more possibility to fall, when they hold the over limit forces longer. However, the specific limitation value can be modified to catering particular study goals. The time complexity of our Algorithm 1 is o($\mathcal{F}$).

3.2.4 Summary

Our MCS is an adequate tool when we target to analyze one agent's surrounding in a crowd. This system gives an instant feedback about both happened collisions and foreseeable collisions at the current running frame. By analyzing feedback for every agent in the crowd, we could form a bar chart like in Figure.15. It shows two groups of

people split by different density-levels. We can easily observe that people in $Crowd\,1$ share a higher density-level situation than $Crowd\,2$ because $Corwd\,1$ has more people in the dense and overdense bar. This kind of information that accumulated by MCS can help us to understand a visually-complex human crowd by a highly summarized data set.

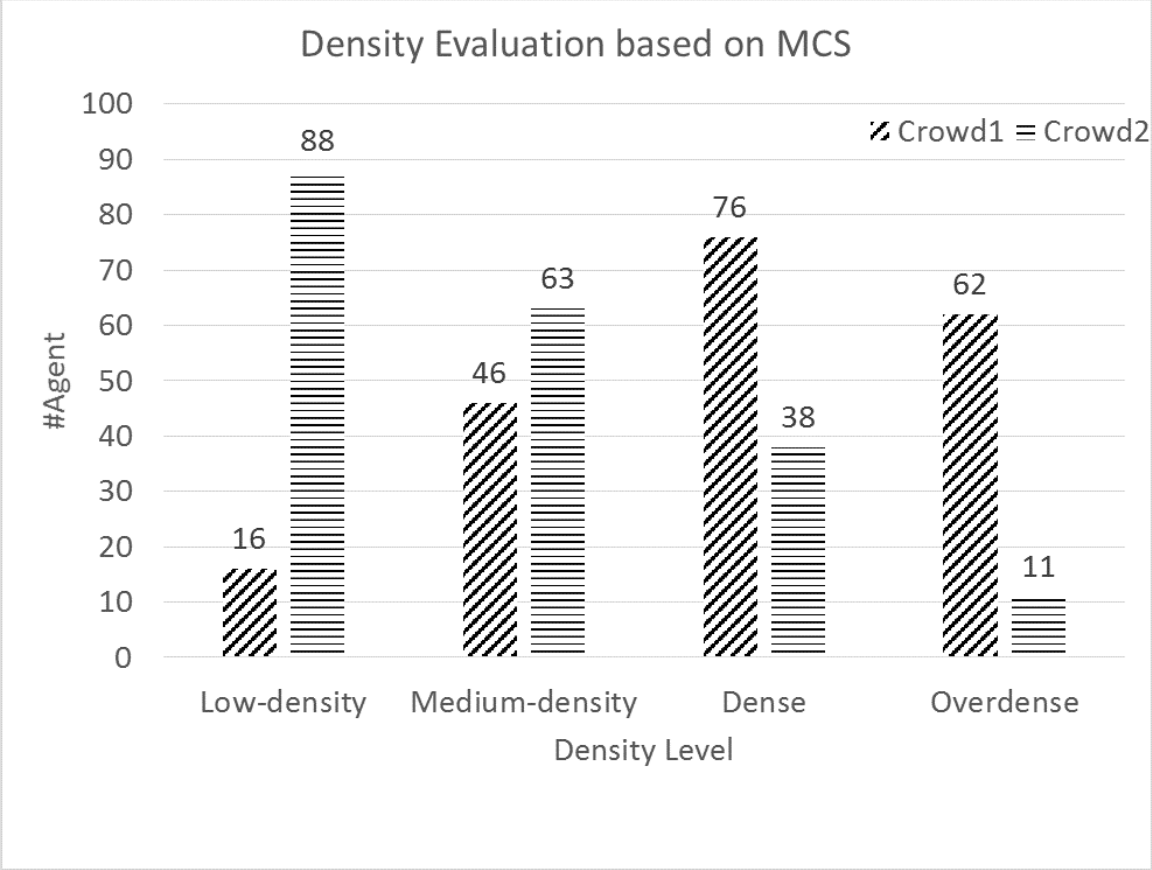


Fig. 15: A density evaluation chart showing the analysis on two crowds density spread-level under the same space size

3.3 Building & city planning Generating System

Besides our model generating method, the DAGM, and our dynamic system, the MCS, we have built an environment & obstacles system for our ACS system, Building & City planning Generating System (BCGS). This system is designed for providing the basic environment of the evacuation scenario, which is highly demanded as recently

terror attacks happened more frequently in Europe cities and worldwide.

3.3.1 Model Overview

The BCGS contains two generating steps:

1. City planning step: at this step, the BCGS divide the ground into two sections: the 'building' section and the 'crowd' section. One simulation map can contain multiple 'building' and 'crowd' sections in it but each unit area should be assigned by one of this two sections to avoid the undefined problem. After this compartmentalization step, all agent can only be generated on the land marked as 'crowd' section, and all 'building' section would become obstacles to crowds.

2. Building generating step: the second step of our BCGS is creating buildings within the 'building' area. For the convenience purpose, buildings can be auto-generated by user's control. Moreover, this step can be called unlimited times during the simulation to make modifications on buildings' amount, positions and sizes.

After this two steps, the general ground has clearly defined boundaries that prohibited agents generated into. Also, the buildings has restricted the agent's movement. In certain conditions, those obstacles can change the agent's navigation path due to the avoidance function in our navigation system.

3.3.2 City Planning Step

Before the city planning step, our agent can walk around on the entire ground that has defined map size without limitation. Therefore, the default ground is a 'crowd' section originally. What we need to do is adding the 'building' section boundaries above the ground. We consider that with different building section boundaries applied, this system can generate a street block, a residence area, or an urban downtown area by modifying on the blueprints. Figure.16 provides a better explanation of our idea about the city-scheme method. The coloured boxes on the Manhattan map are equal

Fig. 16: A piece of the city-map of Manhattan

to the 'Building' areas in our model. Those sections together formed the streets in a city as the way we construct the 'crowd' section in our model. Our building section can alter the size and shape for different city arrangements. The size editing can not only provide more space to place buildings in it but also oppositely changes the 'street' width from wide to narrow. By enlarging the building section's size, the street is narrower and vice versa. Editing on the geometry shape of each section can provide the stylish design for the urban planning. We have implemented a cross street sample with four building sections and an intersection in the centre. Figure.17 is a simplified $2D$ map of the street block framework in our simulation. Giving more explanation on this, the circle represents the map size and any position inside the circle area can have crowds generated on it, except the grey squares which represent the building areas and buildings are generated inside them. All agents still can move into the grey section after they start the navigation process.

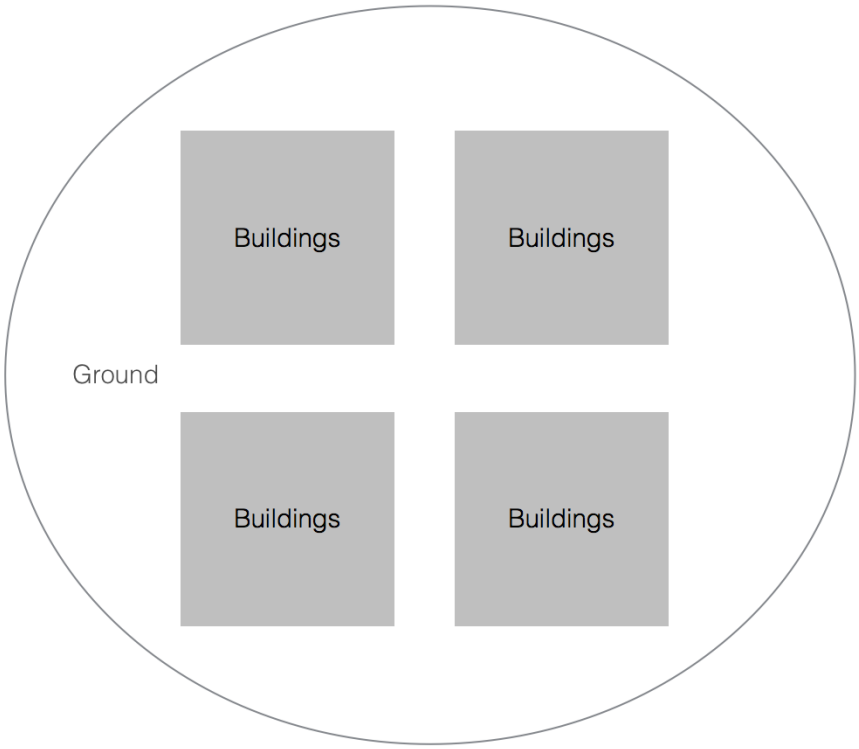


Fig. 17: A 2D map sample of our city scheme design

3.3.3 Building Generating Step

This step is a subsequence step after the city planning step finished. Due to the buildings can only be placed after the city section is specified, and apparently these two steps are nonreversible. The main point of this step is, after the city scheme is constructed, we can to fill it with 3D buildings for placing obstacles and the urban parameter performance testing.

In our model, all buildings are represented by shaded rectangles with the same texture applied. However, to avoid the duplicate looking, we applied a random function on the building's sizes. So that the buildings could have the differences in height and width between each other, which can also increase the 'city-like' feeling. However, we did not provide more modifications on this function and the main reason for this simplification is in consider of the current geometric needs from our program. We want all sub-functions only reach to the necessary level without spending too many computational resources. Thus, we could optimize our computer power on the crowd geometry performance. However, our BCGS is still capable of producing a realistic environmental scenario with same rendering quality as in our agents.

Our surrounding approach can generate a number of buildings at one time. Thus, there are two issues that need to deal with while we setup this function:

1. Recognition ability, buildings should positively recognize the position whether is in the allow-to-place section before they are going to be assigned.

2. Avoiding overlap between buildings, due to our generating method can create multiple buildings simultaneously, that may cause the polygon shapes across with other models. So we need our algorithm has duplication detection function to avoid this problem.

We have designed this Algorithm 2 by using the duplication testing method. It has the ability to solve both the recognition and overlapping problem.

As the algorithm begins, we need to load an input value for $\mathcal{A}$, which is the number of buildings. $\mathcal{X}$ is assigned with a random three-dimensional coordination, which is generated by the pseudo-random number system. A building is placed in the

Algorithm 2 Building Collisions Detection Algorithm

Require: Building position $\mathcal{X}$, Buildings amount $\mathcal{A}$, $i = 0$
  System generate a new position $\mathcal{X}$ automatically, if previous one failed.
  for $i<\mathcal{A}$ **do**
    if Check $\mathcal{X}$ is in building section **then**
      if $\mathcal{X}$ = any value in the array $[\mathcal{X}]$ **then**
        Request a new position $\mathcal{X}$
      else
        Place the building
        Store $\mathcal{X}$ to the array $[\mathcal{X}]$
        i++
      end if
    else
      Request a new position $\mathcal{X}$
    end if
  end for

simulation environment after satisfied our algorithm examination. Its $\mathcal{X}$, which is the position information is stored in the array of $\mathcal{X}$ as $[\mathcal{X}]$ for later duplication checking process. Therefore our method can create non-overlapping buildings inside the fixed area. The time complexity of Algorithm 2 is o($\mathcal{A}$).

The buildings can interact with crowds as solid obstacles that have the function to block all agents to cross or access into it. If an agent is blocked by a building, he[she] can only take a detour instead cross from it. We also provided a 'refresh' function for the buildings' establishing where old buildings are replaced by new if this function is recalled during the simulation.

3.3.4 Performances

We implement a cross street scene by using our BCGS. Once the space size is specified by the user, four 'anchors' are created around each corner of the rectangle ground. Anchors' positions are according to the fixed space size.

Figure.18 illustrates the procedures of implementing the environment by using BCGS. The building section's boundary appears as light green edged cubes. We can observe that some buildings are partially out of the boundary and that is due to the pivot of one building is set at the centre. However, all buildings have same

blocking function as a wall. After the simulation begins, the agent can walk into the boundary but cannot cross through a building. The building's variation on sizes is showed in the right bottom picture of Figure.18, with this feature applied, the building function provide more diverse looks of the modelled urban environment for program observations.

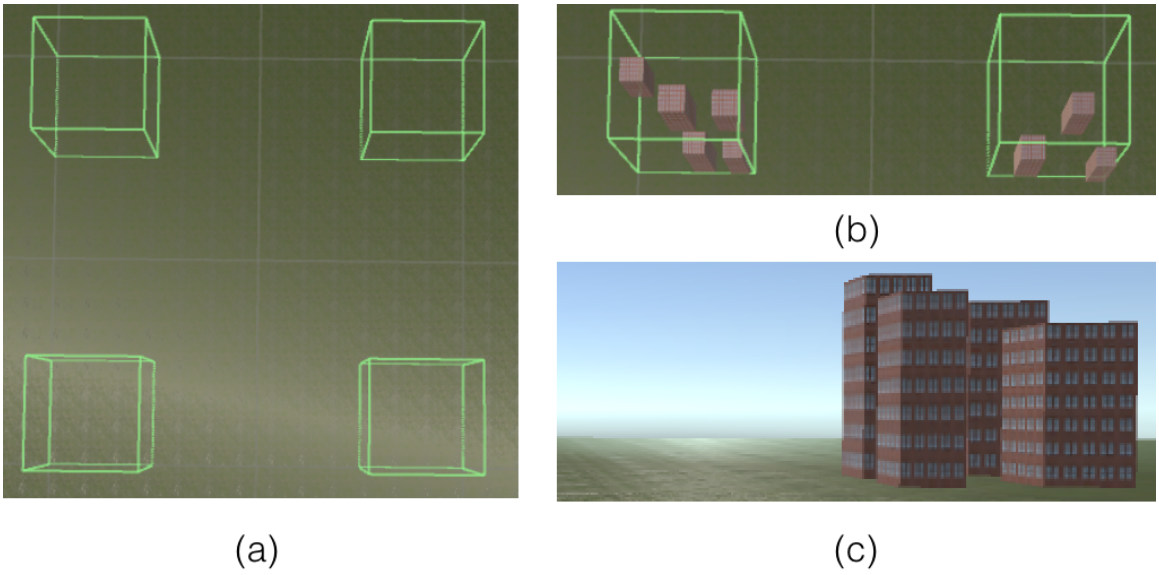


Fig. 18: The BCGS showcase and rendering results.(a) Four anchors placed on the ground with building sections' boundaries, (b) generated buildings without the overlap issue, (c) the perspective view of buildings through the in-game camera

CHAPTER 4

Implementations

In this chapter, we first provide information about the working environment of our crowd simulation program. Next, we give a summary of the in-simulation operating methods and additional system features that we designed for supporting the simulation. After those, we present a step-by-step simulation sample as the user guide of our ACS system, where we simulated a crowd that contains two hundred agents. The experimental results are described in the last section of this chapter.

4.1 Hardware configuration & Software Platforms

We developed and tested our ACS program on a computer with macOS, and its system configuration is 2.7 GHz Intel Core i5 CPU, 8 GB 1867 MHz DDR3 memory, and the Intel Iris Graphics 6100 1536 MB integrated GPU. Our model could generate a crowd with two thousand agents within seconds under the display resolution of 1024 x 768 pixels. Also, we tested our model's performances by using another computer with Windows 10 operating system, and its system configuration is 2.5 GHz Intel Core i7 CPU, 8 GB memory and a NVIDIA GeForce GTX 970m dedicated GPU.

Our program is developed across multiple development platforms:

1. Unity 5.5.1f1 personal (64bit)
2. MonoDevelop 5.9.6
3. AutoDesk Maya 2016 SP5 (Educational version)

4.2 Operational Methods & Animations

Besides the methods we have introduced in Chapter 3, our program also has the controllability features for users through electronic devices like a mouse and keyboard. By interacting with our model through those electronic devices, the gaming experience can be significantly improved. On the other hand, we have created bionic animations for character motions, and animations in the simulation can immensely enhance the realistic feeling on the visualization aspect.

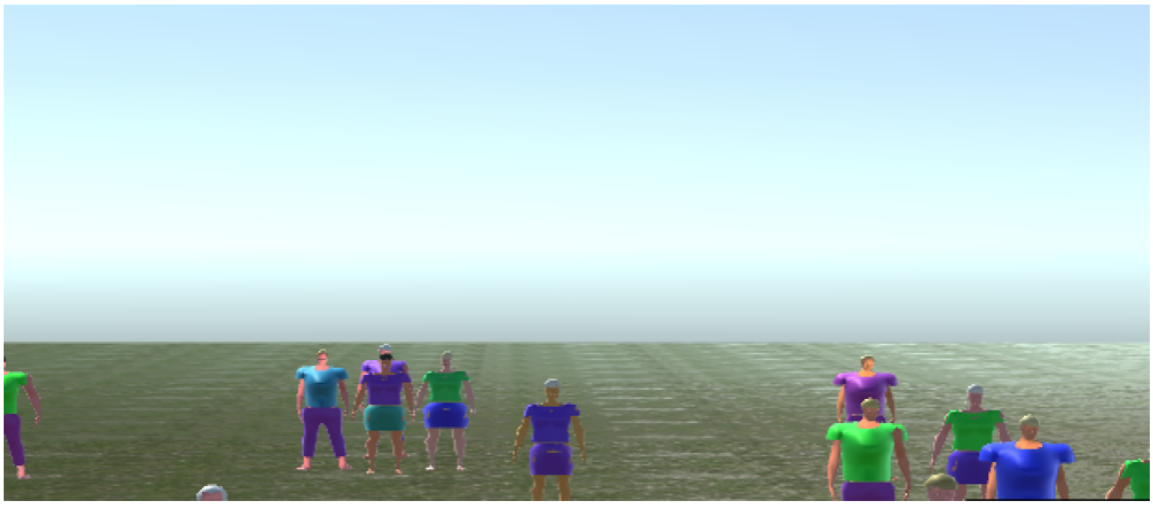
4.2.1 Control Methods

Camera Control: The way to inspect a $3D$ gaming environment is to go through the view camera. At this perspective, we set our camera with a good range of flexibility. The default camera setting has 140^o on the width and 60^o on the height in terms of the field-of-view. The rendering distance of the camera is set at one thousand unit-distance in the $Unity$. This rendering distance can provide sufficient details of all agents without wasting any additional resources. We place the camera ahead of the crowd generating area and shoot with twenty-degree top-down angles for a comfortable viewing angle to catch the most rendered objects in the scene.

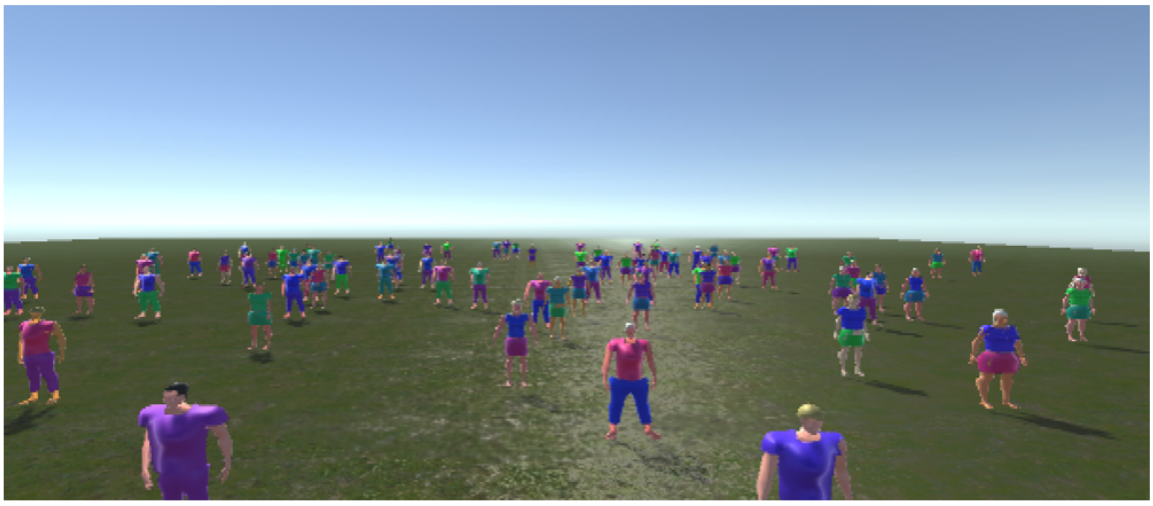
We established some functions to extend the controllability on the camera view; in the case that we want to observe more of the rendered model's details on generated crowds. There are two functions to support that:

1. **Zooming function**: the camera can zoom in or out by scrolling the scroll-wheel on the connected computer mouse while holding the right-click button. The zooming range are not easy to describe by words, thus we have provided the comparison picture in Figure.19.

2. **Rotation function**: the rotation angle of the camera is by tracking the mouse movement that is similar to many first-person shooting games. To avoid the mishandling problem, we need users to move the mouse while holding the right click button to make the camera rotation. Technically, our camera can support



(a)



(b)

Fig. 19: Examples of the camera zooming control function.(a) Zoomed view from our program camera. (b) standard field-of-views from our program camera.

a 360^o full-range view both horizontally and vertically.

Keyboard Control:

We applied several keyboard functions to control different in-simulation events and they are listed below:

1. Key '*R*' and Key '*L*' control the male and female agents navigating system separately. With the associate button pressed, agents in the current scene will start walking towards their target point on the map.

2. Key '*Space*' controls our buildings establishing function. Once '*Space*' is pressed during the simulation, a new group of buildings is created around the building section by our BCGS. However, this function has to be applied after the space size is defined, which is a prerequisite setting.

Mouse Control: Besides supporting zooming and rotating camera view in the program, we have designed an information tracking function. It can acquire individual agent information at the simulation scene. By right clicking on an agent in the scene, we can display the information of the agent's name (we arranged this by gender + number) and his [her] current coordinates on the UI panel.

UI Control: Our crowd simulation system contains panel based user interfaces, which has several interaction functions to control simulation parameters. We listed them below, based on the interaction types:

1. **Input field panel**: this kind of panel can support users to enter any value by a keyboard. As in our case, it is used to define the agent number (population) and space size.

2. **Slide bar panel**: this panel has a slide bar to control the volume of the input value. We use this type of panels to control the spread parameter and gender ratio parameter. Both parameters have one same feature that percentage is more important than the precise value.

3. **Button panel**: this panel has the function to initiate a defined event. We use the buttons to apply the settings we have made in previous panels. To be specific, we have separate buttons to control the crowd generating function and the space generating function.

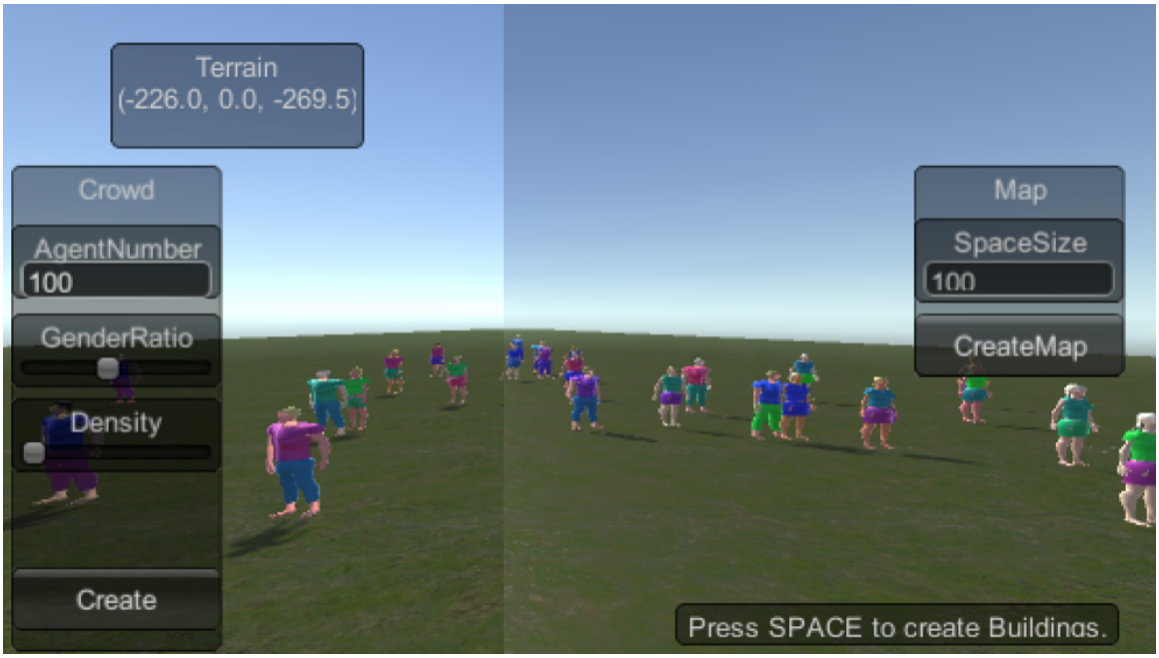


Fig. 20: The in-game UI system overview from the main camera

Figure.20 shows the layout of the UI system in our program. The left side panels group are crowd control panels with AgentNumber input, GenderRatio control, Spread control and the Create button. The one on the top left is the tracking panel, which we have introduced in the last section. On the right side, there are two panels, the upper one is the Map control panel with SpaceSize and CreateMap button. The last one is an instruction notice that tells users how to build buildings in the environment.

4.2.2 Realtime Animations

Providing a realistic simulation process is our primary goal from both agent behaviour aspect and the rendering quality aspect. We want them both as close as possible to real human crowds. On the agent behaviour level, we have MCS to evaluate

the surroundings and provide a sensible decision-making ability. On the rendering quality level, our DAGM can provide high-quality $3D$ characters with random body features. However, we intend to create the animation feature to enhance the visual performance regarding the realism. To achieve that, we crafted several animation clips for different situations on our prototype models. We constructed the animation in $AutodeskMaya$ with its animation system. We imitated the human anatomy to avoid the puppet feeling animations, which is a common issue in other animated crowd simulation studies. The first thing to create bionic animations is to build the 'skeleton' that emulates on human structures. Just like the normal human structure, our agent's skeleton also connected by joints. The skeleton size is adjusted to fit for our agent's size differences. In our case, we built two independent skeletons for male and female characters. Thus, we also need to design animations for two skeleton sets separately. After the skeletons is built, we need to connect them with the agent body. At this step, we painted the 'body weight' for each joint in the skeleton. This step is to define a specific value of how much a joint can control on the 'skin' motions. With properly aligned body weight painting, our agent can present several poses, which are identical to human poses. After this step, our agent's actions can be modified by controlling the positions of skeleton joints. Last, we have created the animations by designing a sequel of motions in the key-frame animation system to produce series of animations for both characters.

To satisfy our simulation requirements, we have created three animation clips for both male and female characters. We named them as 'Idle', 'Walking' and 'Fall'. Because we have made two independent skeletons based on gender differences, so we have six clips of animation in total. Each animation clip has different time, which is indicated at Table.2. Our computer animations' frame rate is placed at 24 frames per second of which we can capture smooth motions of our agent during the animation process.

Animations have different active conditions in our simulation program. To be specific, the 'Idle' animation is played after agents are generated or their movement is stopped. The 'Walking' animation is set to active while the agent starts moving.

Table 2: Animation frames in each clip

	Male	Female
Idle	40 frames	24 frames
Walking	50 frames	40 frames
Falling	27 frames	20 frames

The 'Fall' animation triggers according to the stampede evaluation system, which is introduced in Section 3.2.3. After the agent fallen, his[her] animation system is locked at the last frame of 'Fall' animation. The movability of the agent was frozen too, thus, the agent becomes an obstacle to others.

Figure.21 shows two examples of simulation scenes with animated agents. (a) shows agents walk toward their desired destinations on the map while un-moving agents remain stand. (b) shows one male agent has fallen in the crowd, and other agents try to avoid him or step on his body if avoidance is not possible.

4.3 Program Procedures

Our crowd simulation program has many controllable features, as we have introduced in the previous section. This section gives a step-based operating instruction for one completed simulation procedure. Thus, many control functions can be described in more details.

I : Open the program executable file, the default interface and the $3D$ environment (ground and skybox) show in the program's interface window.

II : Enter the space size, we usually set this at 100, which gives enough space for placing agents. Click the 'Create map' button to apply the ideal map size. Then press 'Space' on the keyboard to create buildings, this part could be repeated multiple times to construct the desired city scene.

III : Enter the ideal agent amount in the 'AgentNumber' panel, then adjust the density and gender ratio settings. Last, click on the 'Create' button after rest

(a)

(b)

Fig. 21: Two animation scenes in our simulation. (a) walking agents in the navigating process, (b) an agent has fallen on the ground due to the high-density.

crowd settings are confirmed. One crowd that related to parameter settings is generated.

IV : After the crowd is placed in the scene, we could observe it from different angles and distance by controlling the camera view. The camera control is set as active during the simulation. Next, we can press 'R' or 'L' to initiate the evacuation simulation on the current crowd.

V : Step III & Step IV can be repeated unlimited times till the simulation result meets the expectation or the computer power is exhausted.

4.4 Experiments

In this section, we provide details of our experiments and discuss the obtained results. The evaluation of an agent-based crowd simulation model can be complex, from the scalability, flexibility to applicability and realism-level. All those factors could be a crucial assessment for one crowd simulation model. Sometimes it is quite frustrating to chase them all at the same time. Typically, a research focuses only on one or two topics to propose the improvements and contributions.

As in our case, we have put our study objective on making improvements at the simulation realism, while other factors still moderately developed. Thus, this section is arranged as follows:

Self Experiment: we first measure our proposed methods performances independently by conducting simulation statistics that is extracted from our program self. Where some factors cannot share the same standard with others work, and the possible reason could be dissimilar of rendering resources, the differences of computer configurations, and etc.

Cross Experiment: We compare our model's features with other proposed models where the same standard could be shared for a comparison, for example, the rendering system and the dynamic methods.

We successfully simulated several crowds with the different population, spread-

level, and gender ratio applied. Due to our program is a real-time simulation approach, we choose the frames per second (FPS) as the performance parameter in our tests. The results are demonstrated in the charts below:

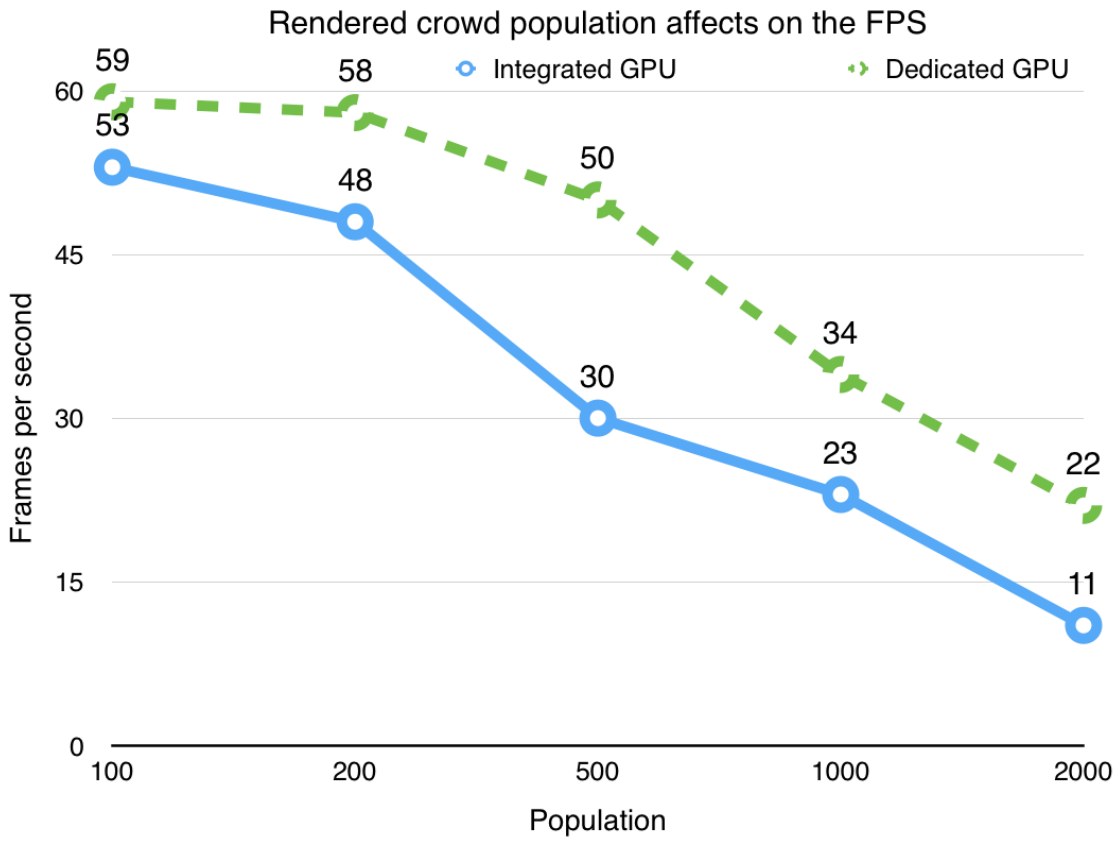


Fig. 22: Crowd population affects on the FPS

The reason for using the FPS as our evaluation parameter other than the generating time is because the FPS can indicate a dynamic performance for a period. However, the generating time is mostly decided by the preprocessing speed which is not crucial in a real-time simulation program. Due to the frame rate value fluctuates during the system running, we decide to take the average frequency value. The maximum frame rate has been locked at 60 FPS in all experiments, because a program with over 60 FPS has no noticeable differences to the observation and it is a waste of computational power.

We used the controlling variables method to obtain the performance of one particular factor in our simulation, then we can measure the impact on our model from

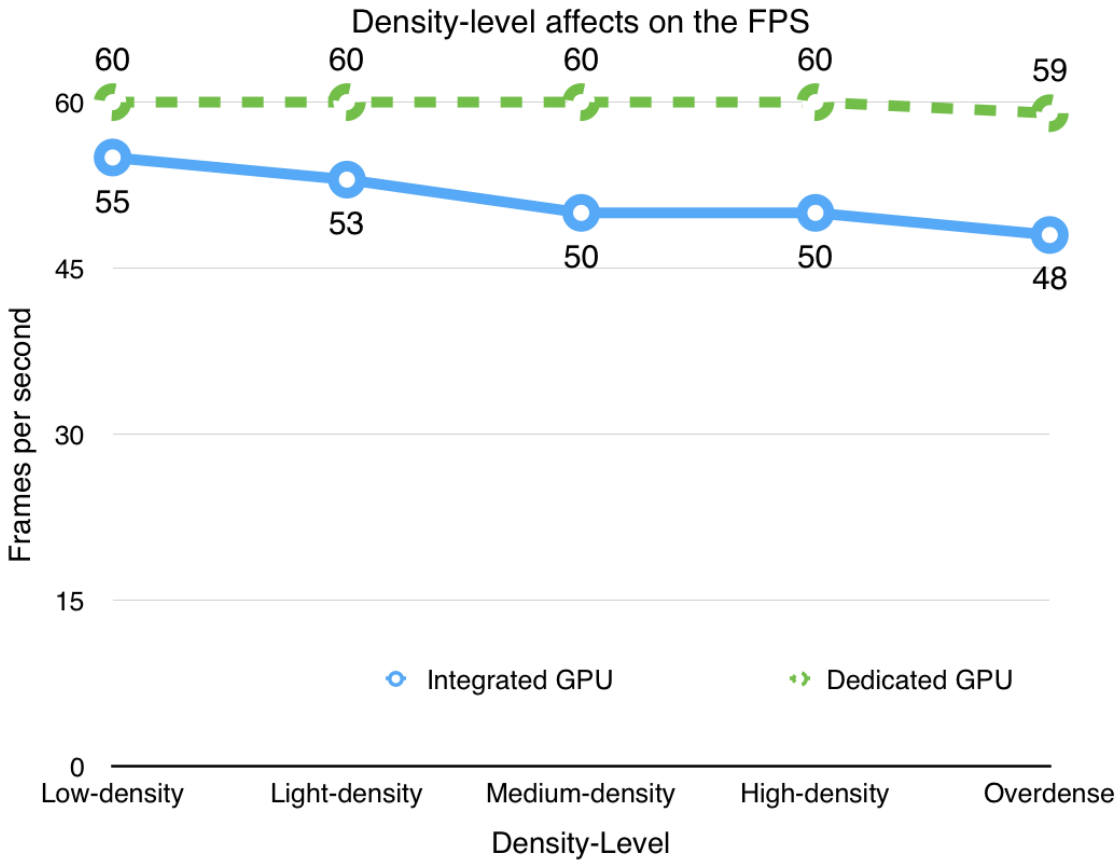


Fig. 23: Density level affects on the FPS

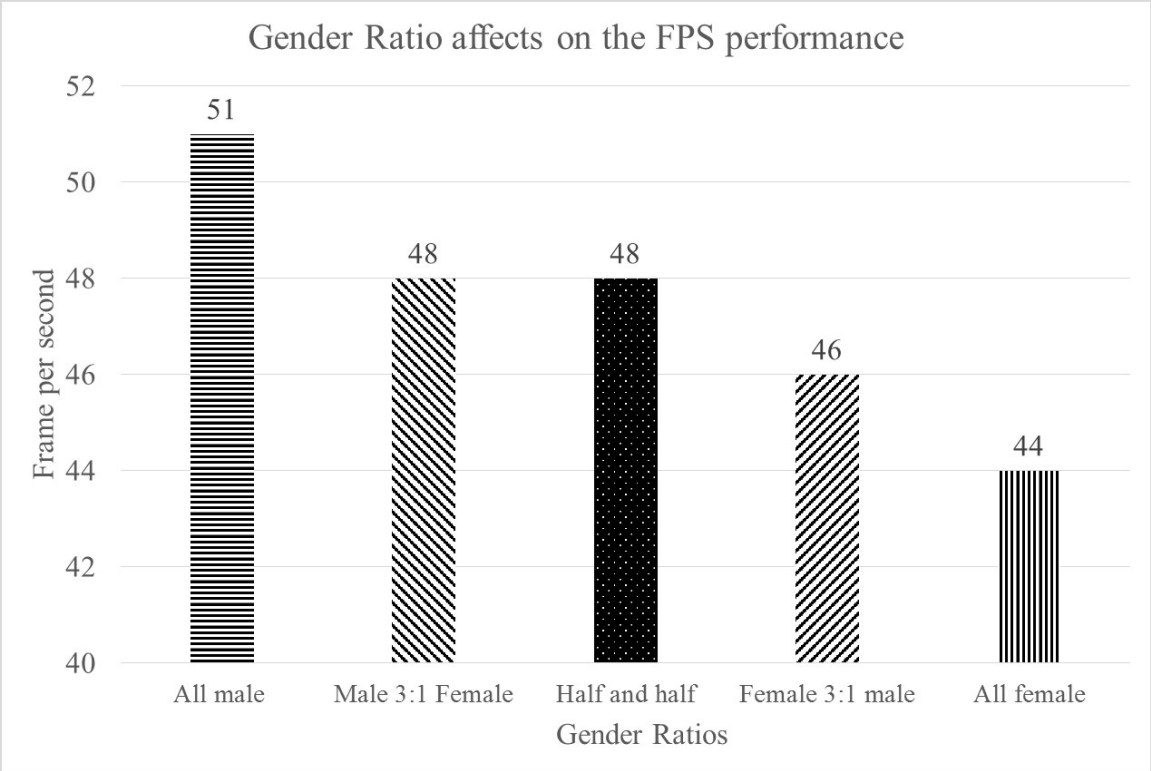


Fig. 24: Gender ratio affects on the FPS

this factor.

Figure.22 shows the frame rate slopes down while the population of rendered crowd increases in both system configurations. The frame rate drops significantly after the crowd size is over five hundred in the computer system with an integrated GPU. We acquired a better performance in the system with a dedicated graphics card, where the frame rate start to drop dramatically after we simulated one thousand agents in the scene. However, we do not set the limitation of the population in our system. Although, it inevitably takes more time to render if we want to expand the population even more. The computational complexity for generating our 3D graphics crowd is $O(c^n)$ where c is the constant that represents the computational requirement per agent and n represents the number of agents have been generated.

Figure.23 shows a moderate frame rate changes due to the density-level increases where we rendered groups of one hundred agents under the same gender ratio. The density-level as the value of X-axis is acquired by adjusting the spread-level value

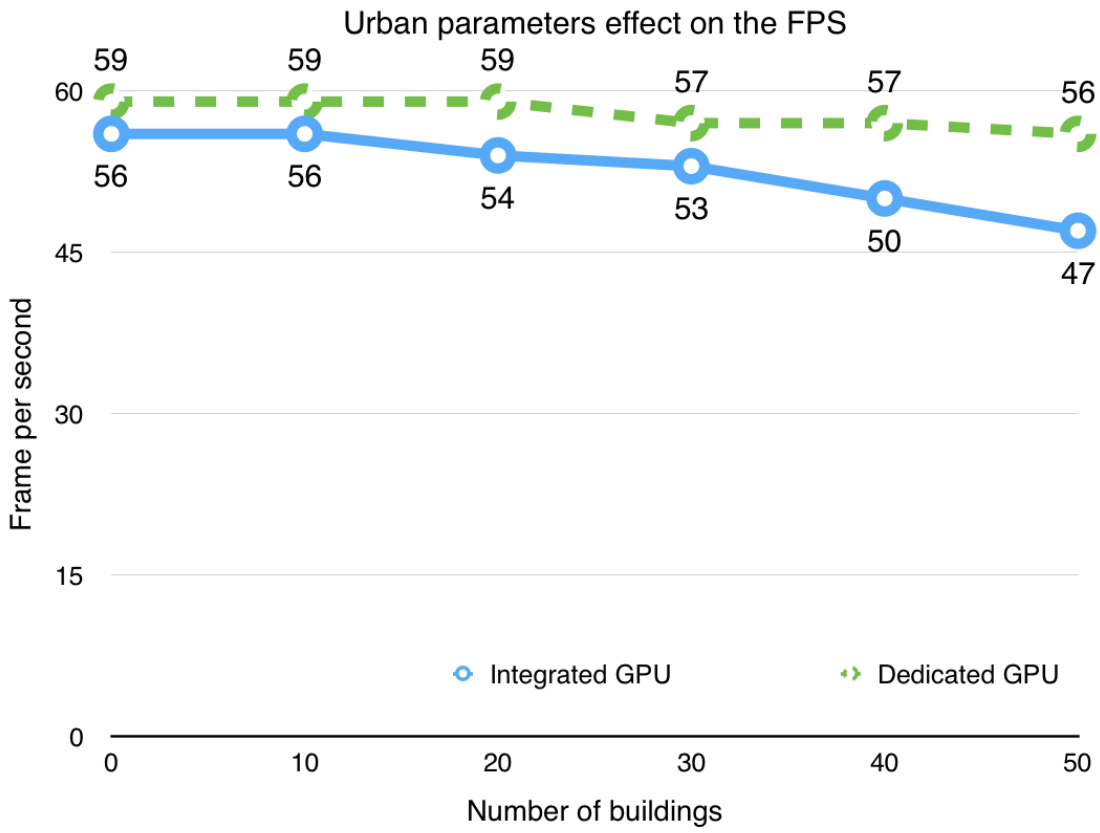


Fig. 25: Urban parameters affects on the FPS

from the minimum to maximum and we select five different values from low to high. From this chart, we can observe that low-density could increase the system running performance slightly better compared with other density-levels. However, the density-level does not have too much impacts on the overall frame rate performance in both system configurations.

At first, we did not consider that the gender ratio could be a factor, which affects on the program running performances. Figure.24 shows the example of the variation on the gender ratio changes the frame rate in our program. Those data were recorded under the condition of two hundred agents and with the same spread-level. Although, the frame rate was not affected dramatically as in the population experiments. We still can explain the reason for this situation. A scene with full male characters has higher frame rates than any other scene, which has female characters existed. Thus, we can find the reason for this is our female model has approximately five hundred more vertices (total 8067 versus 7600) than the male model, which female agent consumes more computational power when rendering and this leads the frame rate dropping.

Figure.25 indicates the frame rate comparison result base on the number of buildings in the urban scene. From no building at all, which leaves a spacious area to place crowds, to fifty buildings, which almost assigned buildings at every possible spot on the map. However, the number of buildings in the environment causes an unremarkable frame rate drop in both system configurations, just around ten FPS dropped in the integrated GPU system.

4.4.1 Priority Rule-based Navigation System

Besides the experiments on the crowd generating system, we also tested our dynamic methods to evaluate the navigation system. At first, our model has experimented with non-priority rule applied to our agents. The non-priority navigation system brings us a behaviour problem that all agents have no 'knowledge' to yield the right of way to others when their navigation paths are crossed. This issue made our walkers sometimes got into an endless trapping condition when two or more agents come face-

to-face without avoiding or detouring actions. This 'deadlock' problem apparently affected our navigation efficiency and it is an immediate issue to our decision-making function. To deal with this issue, we implemented the priority-rule model exclusively for our agents. Agents in our model can detect and avoid buildings [obstacles] when moving. However, when this 'deadlock' situation happens, the agents can only 'stick' with the other walker[s]. That's why we need to give different priority-levels to every walker in the simulation. To make them 'aware' which agent has more the right of way when they met. We use a numeral system to decide the priority level for each agent particularly. Currently, we assigned the priority level with one hundred levels, from the lowest with value 1 to the highest with value 100. The agent with low-priority should present a behaviour of yield the right of way to any other agent who has higher priority value. In our model, low-priority agents need to decrease the current velocity or change the walking route to avoid collision with agents who have greater priority-value. In the reverse case, the high-priority agents can push away other agents to clear the path for themselves. After this rule-based model applied in our model, we can simulate a full evacuation scenario without the 'deadlock' issue.

We experimented the dynamic model performances by several sample tests. We have designed a typical four-way urban model for the testing environment and with buildings that generated by BCGS.

Table 3: The time for evacuation after the application of the rule-based model

Crowd Size	Evacuation Time	Fallen Agents
100	2 min 14 sec	0
200	5 min 12 sec	0
300	14 min 59 sec	2
400	25 min 06 sec	5

We formed four groups of testing samples by changing on number of agents that need to be evacuated. The Table.3 shows the result of the full evacuation time (all agents arrived around the destination except the fallen agents) with our rule-based

model applied. We found that the evacuation time almost doubled with the increase of the number of people in the crowd. With the simulation has over three hundred agents involved, we can find some agents has fallen due to the surrounding density. Figure.26 is a screenshot of our agents were being evacuated from centre area to the destinations. This picture also captured two fallen walkers that have left on the ground.

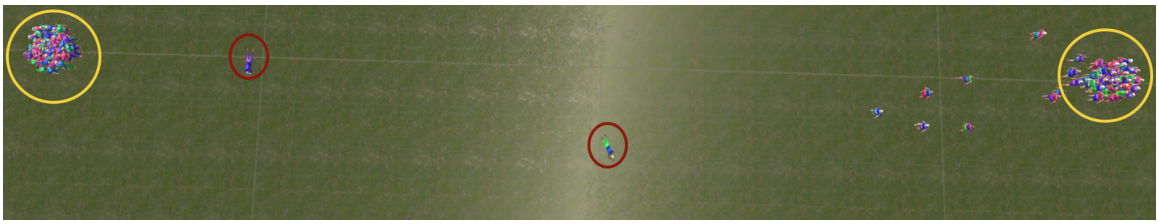


Fig. 26: A sample picture of agents were being evacuated from the city centre

There is a special case that we want to discuss for a clarification of our navigation system computational complexity. In our model, perceived obstacles can interfere the current pathfinding function on one agent at the realtime and our BCGS can generate buildings in any time during the simulation. So, the case is when an agent path is blocked by a newly spawned building, by our dynamic function, he [she] perceives it and reroute. Then, what is the computational complexity to make this detour? Here is the explanation: first, the pathfinding method connects the desired destination and the current agent position in a polynomial computational complexity. In the case that a new obstacle has generated and blocked on one agent's current moving path, we still can reroute a new path in the same computational complexity. Thus, in the worst case (continuously generates new obstacles on the agent path during navigation, which is unlikely to happen in our crowd simulation system), the computational complexity of re-navigation is $o(n^2)$ where n is the pathfinding system complexity.

4.4.2 Result

In this section, we discuss the program functionality and evaluate general performances of our proposed methods with other existed models.

Incorporate Performance

An analysis on the incorporate performance is important for one complete system. In our simulation system, the performance cost is mainly from two sources, the rendering system and the dynamic system. The agent rendering is the main source in rendering system and its computational cost is $O(c^n)$ where c is the unit graphics cost per agent and n represents the number of agents rendered. On the other hand, the cost of our dynamic system is mainly divided by two sub-functions, the collision algorithm with a polynomial computational complexity, and the navigation system with $o(n^2)$ computational complexity. Thus, The general performance of our model is $f(n) = O(c^n + n^2 + n)\ as\ n \rightarrow \infty$ where $f(n)$ represents the incorporate performance complexity.

Comparison with Other Proposed models

We chose four proposed models to compare with our model: Pelechano's psychological model [33], Tecchia's Real-time model [37], Ondřej's synthetic-vision model [28], and Shendarkar's BDI agent model [34]. The principle of choosing which model to present the comparison is based on each method's relevant-degrees with our thesis objectives. In the selected models, we can find the most comparable aspects with our model. Thus, the evaluating criterion could be reliable. The evaluation result is shown in Table.4.

The population capability and the density-level are not included in our evaluation chart, although they are crucial factors in a crowd simulation study. The reason for this is the performance of those two features are firmly related to the computer power and the rendering quality that implemented on the model. To clarify, each agent in our program has roughly 8000 vertices regarding the polygons, which is eighty times more than the 3D agents in Pelechano's HiDAC [30] model as shown in Figure.27. Moreover, in his model, he rendered six hundred human models in one time as limitation. However, our model can render more than two thousand agents at the same time. The reason for this performance differences is partially due to the

Table 4: Major features comparison with other models

	BDI [34]	**Vision [28]**	**Real time [37]**	**PMFserv [35]**	**Our Method**
Crowd Rendering Type	2D graphics	3D Triangle Shapes	3D male model with LOD and IBR	3D Polygons	3D male and female model with DAGM applied
Agent behaviour method	BDI (belief, desire, intention)	Vision-based collision avoidance method	2D grid contains behaviour layer to agents	PMFserv stress-based model	MCS gives the comprehensive evaluation of an agent surroundings
Dynamic Method	Dijkastra's shortest path algorithm	Angular & velocity control	Collision, Intercollision, Behaviour, Callback layers	MACES crowd simulation method	Navigation and pathfinding based on the priority-rule
Computer Animations	No animation applied	No animation applied	Walking animation	Animation included but did not present in the paper	Idle, walking, falling with active conditions
Environment Features	Matrix representation maps	An orthogonal corridors	Collision map with 3D architectures	Blocks	BCGS creates both systematic city plan and visualization buildings
Extra Features	Evacuation under a truck bomb attack	Circle, group swap, crossing model	Program runs in real-time	Combine PMFserv with MACES into one model	UI control, applicability, stampede simulation

computer performances differences. Also, almost every crowd simulation model is established under different system configurations and development platforms. Thus the performances on population capability or density-level cannot become a standard criterion for the evaluation.

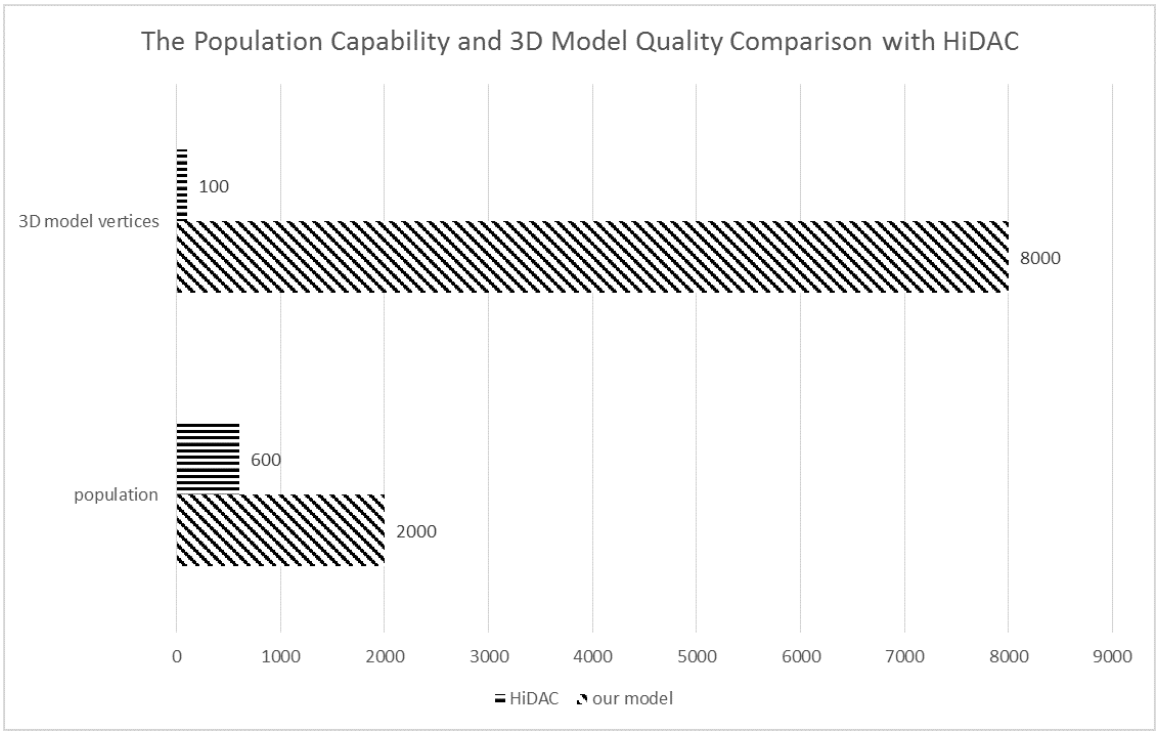


Fig. 27: The comparison with HiDAC model on population capability and graphics quality

As we listed in Table.4, our program has more comprehensive features than all other methods we have reviewed. With the visually diverse crowd generating method DAGM, the agent behaviour method MCS, and environment system BCGS. Our model can instantly produce an animated crowd with specified quantity, high-realism rendered and dynamic function applied agents. We additionally provide a gaming control environment for the simulation process, thus, our model has better accessibility than any other reviewed models. At the last, the entire system can be packed as an independent software that can be executed on Microsoft Windows or MacOS.

CHAPTER 5

Conclusions

In this thesis, we proposed a new realtime Agent-based crowd simulation system which improves the rendering realism by applying our Distinguishable Agents Generating Method (DAGM). Unlike other agent-based models which usually build agents with same and simple appearances; our DAGM offers: (a) high-quality male and female 3D models under the Separating Components Modelling Rule (SCMR), (b) auto-generating agents with different colour, size, and hairstyle choices, and (c) the flexibility on controlling the crowd density and the gender ratio during simulation time. Thus, We can generate a graphically distinctive crowd for our crowd simulation study.

Besides the works on the visual rendering aspect, our model also contributes on the dynamic method for ACS study. First, the Multi-layer Collision System (MCS), which computes and records the realtime information of surrounding densities and collision detections for every agent. Second, the priority rule-based navigation and pathfinding system can solve the popular 'deadlock' problem in many decision-making models. Thus, based on the support from our collision and navigation system, all walkers have a high-level automatic dynamic system to use in a complicated evacuation scenario. Our model can bring out unsystematic evacuation results, in other words, the simulation is unpredictable on agent's movement at all; each single simulation is a unique study case.

Moreover, we designed the Building & City planning Generating System (BCGS) to create city-schemes and obstacles. We tested our crowd's dynamic model in a cross street scenario, which provides obstacles to block agents' movements in the

simulation. Agents who are interfered by buildings can alter their navigation path to arrive at the desired destination.

We evaluated our proposed methods' performances by recording the simulation frame rate under two different system configurations. Also, we compare our model with several up-to-date agent-based models. By comparing the inclusive results, our ACS model has a massive combination of different body and uniform styles to build the $3D$ crowd; more flexibility in controlling the crowd parameters: density, scale and number of agents. The entire process is in a semi-automatic process where users only need to define several key-parameters at the beginning. The simulation is ready for experiment with lots of diversity on crowd moving paths and the possibility of accidents, etc.

Crowd simulation by computer graphics is usually a complex system with lots of elements need to be defined and controlled. However, our program makes this simulation procedure direct and easy to handle. Handful settings required to complete a customized simulation process. Technically, our program has the export-ability to embedded into other programs such as video games or $3D$ applications to extend the practical application of our program.

5.1 Future Work

We summarize several aspects that can be improved in the future works.

1. Making enhancement on the navigation method efficiency. Our current navigation function solves the 'deadlock' problem among agents' moving, but the navigation procedure spent significantly much time while the population was increased.

2. Optimization on the MCS, our collision system has a multi-factorial evaluating criterion to control the agent's status. Currently, we assign the density factor as one system parameter. However, more individualized factors, for example, gender, body type and personalities should be expected. Additionally, with the

multi-factorial evaluating system's help, our stampede simulation could become more reliable and accurate.

3. Design more city scenario templates, for example, an indoor environment map that can support to simulate an evacuation situation inside a building. we also plan to apply more behaviour and interaction models (talking and moving together when acquaintances met) into our agents to achieve an even higher realism-level.

REFERENCES

[1] Abe, K., Abrahams, J., Ando, K., Ota, H., Oki, T., Aoki, T., Ashe, B., Shields, T., Aubé, F., Shield, R., et al. (1986). The science of human panic. *Brain Publ. Co., Tokyo.*

[2] Aguilar, W. G., Luna, M. A., Moya, J. F., Luna, M. P., Abad, V., Ruiz, H., and Parra, H. (2017). Real-time detection and simulation of abnormal crowd behavior. In *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*, pages 420–428. Springer.

[3] Algadhi, S. A. and Mahmassani, H. S. (1990). Modelling crowd behavior and movement: application to makkah pilgrimage. *Transportation and traffic theory*, 1990:59–78.

[4] Ali, S., Nishino, K., Manocha, D., and Shah, M. (2013). Modeling, simulation and visual analysis of crowds: a multidisciplinary perspective. In *Modeling, Simulation and Visual Analysis of Crowds*, pages 1–19. Springer.

[5] Dadova, R. J. B. (2014). Cellular automata and particle systems for crowd simulation in selected environments. Master's thesis, Comenius University.

[6] Duives, D. C., Daamen, W., and Hoogendoorn, S. P. (2013). State-of-the-art crowd motion simulation models. *Transportation research part C: emerging technologies*, 37:193–209.

[7] Dutra, T. B., Marques, R., Cavalcante-Neto, J. B., Vidal, C. A., and Pettré, J.

(2017). Gradient-based steering for vision-based crowd simulation algorithms. In *Computer Graphics Forum*, volume 36, pages 337–348. Wiley Online Library.

[8] Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208.

[9] Fromm, J. E. and Harlow, F. H. (1963). Numerical solution of the problem of vortex street development. *The Physics of Fluids*, 6(7):975–982.

[10] Getchell, A. (2008). Agent-based modeling. *Physics*, 22(6):757–767.

[11] Glynn, C. J., Herbst, S., Lindeman, M., O'Keefe, G. J., and Shapiro, R. Y. (2015). *Public opinion*. Westview Press.

[12] Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D., and Dubey, P. (2009). Clearpath: highly parallel collision avoidance for multi-agent simulation. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 177–187. ACM.

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

[14] Igarashi, T., Matsuoka, S., and Tanaka, H. (2007). Teddy: a sketching interface for 3d freeform design. In *Acm siggraph 2007 courses*, page 21. ACM.

[15] Joseph, A. P., Lagerstedt, I., Patwardhan, A., Topf, M., and Winn, M. (2017). Improved metrics for comparing structures of macromolecular assemblies determined by 3d electron-microscopy. *Journal of Structural Biology*, 199(1):12–26.

[16] Knapp, A. (1938). An introduction to clinical perimetry. *Archives of Ophthalmology*, 20(6):4–5.

[17] Kullu, K., Güdükbay, U., and Manocha, D. (2017). Acmics: an agent communication model for interacting crowd simulation. *Autonomous Agents and Multi-Agent Systems*, pages 1–21.

[18] Kurve, A., Kotobi, K., and Kesidis, G. (2013). An agent-based framework for performance modeling of an optimistic parallel discrete event simulator. *Complex Adaptive Systems Modeling*, 1(1):12.

[19] Lamarche, F. and Donikian, S. (2004). Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. In *Computer graphics forum*, volume 23, pages 509–518. Wiley Online Library.

[20] Larsson, A. (2017). Real-time persistent mesh painting with gpu particle systems. Master's thesis, Liknoping University.

[21] Li, B., Lee-Urban, S., and Riedl, M. O. (2012). Toward autonomous crowd-powered creation of interactive narratives. In *5th Workshop on Intelligent Narrative Technologies, Palo Alto, CA*, volume 8, pages 25–52.

[22] Lu, G., Chen, L., and Luo, W. (2016). Real-time crowd simulation integrating potential fields and agent method. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 26(4):28.

[23] Mahmood, I., Haris, M., and Sarjoughian, H. (2017). Analyzing emergency evacuation strategies for mass gatherings using crowd simulation and analysis framework: Hajj scenario. In *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 231–240. ACM.

[24] Mourino, G., Evans, M., Edzenga, K., Cavaleri, S., Adams, M., and Bisceglio, J. (2017). Populating the crowds in ferdinand. In *ACM SIGGRAPH 2017 Talks*, page 68. ACM.

[25] Moussaïd, M., Kapadia, M., Thrash, T., Sumner, R. W., Gross, M., Helbing, D., and Hölscher, C. (2016). Crowd behaviour during high-stress evacuations in an immersive virtual environment. *Journal of The Royal Society Interface*, 13(122):4–14.

[26] Narain, R., Golas, A., Curtis, S., and Lin, M. C. (2009). Aggregate dynamics for

dense crowd simulation. In *ACM Tnariansactions on Graphics (TOG)*, volume 28, pages 122–129. ACM.

[27] Niazi, M. and Hussain, A. (2011). Agent-based computing from multi-agent systems to agent-based models: a visual survey. *Scientometrics*, 89(2):479.

[28] Ondřej, J., Pettré, J., Olivier, A.-H., and Donikian, S. (2010). A synthetic-vision based steering approach for crowd simulation. In *ACM Transactions on Graphics (TOG)*, volume 29, page 123. ACM.

[29] Pauls, J. (1987). Calculating evacuation times for tall buildings. *Fire Safety Journal*, 12(3):213–236.

[30] Pelechano, N., Allbeck, J. M., and Badler, N. I. (2007). 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.

[31] Pelechano, N. and Badler, N. I. (2006). Modeling crowd and trained leader behavior during building evacuation. *IEEE computer graphics and applications*, 26(6):4–80.

[32] Pelechano, N. and Malkawi, A. (2008). Evacuation simulation models: Challenges in modeling high rise building evacuation with cellular automata approaches. *Automation in construction*, 17(4):377–385.

[33] Pelechano, N., O'Brien, K., Silverman, B., and Badler, N. (2005). Crowd simulation incorporating agent psychological models, roles and communication. Technical report, Pennsylvania Univ Philadelphia Center for Human Modeling and Simulation.

[34] Shendarkar, A., Vasudevan, K., Lee, S., and Son, Y.-J. (2006). Crowd simulation for emergency response using bdi agent based on virtual reality. In *Proceedings of the 38th conference on Winter simulation*, pages 545–553. Winter Simulation Conference.

[35] Silverman, B. G., Badler, N. I., Pelechano, N., and O'Brien, K. (2005). Crowd simulation incorporating agent psychological models, roles and communication. *Center for Human Modeling and Simulation*, page 29.

[36] Sung, M., Gleicher, M., and Chenney, S. (2004). Scalable behaviors for crowd simulation. In *Computer Graphics Forum*, volume 23, pages 519–528. Wiley Online Library.

[37] Tecchia, F., Loscos, C., and Chrysanthou, Y. (2002). Visualizing crowds in real-time. In *Computer Graphics Forum*, volume 21, pages 753–765. Wiley Online Library.

[38] Tong, W. and Cheng, L. (2013). Simulation of pedestrian flow based on multi-agent. *Procedia-Social and Behavioral Sciences*, 96:17–24.

[39] van Toll, W., Cook, I., Atlas, F., van Kreveld, M., and Geraerts, R. (2016). The explicit corridor map: Using the medial axis for real-time path planning and crowd simulation. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 51. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[40] Vermeulen, J. L., Hillebrand, A., and Geraerts, R. (2017). A comparative study of k-nearest neighbour techniques in crowd simulation. *Computer Animation and Virtual Worlds*, 28(1):3–4.

[41] Wagner, N. and Agrawal, V. (2014). An agent-based simulation system for concert venue crowd evacuation modeling in the presence of a fire disaster. *Expert Systems with Applications*, 41(6):2807–2815.

[42] Watts, J. M. (1987). Computer models for evacuation analysis. *Fire Safety Journal*, 12(3):237–245.

[43] Weiss, T., Litteneker, A., Jiang, C., and Terzopoulos, D. (2017). Position-based multi-agent dynamics for real-time crowd simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 27. ACM.

[44] Zheng, X., Zhong, T., and Liu, M. (2009). Modeling crowd evacuation of a building based on seven methodological approaches. *Building and Environment*, 44(3):437–445.

[45] Zhou, S., Chen, D., Cai, W., Luo, L., Low, M. Y. H., Tian, F., Tay, V. S.-H., Ong, D. W. S., and Hamilton, B. D. (2010). Crowd modeling and simulation technologies. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 20(4):20.

VITA AUCTORIS

NAME:	Songqiao Sun
PLACE OF BIRTH:	Beijing, China
YEAR OF BIRTH:	1993
EDUCATION:	Beijing University of Technology, B.Eng., School of Software Engineering, Beijing, China, 2015
	University of Windsor, M.Sc., Computer Science, Windsor, Ontario, 2017