The University Of Sheffield.

Agent-based Pedestrian Simulation on GPUs for use in Decision Support Systems

By:

Twin Karmakharm

A thesis submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy

The University of Sheffield
Faculty of Engineering
Department of Computer Science

May 2018

Abstract

Agent-based simulation of pedestrian crowds in public spaces can give insights into the potential congestion areas and space utilisation allowing new spaces to be better designed and existing ones to be more efficiently managed. This thesis investigates the components essential for a Graphics Processing Unit (GPU) based pedestrian simulation system that has the potential to be integrated into a real-time Decision Support System (DSS).

Two navigation approaches are presented. A novel agent-based navigation grid approach uses a grid of agents to represent navigation information and for static obstacle avoidance. A case study of an urban environment shows it is a straightforward and efficient way of implementing navigation behaviour, allowing for the FLAME GPU agent-based framework to handle the necessary GPU optimisation.

Then, a novel searchable and fully-resolved navigation graph approach is presented that allows pedestrian agents to make branching decisions and minor route changes in the case of congestion. Two cases of a shopping mall and a train station based on real environments show that navigation access times are good and memory use is two orders of magnitude less than the grid-based approach.

A prototype pedestrian simulation software Concoursia is presented which integrates the searchable navigation graph approach and allows the authoring of environments, visualisation, and collection metrics. Support for public transport services, queue agents and waiting behaviours are also implemented. The results show that the system can be used to create and simulate complex environments where pedestrians have a large number of navigation goals.

Finally, a pedestrian multi-simulation system is presented that manages the simulation on multiple machines equipped with GPUs. Multiple simulation instances are merged and run as a single simulation efficiently utilising the parallel architecture of the GPU. An initial trial has shown that the system is able to dispatch and run multiple simulations concurrently on multiple machines.

Acknowledgements

First, I would like to thank my wife Thanitta and my family for their motivation, support and understanding, helping me get through the hardest times and made it all possible. I would like to thank my previous supervisor Professor Daniela Romano for the push to get started down this path and for her support as a supervisor and friend over the years. I would like to thank my current supervisor Dr. Steve Maddock who provided support and advice for the last crucial part of the PhD. I would like to thank Dr. Paul Richmond for his support and guidance throughout a large part of my PhD. I would like to thank Professor Mike Holcombe for his support and advice on both the PhD and during my time in the Advanced Computing Research Centre. I would like to thank Dr. Mark Burkitt for his collaboration on the Concoursia project and for the advice and discussions about the PhD. I would like to thank my friends and colleagues from the Computer Science Group and ACRC, in particular Laura Smith, Lewis Gill and Mariam Kiran for providing support, advice and discussions about different aspects of my project. I would like to thank the external partners, DSTL, BAE Systems, Nottingham and Avon Fire and Rescue Services and Network Rail for the funding, collaboration, case studies and data that provided essential contribution to various parts of the project. Finally, I would like to thank all my friends for the patience, understanding and motivation I've been given throughout the years.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text. This work has not been submitted for any other degree or professional qualification except as specified.

Twin Karmakharm
May 2018

Table of contents

List of figures **xiii**

List of tables **xxiii**

1 Introduction **1**
- 1.1 Research Outline . . . 3
- 1.2 Contribution to Knowledge . . . 4
- 1.3 Thesis Structure . . . 6

2 Related Work **9**
- 2.1 Agent-based Modelling . . . 9
- 2.2 Pedestrian Modelling and Simulation . . . 10
  - 2.2.1 Local Motion . . . 13
  - 2.2.2 Navigation . . . 17
- 2.3 Pedestrian Simulation Systems and Frameworks . . . 20
- 2.4 Pedestrian Simulation and Decision Support Systems . . . 23
  - 2.4.1 Pedestrian Tracking Systems . . . 24
- 2.5 Parallel Computing Architecture for Agent-based Modelling (ABM) . . . 25
  - 2.5.1 GPU Architecture for ABM . . . 26
  - 2.5.2 High Performance Computing (HPC) Architecture for ABM . . . 30
  - 2.5.3 A Summary of GPU and HPC Architectures . . . 32
- 2.6 The FLAME Framework . . . 33
  - 2.6.1 FLAME and FLAME GPU System Overview . . . 33
  - 2.6.2 FLAME for the HPC Architecture . . . 36
  - 2.6.3 FLAME for the GPU Architecture . . . 39
  - 2.6.4 FLAME HPC and GPU Performance Benchmarking . . . 40
- 2.7 Summary . . . 41

3 Navigation for Pedestrian ABM **45**
- 3.1 Agent-based Navigation Grid . . . 46

3.1.1 Pedestrian Agents . . . 48
3.1.2 Navigation Agents . . . 50
3.1.3 Environment Editor . . . 58
3.1.4 Simulation & Results . . . 59
3.2 Searchable Navigation Graph . . . 67
3.2.1 The Environment Graph . . . 69
3.2.2 The Itinerary Graph . . . 70
3.2.3 Routes Generation . . . 72
3.2.4 The Navigation Graph . . . 72
3.2.5 Other Properties of the Navigation Graph . . . 74
3.2.6 Implementation as a Navigation Module in FLAME GPU . . . 75
3.2.7 Static Obstacles & Environment Bounds Detection . . . 80
3.2.8 Evaluation and Discussion . . . 80
3.3 On the Two Navigation Approaches . . . 83
3.3.1 Memory Use . . . 83
3.3.2 Navigation Performance . . . 89
3.3.3 Discussion . . . 89

4 Concoursia, a prototype pedestrian simulation system 91
4.1 System Overview . . . 92
4.2 Creation of A Scenario . . . 93
4.2.1 Environment Objects & Environment Designer Mode . . . 93
4.2.2 Environment Object Graph & Network Editor Mode . . . 97
4.2.3 Event Schedule & Itinerary and Schedules Mode . . . 97
4.3 Building the Simulation . . . 99
4.4 Simulator . . . 102
4.4.1 The Concoursia Agent Model . . . 103
4.4.2 Navigation Module . . . 104
4.4.3 Simulation Mode . . . 105
4.5 Visualisation . . . 106
4.6 Metrics . . . 106
4.7 Quantitative Evaluation of Agent Navigation . . . 109
4.8 Real World Environments in Concoursia . . . 110
4.8.1 Shopping Mall . . . 113
4.8.2 Train Station . . . 122
4.9 Discussion . . . 131

5 A Prototype System for Multi-Simulation of Pedestrian Models 135
5.1 System Overview . . . 136
5.1.1 Simulation Manager . . . 136
5.1.2 Local Simulation Manager . . . 136
5.1.3 Simulator . . . 138
5.1.4 GUI Client . . . 138
5.2 Simulation Batching . . . 138
5.3 Collection of Metrics . . . 140
5.3.1 Obtaining an Accurate Pedestrian Generated Count . . . 143
5.3.2 Crowd Density and Flow . . . 143
5.4 The Scenario: Evacuation After a Dirty Bomb Incident . . . 145
5.5 The Agent Model . . . 146
5.5.1 Pedestrian Agent . . . 146
5.5.2 Environment Agent . . . 147
5.5.3 Model Behaviour . . . 148
5.6 Collection of Zone and Disrobing Metric . . . 149
5.6.1 Zone and Disrobing Metric . . . 149
5.7 Results . . . 150
5.7.1 Environments . . . 150
5.7.2 Model performance . . . 151
5.7.3 Performance on multiple GPUs . . . 152
5.8 Discussion . . . 153

6 Conclusions 175
6.1 Limitations and Future Work . . . 177

References 179

List of figures

2.1 CUDA programs are launched as grids of blocks. Each block contain multiple worker threads. Threads in the same block are guaranteed to run on the same GPU core. 28
2.2 The diagram shows the difference in GPU memory access patterns for internal Array of Structures (AoS) and Structure of Arrays (SoA). SoA leads to fewer memory fetches overall as one request obtains a chunk of memory that can be used by multiple threads. 30
2.3 Process of transition between agent states and its use of the message board. 34
2.4 Example agent model definition with two agents. Each agent has 3 states (AS# and BS#) and 2 transition functions (AF# and BF#) that either send or receive a message. 35
2.5 Architecture of the FLAME and FLAME GPU frameworks. 36
2.6 Dependency graph generated for the example model in Fig. 2.4. . . . 38
2.7 An updated dependency graph with data granularity defined. In this example, the transition functions AF1 and BF2 are independent and hence can be executed concurrently. 38
2.8 Simulation times for various agent populations on different GPU hardware. 42

3.1 Pedestrian agents exist on top of a grid of discrete space navigation agents and their position can be transformed to a relevant navigation agent cell. 47
3.2 Sequence diagram for function call of both agents and their utilization of the message pool. 47
3.3 An example of a 6x6 environment with 2 exits and obstacles. Navigation agents are arranged in a grid as shown (top right) and the position of the agent relates to a particular cell of the vector fields. 52

3.4 A vector representation of the environment is rasterised into a grid of navigation agents. White cells are walkable terrain, black cells are static obstacles and red cells are exits which also double as entrances. 54
3.5 Wavefront propagation performed on the obstacle areas. Black areas are obstacles, white areas are walkable, light blue is the area covered in the first iteration and dark blue areas are covered in the second iteration. 55
3.6 Visibility rules for the wave propagation. Starting from origin (yellow), the green areas will be included while the grey areas are not due to it being blocked off by obstacles. 55
3.7 Rotation of navigation agent cell to pedestrian distance vector $\vec{d}_{iC}$ to align with the the frame of reference of vector $\vec{n}_{iW}$. 56
3.8 Wavefront propgation performed on a destination cell (red) over 5 iterations. 56
3.9 Comparison of the generated force vector fields. (a) A Collision Vector Field (CVF) with force influence distance of 2. (b) Shows an Navigation Vector Field (NVF)s with simple propagation , with backflow propagation but no limit (c) and the result of the final algorithm (d). . 57
3.10 Comparison of the simulation using un-optimised (a) and with backflow smoothed (b) navigation vector fields. 59
3.11 Illustration of backflow propagation in action. Take an environment that has 8 iterations of propagation (a), each blue colour signify a boundary of a wavefront propagation iteration. The colours are repeated at the 5th iteration. (b) Shows examples of backflow propagation three locations. Backflow propagation limit is three iterations. The backflow origin B1 is able to propagate the full three iterations while B2 is stopped at one iteration as it encountered an obstacle. As B3 is next to an obstacle the backflow algorithm does not run. 60
3.12 The plan editor software for generating the vector fields. The example shows a 2-exit environment that was set to be a dimension of 64x64 with maximum backflow propagation iteration of 5. Exits are shown as red squares and obstacles as black squares. The two NVFs and an CVF generated are shown where red arrows are NVF and blue arrows are CVF. 61
3.13 Performance benchmarking for square environments of width 64, 128, 256 and 512 measuring iteration time is measured when running with 1024, 4096, 16384, 65536 and 262144 pedestrian agents. 63

3.14 Screenshot of the urban environment simulation running with 10,000 pedestrians. Red lines signify locations of entrances and exits. 65
3.15 Time utilization by different agent functions with increasing pedestrian agent density 66
3.16 a) For a given scenario, graph data structures are created for use in the simulation. b) The Environment Graph is a representation of the environment and how each part is connected. c) The Itinerary Graph represents a high-level set of objectives. d) The Navigation Graph contains the precise sequence of graph nodes that must be traversed in order to reach an objective location. 68
3.17 The figure shows the process of creating the Environment Graph. (a) The outline of the environment in black. (b) The environment discretised to a binary grid where the white colour is a walkable area and the gray colours are non walkable. (c) The distance transform where deeper green indicates higher distance transform value. (d) The nodes with cicular areas created for the Environment Graph. 70
3.18 Generation of an R-tree from the Environment Graph in Fig. 3.16(b). 71
3.19 The process of creating the Navigation Graph. (a) The Environment graph for an example environment with an entrance, a shop and two exits. (b) The Itinerary Graph. (c) The routing in the physical environment. (d, e & f) The exact sequence of nodes is found for each Route in the Itinerary. Two Navigation Graphs are required for this Itinerary as the nodes overlap in opposite directions. (g) The Navigation Graph from Entrance 1 to the shop (green arrows). (h) The Navigation Graph from the Shop and branches to either Exit 1 (yellow arrows) or Exit 2 (blue arrows). For both Navigation Graphs the brown arrows guide pedestrians back to the itinerary routes. 73
3.20 A simplified information flow diagram of the Navigation Data structure. 77
3.21 The Navigation Data Structure Fig. 3.20 laid out as SoA objects for use on the GPU. 78
3.22 An example of branching in the Navigation Node Connections Array. The pedestrian is at node N1 and there are two routes with IDs 1 and 2 each having 2 branching possibilities. 78

3.23 During the simulation, a trapezoidal shape navigation area (red line) is formed from the two diameter lines of the current and next node and used as a navigation guide. Pedestrians outside of the navigation area try to navigate towards it while ones inside the navigation area will try to keep a proportional distance to the centre line between the current and the next node (blue dotted line). 80
3.24 Comparison between the shape of corridors on pedestrian flow during congestion. (a) During congestion, formation of an arch pattern when moving from a wide passage to a narrow corridor or doorway is shown which matches the results of Helbing et al. [22]. Solid circles are pedestrians, black lines mark the environment boundary, grey lines represent the Environment nodes used in the route, red lines marks the gate/corridor and shows how the environment is bounded if following the approach by Pettre et al. [47]. (b) A simulation when the environment is bounded by corridors creating a funnel shape. The exit corridor size is adjusted to be the same as the first case to provide a fair comparison of the exit flow rate. 82
3.25 The shape of the corridor causes a large difference in flow rate. It can be seen that the funnel shape allows for pedestrians to exit from the environment much faster. 82
3.26 Non-walkable areas in black represent the walls and a set of turnstiles. The pedestrian (red circle) has a desired route (a) but is forced to go through another slot due to congestion (b). The pedestrian tries to go back to the pre-specified route causing congestion (c). With dynamic routing (d), the pedestrian is able to switch to using the alternate turnstile without problem (e). 84
3.27 A subset of the Wedge environments used to compare memory usage between the two navigation approaches (Tables 3.2 and 3.3). Green capsules represents portals. The environment bounded by brown lines is 10x10m and the environment bounded by blue lines is 25x25m. The left side is fixed at 4m and the right side is the same length as the environment. 86

3.28 A subset of the Corridor environments used to compare memory usage between the two navigation approaches (Tables 3.2 and 3.3). The environment is bounded by black lines and the green capsules represents portals. The environments shown in (a)-(b) are of size 10x10m and 25x25m, respectively. For every 5m of the environment a 2m square obstacles is added and evenly distributed to create a corridor layout so that the complexity of the environment grows exponentially with its size. 86

3.29 Memory usage comparison for the navigation types on a wedge shape environment and corridor environment. For grid-based navigation (solid line), the memory use is the same for both environments and increases exponentially with environment size. For graph-based navigation (dashed lines), memory use is highly dependent on the complexity of the environments rather than the size as can be seen by the disparity for the Corridor environment (dashed line) and Wedge environment (dash-dotted line). 87

4.1 A simplified data flow and interaction diagram of Concoursia's different components. 94

4.2 A simple bus station environment with two floors and a set of stairs connecting them. The environment is bounded by the walkable area within the black lines. It shows the Static Obstacles (red areas), Portals (green areas), Waypoints (blue areas), Waiting area (grey areas) and Queues (purple dots and lines). 95

4.3 Overlapping Walkable Objects are merged into a single walkable area during the building of the simulation. 96

4.4 Environment objects of the same type can be grouped in the Network Editor Mode. To the simulation they are seen as branches with equal probability that a pedestrian can choose from. 97

4.5 The connections created for the simple bus station environment in Fig. 4.2. Each orange dot represents a networkable Environment object. They are connected with the yellow lines on the same floor and a dotted grey line on different floors representing stairs. Waiting areas and queues are special cases where association with a Portal or Waypoint is set through object properties rather than in the network. They are shown connected to the Portal or Waypoint object with dotted purple lines. 98

4.6 The Portal, Activity, Services, Itinerary and Schedule pages in the Itinerary and Schedules Mode. 100
4.7 The Itinerary Graph generated with the environment in Fig. 4.5 for the example in Fig. 4.6. 101
4.8 Concoursia's agent model showing the Pedestrian and Queue agent functions and the messages used. 104
4.9 Simulation and 3D visualisation of the bus station model. The 'Current speed' bar at the top allows for adjustment of simulation speed either to gain increased simulation performance (speeding up) or easier visual inspection (slowing down). 105
4.10 Screenshot of the pedestrian count metric showing number of pedestrians in the simulation over time for all simulation instances. X axis is time (hours and minutes) and Y axis is the pedestrian count. 106
4.11 The evacuation graph shows the number of people who have exited the environment during a certain time period, which is used for analysing the rate of exit. X axis is time (hours and minutes) and Y axis is the number of pedestrians that exited during the period. 107
4.12 Origin and Destination matrix shown as a circular migration flow plot. 108
4.13 The four Tree environments used for testing branching probability during pedestrian navigation. Environment 2 has been enlarged and annotated to show the location of the entrance (green), waypoints (blue) and exits (red). Yellow lines represent branching connections in the itinerary. 110
4.14 Exit time and Distance travelled distribution for the Tree environment with one level of branching. Distributions are separated by the pedestrians' chosen exit. The distribution is obtained over 100 simulation runs. 111
4.15 Exit time and Distance travelled distribution for the Tree environment with two levels of branching. Distributions are separated by the pedestrians' chosen exit. The distribution is obtained over 100 simulation runs. 112
4.16 Exit time and Distance travelled distribution for the Tree environment with three levels of branching. Distributions are separated by the pedestrians' chosen exit. The distribution is obtained over 100 simulation runs. 112

4.17 Exit time and Distance travelled distribution for the Tree environment with four levels of branching. Distributions are separated by the pedestrians' chosen exit. The distribution is obtained over 100 simulation runs. 113
4.18 A shopping mall environment with four entrances and 12 shops. The grey colour represent walkable areas, green capsules are portals, and blue rectangles are shops. 116
4.19 Shopping mall normal scenario - a cumulative Level of Service (LoS) map of the simulation run. 117
4.20 Shopping mall normal scenario - an iteration time over number of pedestrian graph broken down by the agent functions specified in Fig. 4.8 averaged over 100 simulation run. 118
4.21 Shopping mall evacuation scenario - a cumulative LoS map of the simulation run. 119
4.22 Shopping mall evacuation scenario - an iteration time over number of pedestrian graph broken down by the agent functions specified in Fig. 4.8 averaged over 100 simulation runs. 120
4.23 Shopping mall evacuation scenario. A scatter plot of Exit time (time to exit the environment) against Distance travelled over 100 simulation runs. The red line represents the locally weighted scatterplot smoothing (LOWESS) line. 121
4.24 A train station environment with three separate sections, 4 entrances, 1 bus service and 17 platforms. Gray colour represent walkable areas. The sections are connected by stairs grouped by blue, orange, red and green rectangles. Each platform is a group of portals representing each set of train doors. The purple rectangle with black outline represent ticket machines. 124
4.25 Train station normal scenario - a cumulative LoS map of the simulation run. 125
4.26 Train station normal scenario - an iteration time over number of pedestrian graph broken down by the agent functions specified in Fig. 4.8 averaged over 100 simulation runs. 126
4.27 Train station evacuation scenario - a cumulative LoS map of the simulation run. 127
4.28 Train station evacuation scenario - an iteration time over number of pedestrian graph broken down by the agent functions specified in Fig. 4.8 averaged over 100 simulation runs. 128

4.29 Train station evacuation scenario - A scatter plot of Exit time (time to exit the environment) against Distance travelled over 100 simulation runs. The red line represents the LOWESS line. 129
4.30 Screenshot of the train station scenario (Fig. 4.24) running in Concoursia. The view shows the station's Entrance 1. 130
4.31 Screenshot of the train station scenario (Fig. 4.24) running in Concoursia. The view focuses on one of the the station's platform. The red areas indicate train carriage entrance. 130

5.1 The network diagram shows how each of the application in the system interact. 137
5.2 Multiple simulations can run concurrently in the global environment and make use of the Global Pedestrian List. Coordinates transformation must be performed in order to move from local to global and vice-versa. 141
5.3 Simulator visualising two simulation instances running side-by-side. 142
5.4 Parallel removal of pedestrian from the Global Pedestrian List. When a simulation instance stops, pedestrians of the stopped instance are removed from the simulation. 142
5.5 The colours green, orange, blue and red represent an individual simulation. (a) The grid of generated pedestrian counts. (b) These are separated by row for each simulation instance. (c) A row-wise parallel sum can then be performed and the counts can be obtained in the last column. 144
5.6 Parallel generation of LoS grid for multiple simulation instances. Each simulation instance is represented by a different colour. Local simulation space is spatially divided into a custom LoS grid of 2x2 in this example. 155
5.7 The different cordon zones that are set up in order to deal with mass decontamination incidents. The size and layout of these zones can change due to the nature of the threat and environmental factors such as wind direction. The hot zone (red) is an area at high risk of contamination or with active dangers such as fires. The warm zone (orange) is an area where there are no immediate danger but it could also be contaminated. Disrobing points (purple) contain kits to enable pedestrians to remove contaminated clothes modestly. The cold zone (blue) is a safe, non-contaminated area. 156

5.8 The pedestrian agent and environment agent functions used in the model. 156
5.9 This figure shows the different types of maps used to represent the environment. (a) The CVF map is used in avoidance of static obstacles. (b) The exit NVF map guides the pedestrians to the nearest exit in the environment. (c) The disrobing NVF map guides the pedestrian to the nearest disrobing points. (d) Other information such as emitter (green blocks) , disrobing points (orange blocks). (e) The hot, warm and cold cordon zones are represented in as red, orange and blue blocks respectively. 157
5.10 (a) Pedestrians enters the environment. (b) They then follow the evacuation map. (c) After a certain time, cordon zones are established, red is hot, orange is warm and blue is cold. (d) After the cordon has been established, pedestrian outside the warm zone carries on walking to the exit. (e) Pedestrians inside the warm or hot zones walks to the nearest disrobing point. 158
5.11 The colours orange and green represents separate simulations. Pedestrians keep track of the zones that they're currently in and whether they have obtained a disrobing kit. (a) An example of this pedestrian list whereby H, W, C and D refers to hot zone, warm zone, cold zone and disrobed respectively. (b) All of these properties are divided into separate arrays. Different simulations are also separated. (c) A parallel sum can be performed on all of these arrays in order to obtain the final counts. 159
5.12 Nottingham shopping mall environment used for testing the multi-simulation system. 160
5.13 Nottingham shopping mall evacuation plan 1 - the decontamination tunnel is placed at the left most exit, with a disrobing point on the left side of the large obstacle left of the mall. 161
5.14 Nottingham shopping mall evacuation plan 2 - the decontamination tunnel is placed at one of the top exits, with a disrobing point on the left side of the Shopping Mall before the large obstacle. 162
5.15 Sheffield town hall evacuation plan 1 - the decontamination tunnel is placed at the bottom left exit. 163
5.16 Sheffield town hall evacuation plan 2 - the decontamination tunnel is placed at the top left exit. 164
5.17 Sheffield city hall evacuation plan 1 - the decontamination tunnel is placed at the bottom right exit. 165

5.18 Sheffield city hall evacuation plan 2 - the decontamination tunnel is placed at a lower right exit. 166
5.19 Two concurrent running simulations. The scene shows pedestrians gathering at the disrobing point. Nottingham shopping mall environment plan 1 (Fig. 5.13) is shown on the left and plan 2 (Fig. 5.14) is shown on the right. 167
5.20 Four concurrent running simulations. The scene shows pedestrians gathering at the disrobing point. The Sheffield town hall plan 1 (Fig. 5.15) is shown running at the top left, and plan 2 (Fig. 5.16) is running at the top right. The Sheffield city hall plan 1 (Fig. 5.17) is shown running at the bottom left and plan 2 (Fig. 5.18) at the bottom right. 168
5.21 The three graphs shows how the real simulation speed (dt/rdt) changes throughout a simulation in normal conditions. The value dt stands for the dynamic time step and rdt stands for the amount of real time taken to process that step of the simulation. Both these values are averaged over 100 steps. Real simulation speed values over 1 (above the red line) means it is running faster than real time. The line in blue, along with the right axis, shows the number of pedestrians currently in the simulation. 169
5.22 Nottingham shopping mall - start of evacuation. Pedestrians are emitted from the shopping mall (green lines) and walks to the nearest exit (red lines). 170
5.23 Nottingham shopping mall plan 2 - zones as visualised in the simulator. Green is the cold zone, orange is the warm zone and red is the hot zone. 170
5.24 Nottingham shopping mall evacuation plan 1 - pedestrians gathering at the disrobing point. Pedestrians change colour according to their LoS value. 171
5.25 Nottingham shopping mall - simulation visualised with 3D plan of the town centre extracted from GIS data. 171
5.26 Nottingham shopping mall evacuation plan 2 - pedestrians gathering at the disrobing point. The red pedestrians are injured and walk slower. The pink pedestrians are not injured. 172
5.27 Comparison of running the Simulator running the Nottingham shopping mall plan 1 over a single GPU and two GPUs. Each GPU is running 4 instances of the scenario concurrently. 173

List of tables

3.1 The values that would be found in the Navigation Node Connections Array for the example in Fig. 3.22. 79
3.2 Memory use for Grid-based navigation at various environment sizes. . 87
3.3 Memory use for Graph-based navigation at various environment sizes. 88

4.1 The LoS values used for measuring crowdedness of the environment. . 107
4.2 Exit count and percentage for the Tree environments from branching number 1 to 4 over 100 simulations 111
4.3 Graph-based memory use for the Shopping mall environment. 114
4.4 Grid-based memory use for the Shopping mall environment. 114
4.5 Graph-based memory use for the Train station environment. 123
4.6 Grid-based memory use for the Train station environment. 123

Chapter 1

Introduction

Simulation of pedestrian crowds is used widely in planning of public spaces and buildings [1–8]. It gives an insight into the potential congestion areas and space utilisation allowing new spaces to be better designed and existing ones to be more efficiently managed. These public spaces are often large, multi-leveled and complex environments that contain large numbers of pedestrians exhibiting a wide range of behaviour. Within such a simulation, a pedestrian's journey is driven by their own set of goals and objectives, which can be simple as an implicit need to reach a destination or more complex cognitive models [9–14]. How this translates to behaviour can depend on a number of internal and external factors such as emotion, social factors, environment familiarity or congestion [15–20].

Many pedestrian models [21–37] are based on the Agent-based Modelling (ABM) approach where the behaviour is modelled at the micro level, i.e. at the level of individual pedestrians, in order to study systematic effects such as the flow that pedestrians generate. In this thesis, the term pedestrian modelling and simulation will refer to pedestrian models based on ABM principles. The simulation of many individuals has a high computation cost compared to the traditional Equation-based Modelling (EBM) approach [38]. In order to mitigate ABM's high computational cost, the models can be parallelised. This allows the ABM model to take advantage of specialist parallel computing architecture such as the High Performance Computing (HPC) and the Graphics Processing Unit (GPU) to accelerate computation [39, 40].

In order to create a pedestrian model for the prediction of systematic behaviours such as pedestrian flow in real environments, pedestrian agents should be given abilities and behaviours similar to their real-life counterparts. At the most basic level is the ability for them to navigate through a complex environment in order to satisfy a set of defined objectives or goals and to have a perception of their surroundings in order to interact with or avoid other pedestrians and obstacles. The navigation system used with

ABM must be able to scale up to the large number of thousands to tens of thousands of pedestrians as would be expected in large public spaces such as high rises [3, 41], train stations [1, 8, 42] or sections of a city [43, 44].

Many strategies can be used in the representation of a physical environment and for indicating the directions where pedestrians must go. The use of a grid-based [45, 46] dense data structure to hold vector fields is a common approach which is robust, relatively simple to implement and fast to access, but not scalable with environment size. Other approaches use a graph-based representation [47–56] that partitions the environment into a set of nodes. These are more memory efficient and are scalable with environment size. Some other approaches combine the two types of representation [37, 57, 58] in order to take advantage of both systems but come with its own caveats such as needing the environment to not be uniformly dense.

In addition to the pedestrian model, a pedestrian simulation system [26, 30, 59–61] often includes a way to import environments created in design packages, a GUI to help users easily create a scenario and manage the simulation, visualisation of the running simulation and a way to collect of metrics about the simulation for further analysis. These systems [26, 60, 61, 30] are designed for user-guided analysis. From their stated features, they can offer help in the creation of scenarios, providing parameters for pedestrian behaviour and helping with the analysis of the simulation. However, they do not provide alternative solutions based on automated analysis of the simulation results.

A Decision Support System (DSS) [62–67] goes further in helping with the decision-making process by providing analysis of simulation results, exploring possible alternative solutions and suggesting optimal results according to specified criteria. An integrated real-time DSS for crowd flow management would be composed of many elements such as the prediction model based on historical and real-time data, an intervention matrix for suggestion of alternative solutions for managing the crowd flow, and an evaluation system for assessing and comparing the results of prediction for both intervention and non-intervention. A facility is also required for collection of new data (e.g. manual reporting from staff or with the use of sensors) to continuously re-assess the current situation, and to update and validate the prediction model.

A pedestrian simulation system would fit within the prediction model and provide outputs that could be used by the evaluation system. For a continuous real-time system, this means it must be able to provide predictions and suggestions in a time window that allows interventions to be carried out, which can be over a few minutes or hours depending on the situation. It is hence necessary to run multiple simulations in parallel at speeds much faster than real-time. The metrics that can be used to evaluate the simulations must also be collected in a timely manner and automatically ranked

according to specified criteria. In addition, the system should provide a graphical interface which allow users to create and configure the environment and potential scenarios, similar to existing pedestrian simulation packages [26, 30, 60, 61].

1.1 Research Outline

This thesis investigates the use of GPUs for the creation of pedestrian ABM simulation systems. The GPU is a high throughput computational device. Unlike CPUs which have small number of threads and a high clock speed for running a small number of operations quickly, GPUs run at a comparatively lower clock speed but use a large number of threads to perform many operations in parallel.

While many ABM models and systems are available for the CPU, researchers are also creating ABM systems that can take advantage of the parallel nature of the GPU architecture. This is due to the fact that GPU has proven to be a good fit for the running of ABM models where agents are independent and autonomous. The architectural benefits are shown in [68, 69] where, in optimal conditions, models can run 300 times faster when compared to serial implementation on the CPU. Chapter 2 provides further discussions on the topic of ABM and its use on CPU, GPU and HPC platforms.

However, due to the nature of its parallel architecture, the optimal use of GPUs present extra complexities in implementing these ABM systems and gives rise to many considerations that must be accounted for. Programming for the parallel architecture presents a paradigm shift when compared to the serial architecture. Strategies must be used to hide latency, ensuring that data can be sent to the processing cores fast enough. It is necessary to consider the memory access patterns of the GPU, how the memory is allocated and the transfer of data between the CPU and GPU.

The use of GPUs presents an opportunity for accelerating pedestrian ABM simulations in a fast, scalable and cost-effective way. This is especially important when there are many simulations of various configurations to run. These simulation configurations, for example, could be generated as part of the prediction and optimisation process in a DSS or other automated analysis tools. Any stochasticity in the pedestrian model compounds this problem as extra simulation runs of each configuration are needed in order ensure that the result is statistically significant. As the simulation configurations are independent, if each one can fit within GPU memory, each simulation can be run on a different GPU device without the need for domain decomposition techniques. However, a simulation configuration that is too small will essentially waste the computational capability of the GPU. A system is then required that can take advantage of

available GPU resource for simulation whether it resides on the same machine or over the network, a topic that is discussed in Chapter 5.

In order to explore these possibilities, a pedestrian simulation system needs to be built. One of the key parts of the pedestrian simulation system is the navigation. Having a navigation system allows pedestrians to exhibit complex flow resembling real world patterns. Considerations must be made between different approaches to navigation, whether the underlying data structure of each approach will affect the scalability of the system and architectural issues that will be faced in implementing the system on the GPU, a topic covered in Chapters 3 and 4.

The research questions in this thesis are as follows:

- Can a pedestrian ABM simulation system created for the GPU be made fast, efficient and scalable?
  - What are the essential components that are needed for a GPU based pedestrian simulation system?
  - What are the complexities faced when developing the system for the GPU?
  - What navigation system can be used in combination with the pedestrian simulation system in order to achieve this goal?
- How can the pedestrian ABM system cope with the problem of needing to run many simulation of various configurations?
  - What are the complexities involved in creating such a system?
  - How can the GPU be effectively utilised when running many small simulations?

1.2 Contribution to Knowledge

The novel contributions of this thesis are as follows:

- An investigation in to the use of GPUs to create a fast, efficient and scalable pedestrian simulation system. Chapters 3 to 5 highlights the complexity in the development of pedestrian simulation systems and its components, optimised for the GPU architecture.
- The use of discrete navigation agents to provide grid-based vector field navigation for pedestrian simulation. The method described in Section 3.1 fully adopts the ABM approach and so can be implemented in any generic ABM

framework allowing it to take advantage of any parallel architecture supported by the framework. The implementation on the FLAME GPU framework with a real-world scenario and its benchmarking shows that the approach can be applied to complex environments and runs efficiently. The work was published at TPCG 2010 [70].

- The use of searchable graph-based navigation for pedestrian simulation based on the work of Pettré et al. [47]. The method outlined in Section 3.2 addresses the problem of a pedestrian's possible route-change behaviour during congestion by providing a searchable lookup graph for re-routing and also fully utilises the whole environment. Examples are given to show that the approach is scalable and can be efficiently applied to GPU architectures. The work is currently being prepared for publication. It was developed and used in the following projects:
  - "High Performance Pedestrian Simulation on the GPU" an EPSRC Pathway to Impact grant by Prof. Daniela Romano and Dr. Paul Richmond in 2012.
  - PACE, a prototype high-performance pedestrian simulation software as part of an Early Career Researcher Grant for Dr. Paul Richmond in 2012.
  - Concoursia, a protype high-performance pedestrian simulation software developed as part of the Advanced Computing Research Centre headed by Prof. Mike Holcombe and funded by HEFCE between 2014 and 2016.
- A description of how GPUs were used in the creation of Concoursia, a prototype pedestrian simulation system that uses the searchable graph-based navigation system proposed in Section 3.2 and provides a GUI making the system more usable for general users. The software, described in Chapter 4 has been used to model scenarios including a shopping mall and a train station, and metrics can be collected to perform post-simulation analysis. The software was developed in collaboration with Dr. Mark Burkitt for the Advanced Computing Research Centre headed by Prof. Mike Holcombe, funded by HEFCE between 2014 and 2016.
- A description of how GPUs were used in the creation of a prototype pedestrian multi-simulation system, covered in Chapter 5, that can perform multiple simulations concurrently with differently seeded starting configurations on a network of machines equipped with GPUs. Multiple simulation instances are batched on a single GPU to provide higher efficiency and metrics are collected from these batched simulation using a parallel GPU optimised process. The work was

part of an externally funded project by BAE Systems to create a "Simulation Decision Support Tool" headed by Dr. Paul Richmond in 2011. The work has been published as a short paper at TPCG 2012 [71].

1.3 Thesis Structure

Chapter 2
Chapter 2 provides background literature on pedestrian simulation with the emphasis on navigation and use of pedestrian simulation as a DSS. An overview of ABM is given and how most modern day pedestrian simulations adopt the approach. The different systems available for implementing pedestrian navigation behaviours are discussed. The use of parallel computation architectures, HPC and GPU, are discussed based on their importance for obtaining simulation results in acceptable time. The chapter discusses the use of simulation software and frameworks for the purpose of decision support, providing a detailed review of the FLAME and FLAME GPU simulation framework which is used in all of the work throughout the thesis. An overview of FLAME's template based system is provided and how the frameworks are implemented for the HPC and GPU platform. Comparisons with benchmarks are made between the two platforms and the advantages and limitations of each platforms are discussed.

Chapter 3
Chapter 3 describes the implementation of two navigation systems for pedestrian simulation. The first approach uses a grid of agents to provide navigation functions which integrates fully into the ABM approach. The second uses graph based navigation which provides better scalability in larger environments with more complex pedestrian agent goals. Benchmarks are provided for both approaches and a discussion outlines advantages and limitations for each approach.

Chapter 4
Chapter 4 describes Concoursia, a prototype pedestrian simulation software for use as a DSS. It is based on the FLAME GPU framework and provides users with a graphical user interface for preparing and running the simulation. The different parts of the software are discussed including the simulator, GUI interface, the agent model used and the metrics that are collected for analysis. Two simulations based on real-world environments are used to show suitability as a DSS and provide benchmarking for the software.

Chapter 5

Chapter 5 describes a prototype pedestrian simulation system that is able to run multiple simulations in parallel over a network of machines equipped with GPUs. The system uses batching to group multiple small simulation together for more efficient processing and manages simulation jobs over a network. The methods to collect metrics efficiently from multiple simulation instances on a single GPU are explained. An evacuation scenario is used as a basis for the model in the system and provides the benchmarking.

Chapter 6

Chapter 6 provides the conclusions to the body of work in the thesis, outlines its limitations and gives suggestions for future work.

Chapter 2

Related Work

This chapter provides an overview of pedestrian Agent-based Modelling (ABM). This includes pedestrian ABM software and frameworks, Decision Support System (DSS) for ABM and parallel hardware platform suitable for accelerating pedestrian ABM. The chapter begins with an introduction to agent-based modelling and simulation in Section 2.1. Then, Section 2.2 covers the background on modelling and simulation of pedestrians with a particular focus on ABM including local motion (Section 2.2.1) and navigation (Section 2.2.2). Section 2.3 then provides an overview of the applications of pedestrian modelling simulation, including the tools and frameworks that provide a better workflow for creating a simulation scenario and give results that makes it easier to perform analysis. Section 2.4 discusses the use of pedestrian simulation in a decision support system. It explains the need for multi-simulation to explore the vast problem space when applying optimisation and the need for a pedestrian tracking system (Section 2.4.1) for providing data about the system's current and historical states. Section 2.5 investigates the parallel computing architecture that can be used to accelerate pedestrian simulation, namely the use of Graphics Processing Unit (GPU)s (Section 2.5.1) and High Performance Computing (HPC) (Section 2.5.2). Section 2.6 explores the detail of the FLAME framework (in particular FLAME GPU) which has been used extensively in this thesis as the base platform for the implementation of the pedestrian models. Both HPC and GPU versions are covered. Finally, Section 2.7 provides a summary of the chapter.

2.1 Agent-based Modelling

ABM is a microscopic modelling approach that focuses on the simulation of many interacting agents. Agents are heterogeneous individuals that can act independently and autonomously using their own set of behavioural rules. They are given the ability

to perceive and react to the environment and to also communicate with other agents. It is the resulting emergent systematic behaviour from these interacting agents which is often of interest to the modeller, for example the modelling of pedestrians to observe flow and congestion patterns [1, 3, 4, 72] or the modelling of different financial institutions to observe the effect on the economy [73]. ABM provides a bottom-up alternative to the top-down Equation-based Modelling (EBM) [38] approach where a simulation consists of sets of ordered differential equations. Its individualistic representation makes many types of behaviour easier to formalise and construct [74]. Observed higher accuracy in simulation using the ABM approach is also due to its finer level of representation when compared to the aggregate behaviours found in EBM [38, 75]. ABM models are, however, much more computationally expensive than EBM as real complex systems can involve the simulation of thousands or millions of agents, e.g. pedestrians in a train station or cells in a small part of a human body. It is the rapid improvement of computational power and, more recently, the use of specific hardware with parallel computing architecture that allows the simulation of these models in an acceptable time-frame (Section 2.5).

The earliest example of ABM is Cellular Automata (CA), a discrete model consisting of a grid of cells with each one in a finite number of states where each cell can interact with surrounding cells. An example of a well-known CA model is Conway's game of life where each cell is "alive" or "dead" according to the state of their neighbouring cells, producing different grown pattens over time depending on initial conditions and propagation rules [76].

The seminal paper by Reynolds [77] on the flocking behaviour of birds is one of the first models with social characteristics. The behaviour can be described as three simple rules. An avoidance rule stops animals from colliding with each other by moving away from individuals closer than a certain distance. An attraction rule keeps group close together by having individuals move towards the centre of the flock. Finally, an alignment rule averages out the group's travel direction to make sure the flock moves together. These bird agents only required local perception of the environment to perform this coordinated movement. They also occupy 3D continuous (Euclidean) space instead of the discrete space used by CA models.

2.2 Pedestrian Modelling and Simulation

The term pedestrian modelling in the context of this thesis refers to the representation of human pedestrians using the ABM approach and pedestrian simulation refers to the simulation of these pedestrian models. Each individual pedestrian is an agent that can

interact with other agents. Such agents exist within a shared physical environment and can move around while constrained by it. In addition, each pedestrian is able to navigate, routing around walls and obstacles, from one location of the environment to another. These two behaviours, local motion (Section 2.2.1), and navigation (Section 2.2.2) form the basis of a pedestrian model. Pedestrian simulation has been applied to various fields [78] and major uses can be found in architecture and urban planning [5, 6, 9, 72, 79, 80], safety [3, 4, 8, 44, 81–86], and media and entertainment [87–98].

The use of pedestrian simulation in architecture and urban planning is concerned with the designing of an optimal space. In the context of pedestrian simulation, this refers to spaces designed with enough capacity, good pedestrian flow and minimal congestion. When a number of pedestrians are navigating and interacting, emergent systematic behaviours such as flow patterns or congestion can be observed and undesirable features can be negated with various interventions. These inventions can be changes in the design of the physical space, introduction of barriers, usage of signs and other means of communication or limiting the number of people that can enter in the first place in order to ensure optimum flow rate and avoid blockages and congestion. As an example, for predicting future capacity, Hoy et al. [72] used the MassMotion [30] software to simulate the Union Station in Toronto. The model was calibrated by undertaking two surveys in a similar period to count passengers at every entrance and exit, using existing pedestrian volume data and train schedules. Their work predicted that the station will be over-capacity when there is over 10% more passengers which are the numbers projected for 2031. To improve flow and reduce congestion, Nana et al. [5] applied simulation to form an optimal egress plan from the stadiums in the 2018 Beijing Olympic games. Lili Lu et al. [6] tackles the problem of flow capacity by using simulation to find the optimal pedestrian signal crossing width in relation to expected demand. Yue et al. [7] uses pedestrian simulation to analyse the pedestrian flow and congestion for the Shanghai Expo 2010 to suggest re-location of entrances of problematic exhibits away from the major corridor and removal of landscape structures, e.g. fountains in busy areas in order to reduce the congestion. The pedestrians in the models used in the field of architecture and urban planning are expected to be in their normal behaviour. This means the behaviour can be observed directly, in other similar environments or within a controlled study [21, 99–102]. In fact, algorithms used within the pedestrian models are derived from these behavioural observations, for example the individual's behavioural rules such as interpersonal distance, grouping, queueing and rushing. At the systematic level, emergent self-organisation behaviours can be observed, such as lane formation when there's pedestrian flow in opposite directions or oscillation of passing direction at narrow doorways.

In the field of safety, evacuation models are created to assess whether people are able to evacuate from a space within reasonable time. Evacuation from fire is a major topic of research and such fire evacuation models can also combine the simulation of fire and smoke [8, 82, 83, 85, 86] to affect the movement and decision-making of evacuees. Evacuation models are also used for investigating occurrences such as terrorist attacks [3], and natural disasters such as flooding [44] or tsunami [84] where an evacuation of a large portion or entire city needs to be considered [85]. These evacuation models can then be used to feed into the design of architecture or in crisis training and management where optimal strategies can devised. These strategies can be in terms of evacuation timing, chosen route, gathering point, etc. [4].

As lives are concerned, pedestrian models related to safety must meet stringent Verification and Validation (V&V) tests. One such test specification was created by the National Institute of Standards and Technology (NIST) [103]. The verification tests ensuring the model has the expected behaviours are broadly divided into five categories which are pre-evacuation time, movement and navigation, exit choice and usage, route availability, and flow constraints. The pre-evacuation time test involves testing for the length of time until the pedestrian starts to evacuate. The movement and navigation test checks for the speed in various situations such as walking through a corridor or stairs. It also tests for things such as movement around a corner and group behaviours. The exit choice and usage test checks that pedestrians can move to the allocated exits, that social influence affects the choice of an exit route, and that a pedestrian's familiarity of the environment is taken into account. Route availability checks that pedestrians are able to respond to exits closing dynamically and choosing a different route. Lastly, the flow constraints test checks that slowdown caused by congestion matches the expected pre-measured rate. The validation tests are then used to compare the model's evacuation times to various small and large scale evacuation scenarios.

In media and entertainment, examples of pedestrian simulation can be seen in many movies and games [104, 105]. Their uses range from simply populating the world to give a scene a higher sense of believability or realism to being used as characters in their own right by representing large gatherings or clashing armies [104–108]. Interactivity is also possible in the game media such as war games or god games where players can control large armies or movements of large amount of people e.g. the Total War franchise [109]. In this visual medium, instead of statical accuracy in crowd behaviour, it is the visual quality or visual believability of the crowd that takes precedence. One of the ways to improve the visual quality of the crowd is to introduce variety to the human models [87–89]. This variety can come from appearance, such as face, gender, body type, ethnicity, clothing, or accessory. A study by McDonnell

et al. [98] shows that the aspect of visual variety is the most important to focus on. The other factor that can add variety is in the motion of a human such as walk gait or gestures. Increasing visual fidelity of individual humans can also be applied, such as approximating subsurface scattering for skin rendering [110]. For interactive application such as video games, it is also necessary to render the crowds in real-time. Techniques such as Level of Detail (LOD), billboarding, instancing or a combination of these techniques can be used to reduce the amount of processing required to render the crowd at the time it is presented [111–114]. There is also the interactive control of crowds whether it be in the control of their appearance [115] or in their movement as a group [90–97].

2.2.1 Local Motion

Pedestrian local motion consists of a set of rules or behaviour that applies to a pedestrian's motion over a short distance or within their perception range. They can be reactive rules such as a pedestrian avoiding other pedestrians and obstacles, or active rules where individuals form groups and move together.

The way a pedestrian's position in the environment is represented has an effect on what local motion rules can be applied. Broadly speaking, there are three categories for representing a pedestrian's position: coarse, fine, and continuous network. The network in this case refers to a network of inter-connected traversable areas.

With coarse network models [116, 117], multiple pedestrians can occupy a single node, for example an entire room. The edges of the network then represent the connectivity between the nodes, for example, which rooms are connected and thus traversable. As no exact location is represented, it is not possible to apply local motion rules.

Fine network models [27] are akin to cellular automata where the space is represented as a discrete grid where each cell is normally the size of a single pedestrian. With fine network models, it is possible to apply local motion rules such as collision avoidance or grouping. In models such as [24], where each cell can be occupied by only a single pedestrian, should multiple pedestrian want to occupy the same cell at the same time, an algorithm is needed which decides on which pedestrian has priority. Burstede et al. [23] introduced the concept of 'floor fields', a grid of the same dimension but separate to the one that pedestrians occupy. The 'floor field' grid can exist in discrete or continuous space. The field is informed by the location of pedestrians, and has diffusion and decay effects creating a field of gradients. By following this field, pedestrians are able to seek out the cell they can move to in order to achieve optimal flow. It was observed that this approach shows similar self-organising behaviour as

found in the social force model. Whereas Burstede et al. [23] stores probability within the floor fields, Charibi et al. [29] records the repulsive influences of nearby objects.

Continuous network models [21, 25, 26, 118], where pedestrians exist in continuous space, can allow for more accurate trajectory and velocity as movements do not have to be discretised to grid cells. While the dimension of the pedestrian is limited to the size of the cell in the fine network model, there are no such restrictions in the continuous network models. Space utilisation by the pedestrians can be made more realistic during congestion as Alonso-Marroquin et al. [119] showed by the use of spheropolygons to map a pedestrian's contours. These are used for collision detection that more closely matches to the profile of a real human. The use of continuous space however, means a more complex approach to local collision avoidance is required to avoid overlapping of pedestrians.

It is also possible to combine the various network types [85, 120], for example using a continuous network to represent areas with the highest congestion (e.g. doorways) or more geometrically complex layouts, and a coarse network for a lower congestion area or one with less complex geometry.

Continuous Network Models

While continuous network models presents a bigger challenge in the formalisation of the local motion equations, the increased fidelity in the representation of pedestrian movement and space occupied have proved attractive to researchers.

One of the earliest continuous network models is Helbing's [21, 118] social force model, which is a socio-psychological force based model where movements are resolved through a set of force equations. It includes a goal directed force that varies according to the time needed to arrive at the destination, e.g. a pedestrian in a rush will have a higher desired speed. Each pedestrian has their own personal space, an ideal distance away from other people, which is translated into a repulsive force inversely proportional to the distance from others. The repulsive force stops pedestrians from walking into each other. A separate repulsive force is also applied to the border of buildings as pedestrians prefers to keep a certain distance from walls [100]. A similar but opposite force attracts pedestrians to things of interest such as friends or a street performer. This force can be used to keep groups of pedestrians together and may diminish with time. The model has been shown to exhibit self-organising behaviour observed in real pedestrian traffic such as lane formation and alternating flows at congested doorways [118]. In order to prevent numerical instability from large forces generated from the contact between pedestrians or obstacles, it is necessary to reduce the time step of the simulation according to a maximum velocity change threshold.

The variable time step in practice means that simulations take longer to perform the more congestion there is. The social force model was later extended to include crushing forces experienced during congestion in emergency situations [22] with the addition of a 'body force' for counteracting body compression on contact with other pedestrians or obstacles and a 'sliding friction force' for impeding tangential motion so that pedestrians don't 'slide' between each other as congestion increases.

During times of high congestion when pedestrian movements are restricted by others surrounding them, researchers have observed striking similarity to the movement of fluids [121] and, while Helbing et al. [118] suggested that it is more flexible and practical to model pedestrians as individuals, Hughes [122–124] shows that "thinking fluids" can also be applied for simulating dense crowds such as the Hajj pilgrimage and other scenarios.

Seer et al. [125] later on showed that while the social force model is good for representing the macroscopic characteristics, e.g. flow-density relations, it does not represent movement trajectories well at the individual level. An experimental study by Moussaïd et al. [101] further expanded the social force model by applying the observed avoidance behaviour where pedestrians simply adjust their motion for side-on encounters, but with head-on encounters pedestrians make a binary decision to either make an avoidance manoeuvre to one side or the other.

In Moussaïd et al. [126], perception information was used to provide additional cognitive heuristics. For example, in pedestrian avoidance, a visual model is used to provide distance to obstacles. The visual model works similarly to human perception in that obstacles in the background can be occluded. This means instead of adding on repulsion forces from every nearby obstacle in the social force model, a pedestrian agent is only reacting to the visually perceived obstacle. Moussaïd [127] later elaborated on the cognitive heuristic model stating that while it solves the problem of how to combine multiple simultaneous interaction, it is more difficult to formalise and extend behaviours than force-based models.

Baglietto et al. [28] offers an alternative collision avoidance model that avoids the force calculations by adding a contact rule that slows down or stops pedestrians should their trajectories overlap. The approach avoids the calculation of force models and allows for larger time steps to be used during simulation.

Velocity-based models that use the concept of velocity obstacles by Fiorini et al. [128] offers another alternative to the forces based models. Originally envisioned for local path-planning for robots, this approach has been adapted for many pedestrian simulation models [31–35]. The approach works by finding the sets of velocities that, if taken, will result in a collision with obstacles. This velocity-based approach allows

the use of larger time steps but the avoidance calculations are more computationally expensive compared to each step of the force-based approach.

Van den Berg et al. [33] extended the velocity obstacle model by presenting the reciprocal velocity obstacle approach which takes into account the fact that other pedestrians will be performing the same avoidance manoeuvre and so can avoid oscillations in movement resulting from velocity re-adjustments when incorrectly assuming that the other agent is a non-reactive dynamic obstacle. Van den Berg et al. [129] later introduced Optimal Reciprocal Collision Avoidance (ORCA) which reduces the problem to solving a low-dimensional linear program and uses a KD-tree for storing static obstacles. Snape et al. [130] provided an example for the new approach by using it for collision avoidance between characters in video games.

Wolinski et al. [36] presented a stochastic context-aware location motion prediction model. It uses a collision probability field to represent future action of other pedestrians where gradient descent can then be used on the field to find an optimal path. Unlike Treuille's [37] continuum crowd approach that operates at the macroscopic level, the field is generated for each individual pedestrian and is applied only to the range of local motion. The forward planning of motion, informed by other pedestrian's motion and the environment layout, allows the pedestrian's trajectories to show higher levels of self-organisation with less deadlock and backtracking as was compared to the social force and ORCA model but at the cost of having the highest computational cost of all the approaches mentioned.

Other Local Motion Behaviours

Other local motion behaviour has been observed. Some are relevant to specific case, e.g. falling over during evacuation but many others are relevant to more general pedestrian models, e.g. grouping and leading behaviours. Moussaïd observed that pedestrian groups walk in a 'V' formation to maximise social communication [102]. HiDAC [131] have added features where pedestrians can fall over and become an obstacle during evacuation. Qui et al. [132] and Murakami et al. [133] showed that group formation by using a leader-follower model does affect the pedestrian flow. Ondřej et al. [134] gave pedestrians a synthetic vision-based system for collision avoidance which gave more natural collision avoidance behaviour than other methods but at the cost of much higher computational cost as each pedestrian's field of vision has to be separately rendered and processed. Pedestrian simulation packages such as Exodus [60] incorporate sociological factors such as age, sex and running speed. Other dangers such as smoke, fire and heat, as simulated in STEPS [135] and EXODUS [60] can restrict movements and block evacuation paths. Yuan et al. [136] explores

an integrated network approach which divides the evacuation spaces into different zones consisting of coarse and fine environment nodes in order to increase simulation performance. Lemercier et al. [137] added high-level behavioural rules for groups for following and avoidance behaviour of groups of pedestrians.

2.2.2 Navigation

In addition to local motion behaviours, navigation is an essential component of a pedestrian model. Inter-pedestrian and environment interaction gives rise to the emergent systematic behaviour that can be observed and analysed, such as flow or congestion patterns, in order to inform the design and planning of an environment. To obtain the correct systematic behaviour from the model, it stands to reason that pedestrians in the model should be given goals and choose a path through the environment to achieve the goal in a way that closely resembles their real-life counterpart. These goals can be as simple as moving from one location to another or more complex such as "catching a train". The locations that pedestrians must travel are often obscured by walls or lie far outside their range of perception and so they must be directed by a system with a global view of the environment.

A navigation system turns pedestrian's goals into a set of objectives with a destination location within the environment and provides routes for reaching the destination. For example, should a pedestrian's goal be to exit a building, the navigation system would mark exit doors as destination locations and provide the routes for reaching them. This routing information can then be shared between pedestrians with similar objectives. Additionally, in circumstances such as high congestion or blocked pathways, pedestrians may be required to make minor route alterations or choose a different route altogether.

The systems for generating global paths have for a long time been an active research are in Robotics and Artificial Intelligence [138–140] and more recently applied to the planning of paths for virtual characters [141, 142]. Many algorithms have emerged for finding an optimal path according to heuristics such as distance [143]. Dijkstara's algorithm, for example takes a greedy approach that finds an optimal route by considering all routes possible to reach the destination. The A* algorithm on the other hand reduces the search space by searching for what looks to be the most 'optimal' direction first, according to heuristics [144]. When compared to the motion planning of robots [145], the pedestrian global path planning problem is limited to only 3 and sometimes even 2 degrees of freedom which simplifies the problem significantly. The complexity introduced when used with a pedestrian ABM approach is the fact that paths must be found for every pedestrian agent in the environment.

Navigation systems can be broadly divided into grid and graph-based approaches. The grid-based approach [45] discretises the environment into fixed sized cells which can be encoded with navigation information such as obstacles occupying the cell or terrain height. Most commonly the cells are used to hold a directional force. A grid of these cells is called a vector field (also known as force fields or flow fields). Multiple fields can be used for each group of pedestrians to represent differing goals and introduce complex flows. The approach is very effective when combined with good local collision avoidance and computationally very efficient as access to navigation information such as the pedestrian's location can be mapped directly to the grid coordinates. The authoring of vector fields can either be done manually using software that allows a user to 'paint' the field directions [146], or alternatively they can be automatically generated using algorithms such as wavefront propagation where the origins of the waves represent the destination(s) of the pedestrians [147]. The approach requires a dense data-structure leading to exponential memory requirement with the environment size. Each grid can only represent a single objective (e.g. go to the nearest exit) so multiple grids are necessary to represent the full range of individual objectives further increasing the memory requirement.

The flow tiles concept introduced by Chenney [148] makes the process more scalable by re-using tiles of pre-defined vector fields to save on repetitious regions of flow that are likely to occur within the environment. Work by Jin et al. [149] presents the use of continuous vector fields constructed using radial basis functions. Anchor points are used in order to define directional flows of pedestrian agents and can be placed dynamically on the scene making crowd control interactive. Since the field is continuous, performance is dependent on the number of anchor points rather than the size of the environment and has been shown to scale well with increasing pedestrian sizes. Jin et al. does not, however, discuss how well it would apply to more complex environments and goal-based situations.

With the graph-based approach, areas of the environment can be represented by a set of nodes in a graph while edges mean the connected areas are traversable [48]. Routes through the environment are represented as a sequence of nodes which can be found by using path-finding algorithms such as A* or Dijkstra. Sparse representation of the physical environment means memory consumption is not directly proportional to the environment size but instead its complexity. Node sequences are also a memory efficient form of representing each pedestrian's individual objectives.

Instead of planning a path specific to each agent [49, 145, 150], modern approaches use a strategy for space sub-division for creating a general collision-free route through the environment. Wein et al. [50] creates a Voronoi graph of the physical environment with added visibility rules to generate more natural looking paths. Sud et al.

[51, 52] applied the Voronoi space subdivision concept to dynamic environments by representing the environment as a series of points elastically connected to each other. Connections bend and break as obstacles pass through these connections. Directly integrating dynamic obstacles into the graph ensures optimal routing around them. However, as the map could be constantly changing, paths must be recomputed which may affect performance in very large environments or where there are large number of pedestrians with different objectives.

The concept of navigation meshes [53] where the environment is divided into a set of convex shapes has been widely used in gaming and other real-time applications. The use of a bounded area rather exact paths means it can be used by more than one character. It is also possible to pre-generate the navigation information [54].

Toll et al. [55] used a mesh-based approach for multi-level navigation by first generating a medial axis [151–153] path for each of its levels each existing on its own 2D plane. The medial axis is then modified with the connection between floors added through the pre-annotated connection points. The advantage of the approach is the fact that pedestrians can use the navigation mesh to bound their walkable areas. The concept was extended further to provide real-time replanning [154] by adding a pruning rule to the standard A* algorithm.

Barnett et al. [56] presents a method for the interactive control of crowds where the Reeb graph [155] is used for representing the environment. Hilaga et al. [156] showed that the use of Reeb graph has been shown to have lower computational cost and is less sensitive to noise and small undulations than the medial axis approach.

Instead of generating navigation graphs straight from polygonal information, some methods opted to voxelise the environment to simplify the process of boundary generation [47, 157, 158]. It is robust approach that works well for very complex environments with many un-optimised polygons and non-convex obstacles. In the case of Pettré et al. [47], OpenGL is used to quickly render slices of the environment.

In the work of Pettré et al. [47, 114], walkable area in the environment is covered by connected circular areas, termed 'Disk Graphs' or 'Circular Clearance Graphs' [54]. Sequences of these areas form routes through the environment which are also converted to rectangular corridors that serve to bound the space. The system is efficient for its intended purpose of pedestrian navigation for visualisation purposes but the rough bounding of the environment means the space is not accurately utilised during heavy congestion. Additionally, pedestrians have fixed pre-calculated routes through the environment which does not account for minor route-changes required to avoid congestion. Yersin et al. [159] directly tagged the graph nodes with high-level semantic information (e.g. the node is a park or a hotel) making it easier for users to specify pedestrian objectives through the GUI.

Gloor et al. [57] presented a hybrid approach where graphs are used to indicate a general path combined with the use of potential fields in areas that require more detailed navigation information. This concept is extended by Kneidl et al. [58] where the graph is used to find alternative route choices to represent different pedestrian behaviour according to the familiarity of the environment. The approach relies on the fact that large sections of the environment are sparse enough to not require the use of potential fields. Should the environment be uniformly dense the advantage of this hybrid approach is negated.

Treuille et al. [37] presented an alternative approach, the continuum crowd model, which is not based on ABM. Here, local collision avoidance and global navigation is resolved together using grids of potential fields. The shortcoming of the approach is similar to grid-based navigation in that it is not scalable to simulations with very large environments or one that requires pedestrians to have a large number of navigational objectives. Shopf et al. [46] dealt with the large environment problem by using a lower-resolution field and augmenting the collision avoidance behaviour based on the velocity obstacle model.

Navigation at its most fundamental level is a path-finding problem, a way to find an ‘optimal’ route through an environment according to certain heuristics. Taking the shortest route possible to a destination is certainly an important heuristic [160, 161] and has been observed to create ‘desire lines’, worn trails resulting from pedestrian taking short cuts outside of planned paths [100]. There are also many other heuristics to consider such as safety factors, aesthetics, path-width, congestion, number of turns, elevation, stairs, signage and whether a path goes back on itself [160–162]. Armeni et al. [163] observed when tracking pedestrian journeys, that they often choose a route which is longer but is able reach their destination with the same amount of time or faster as there are fewer obstacles and traffic. Golledge [162] showed that even with the same origin and destination specified, pedestrians may take different routes on the return trip.

2.3 Pedestrian Simulation Systems and Frameworks

Many software systems specific to the domain of pedestrian simulation have been created for the purposes of research and commercial use [59, 164]. Should the existing systems not be suitable, generic ABM frameworks can be used as a basis for the implementation of a pedestrian simulation system. This section gives an overview on the available microscopic pedestrian simulation systems and ABM frameworks widely used in research and commercial sectors.

A table listing a number of pedestrian simulation systems can be found in the review of evacuation models by Kuligowski et al. [164]. While the table was compiled from the perspective of pedestrian evacuation, many systems listed can be used for wider range of scenarios, such as simulating the normal running of a facility [26].

In terms of availability, some systems are offered commercially either as a software package or through consultation such as Exodus [60], Legion [26], MassMotion [30], Urban Analytics Framework [61], Pathfinder [165], Simulex [166], Simwalk [167], and STEPS [135]. Other systems are available as open source software or showcased as part of scientific literature such as EPT [168], FDS+Evac [25], PEDFLOW [169], Gridflow [170], or MASSEgress [171].

In addition to the basic pedestrian model, these systems often provide additional pedestrian behaviours in addition to local collision avoidance and basic point-to-point navigation. Some systems [25, 165–167] are partial behaviour models that can include implicit behaviours such as unique occupant characteristics, overtaking behaviour, and the introduction of smoke or smoke effects to the occupant. Other systems [26, 30, 60, 61, 135, 168, 169] have behaviour models that explicitly simulate behavioural actions and decision-making. The result of the decision-making can also depend on the condition of the surrounding physical environment.

Various approaches are taken by these systems for representing the local movements of the pedestrians. Some systems use fine network representation [135, 168] while others use the continuous network approach [25, 26, 30, 165–167, 169, 171]. Systems such as buildingExodus [60] and Urban Analytics Framework [61] however, uses a mixture of coarse, fine and continuous network models.

Other than the pedestrian simulation, the systems also provide additional features. FDS+Evac [25] has a built-in fire simulator. All the aforementioned systems can implicitly, or with the help of a plug-in, import environments from external authoring tools (e.g. from a CAD file). The also offer visualisation of the running simulation and output of metrics for evaluating the simulation.

It is also possible to use a generic ABM frameworks to implement a pedestrian simulation system. Some of these frameworks can be found in a review by Railsback et al. [172]. There are frameworks available that takes advantage of the HPC architecture. Lohner et al. [2] used CPU-based HPC for faster than real-time simulation of over a million pedestrians. It uses an optimised PEDFLOW model [173] where the environment bounding areas (domains) that have been created by Delaunay triangulation are used to effectively define the communication boundary. Other general ABM frameworks that can utilise the HPC architecture can be seen in the survey by Rousset et al. [174] and these can be adapted for the simulation of pedestrians. The survey also provided benchmarking for D-MASON [175], RepastHPC [176], Pandora [177] and

FLAME [39] frameworks. As the FLAME GPU framework is used extensively in this PhD, a review of the precursor FLAME framework is provided in Section 2.6.2.

Case studies are available that show the wide range of uses for pedestrian simulation and highlight the multitude of data sources that must be taken into account and the steps taken to verify and calibrate a pedestrian model. Cao et al. [1] used the AnyLogic [178] software for suggesting design optimisation of the Nanjing Rail station. A database of passenger behaviours was created by on-site observation and video tracking. From the simulation, the author identified various problems such as improper location of VIP areas and that there were a lack of enquiry desks. Multiple design solutions were suggested and implemented with success according to the author.

Galea et al. [3] investigated the evacuation of the World Trade centre using the buildingEXODUS software the architect's plan obtained from the National Institute of Standards and Technology. The investigation was to try and identify the likely outcome of 'what if' scenarios and suggests that most pedestrians that survived the initial shock could have escaped should there be a set of stairways intact from top to bottom.

Veeraswamy et al. [4] extended the model in buildingEXODUS and applied it to a rural and urban evacuation by including road networks, buildings and open spaces. When applied to the scenario of Swinley forest fire in 2011, with fire spread information coming from the Prometheus tool [179], it suggested an optimal scenario which was counter-intuitive in that it resulted in maximum assembly time and longest distance travelled.

Hoy et al. [72] used MassMotion [30] to simulate the Union Station in Toronto. The model was calibrated by undertaking two surveys in a similar period to count passengers at every entrance and exit, using existing pedestrian volume data and train schedules. The paper predicted that the station will be over-capacity when there is over 10% more passengers which are the numbers projected for 2031.

Zia et al. [44] used Repast HPC software to model the evacuation of an entire city with 200,000 pedestrian agents during flooding. It uses a combination of GIS and other data to create a simplified CA grid of static agents that represent navigable areas of the city and pedestrian agents are overlaid on top of the static environment agents. The pedestrian agents are a combination of ones that can navigate with perfect information and ones that navigate by relying only on local perception. They are also social and will follow agents with better knowledge and can warn others about the closing of evacuation points (depending on the model strategy).

There are also many other applications such as redesigning a pedestrian cross-walk [6], modelling of World Expo 2010 in Shanghai [7], evacuation in the case of a subway

fire [8], optimise pedestrian flow at the the main stadium at the Beijing Olympic games [5] and tsunami evacuation [84].

2.4 Pedestrian Simulation and Decision Support Systems

A DSS is a tool that facilitates the analysis and decision-making process. In particular, it refers to a computer-based tool that can combine and analyse data from models, documents, databases, knowledge systems, etc. in order to identify problems and suggest possible solutions. These offered solutions can potentially be ranked according to a specific criteria identified by the user. DSS are extensively used in business and management [62–64] but the principles can be applied to any domain, for example clinical [180], agriculture [65], forest management [66] and transport [67].

Pedestrian simulation packages as mentioned in Section 2.3 can, in a sense, be classified as a form of decision support system, whereby reports according to a specified criteria are generated from the results of simulations and can be used to identify potential problems. The systems still require a degree of user-directed analysis along with their expertise to create designs and scenarios that can solve the identified problem.

It is this lack of automatic optimisation and exploration of the problem space that is still lacking with current tools and is very much an ongoing research problem. One of the difficulties lies in the range of optimisation problems involved to cover all cases. For evacuation, the usual problem is in finding the route that is fastest or safest for the pedestrian. Kneidl et al. [181] tackled the problem by using a dynamic navigation graph to direct pedestrians in various scenarios and running the simulation to test whether the evacuation time improved. Feng et al. [182] similarly used a network based approach but for reconfiguring the environment for more efficient crowd movement. Hoogendoorn et al. [183] in addition looks at shaping of pedestrian flow and distributing agents between multiple exits to reduce congestion. Bish et al. [184] looks at evacuation of a larger area at a more macroscopic level using different staging (households are told to evacuate at different times) and routing (households are told which route to follow).

At the design stage it may be that a change in the layout of the physical space is practical and automatic design optimisation tools using genetic algorithms or artificial intelligence could be incorporated. Multiple other intervention strategies could also be used to shape crowd flow, for example the introduction of safety staff, erecting and placement of barriers, use of dynamic signs, closing off an exit/hallway, delaying

arrival or public transport. All the mentioned interventions have associated cost and may have varying effectiveness depending on certain situations. Even if a way is found to objectively measure and compare all types of intervention, it is still necessary to run simulations of the scenario that explore the solution space and verify its efficacy. An example of this can be found in [4, 185] where multiple evacuations from a building and forest fire are generated, respectively, with each route evaluated by running a simulation for each one. Various stochastic behaviour can be introduced into the model which can include probabilistic route choice, probabilistic decision making, probabilistic emission of pedestrians, or the addition of noise to pedestrian movements. When stochasticity is introduced into the model, for each solution proposed, it is essential to run multiple simulations of the same scenario to obtain a good statistical distribution spread. It is then easy to see a need for a system that can perform these multiple similar simulations in parallel, and combine, analyse and rank the results within an acceptable time. The multi-simulation issue is a topic addressed in Chapter 5.

For a DSS to be able to run and continually forecast problems, it is also necessary to collect data about the state of the current system. In the case of pedestrians models, an ideal system would be one that can count pedestrians coming in to the environment, the path that they take through the environment, their activities in the environment and their ultimate destination.

2.4.1 Pedestrian Tracking Systems

As well as technical challenges of pedestrian tracking, there are also the addition of privacy concerns, safety requirements and practicalities of deploying tracking devices that can cover a large environment. Many public transport hubs in the UK employ a rudimentary form of pedestrian monitoring in the form of CCTV that is used to identify criminal activity but not for automated counting or tracking (to public knowledge). In UK rail, companies are employed to perform manual counting of passengers [186] and the count data is combined with Origin Destination Matrix (ODM) data obtained from rail schedules, ticket sales and gate information to give a more complete picture of station utilisation.

An autonomous pedestrian tracking system ideally requires that no special equipment needs to be carried on persons being tracked, is able to uniquely distinguish each pedestrian continuously through the environment and is able provide spatial coordinates and heading information at a good constant rate. Current pedestrian tracking research can be divided broadly into two categories, the type that uses RF signals from mobile/smartphones which are now almost ubiquitously carried on person, or vision-based systems.

The RF tracking approach can use one or a combination of mobile (e.g. GSM, CDMA, LTE), Wifi or Bluetooth signal that can be obtained from a mobile phone [187, 188] but, as the use of mobile signals requires the co-operation of service carriers, many approaches are limited to using only Wifi and Bluetooth signals, which have a low capture rate and are not suitable for precise localisation [189–194].

Computer vision-based tracking of pedestrian is an active area of research and, with the rise of deep learning, especially the use of convolutional neural networks, the algorithms can learn to recognise and classify images and videos more accurately than humans [195, 196]. The premise is to use video images captured by visible light or infrared camera to detect pedestrians and their movement in time through a sequence of video frames [197–200]. The approach is attractive in that it does not require pedestrians to be carrying any special devices and hence can potentially be used to track all pedestrians visible within the image. However, vision-based tracking does come with its own domain specific problems which are occlusion [197, 201, 202], i.e. when pedestrians overlap each other, detection in extreme crowd density, lighting conditions [203, 204] and continuity [205–207], i.e. re-identifying the same pedestrian when he/she has walked out of frame.

2.5 Parallel Computing Architecture for ABM

Computability poses a challenge in many scientific endeavours. Modelling of complex physical phenomena such as weather, fluid dynamics, molecular interactions, astronomical calculations, and engineering design are well known to require vast amounts of computational resource. Fortunately, the computation needed for running these models can be divided up and parallelised to a certain extent depending on the application. This has given rise to the field of HPC which is devoted to the use of supercomputers (e.g. a cluster of machines) and parallel processing techniques for solving complex computational problems. While the clock frequency of modern processors has remained stable over the last few years, the trend has been to add additional cores on a single chip thus increasing its ability to perform parallel computation. At the same time processors specifically designed with parallel architecture such as the GPU or the Xeon Phi have become commonplace in some of the largest HPC clusters [208, 209].

The same computability challenge is faced when simulating microscopic pedestrian models as simulating many individuals and their interactions requires a large amount of processing power, even more so in a system that is to be used for continuous monitoring and prediction where it is required to run multiple simulations concurrently

at speeds faster than real-time. The independent nature of pedestrian agents makes the models suitable for concurrent and distributed processing on the aforementioned hardware architectures.

The use of parallel hardware presents its own challenges in implementing inter-agent communication and synchronisation of agent states across multiple cores, devices, or nodes. Efficient utilisation of these systems, through load balancing and code optimisation is non-trivial, requiring expertise in the architectural constraints for each system and of concurrent programming paradigms. In order to reduce this complexity and need for architectural expertise, many generic ABM frameworks exist. The FLAME GPU ABM framework which is extensively used in the thesis is described in Section 2.6.

2.5.1 GPU Architecture for ABM

The GPU was originally designed to accelerate the real-time rendering of 2D and 3D graphics. With the introduction of a programmable pipeline and the CUDA parallel programming language, the GPU has become more accessible for use within research and industry. The use of GPUs can now be found in many research fields such as Medical Imaging [210], Computer Vision [211, 212], Robotics [213, 214], Machine Learning & Artificial Intelligence [215, 216], and in various types of physical simulation [217–221].

Rendering of a 3D scene is particular throughput intensive. A large number of independent matrix multiplication and floating point operations must be performed in order to render images at 25-30 frames per second. To give a comparison on pure arithmetic performance, it can be seen from [222] that the best GPU has a five to fifteen fold margin over an equivalent range CPU for theoretical single-precision floating point operation rates.

The need to perform highly parallel floating point operations is reflected in the design of the GPU hardware. Using an Nvidia GPU as an example, to enable high throughput each GPU contains multiple (tens to hundreds) Streaming Multiprocessor (SM) units with each one in turn composed of many Streaming Processor (SP) units for arithmetic operations, cache for data, texture and instruction, and warp schedulers. The warp is specific to the Nvidia GPU architecture and simply means a group of 32 threads that gets executed together. This SM configuration allows each one to run hundreds of threads concurrently. Single Program Multiple Data (SPMD) parallelism is used whereby each thread in the same warp is always executing instructions of a program in lock-step but with different register addresses so that the data being processed in each thread may be different. Should the program diverge, e.g. loops or

branches, the thread outside of the diverged instruction simply freezes until the code converges again.

In order to feed enough memory to the SMs, the GPU is equipped with fast on-board DRAM memory and a high bandwidth bus (hundreds of GB/s as opposed to tens of GB/s bandwidth between CPU and its own DRAM memory). The host (CPU) and device (GPU) is traditionally connected via the PCIe bus that operates at roughly similar bandwidth as the host CPU and memory. At the same time, the amount of expensive on-board memory is fixed and often smaller than the host memory, e.g. while the Nvidia Tesla P100 card has 16GB, it is not uncommon to see servers equipped with hundreds of GB. Memory management on the GPU is then something that must be taken into account. It is necessary to reduce the relatively slow transfers between the host and device as much as possible. When working with a dataset that does not fit on the GPU, it is necessary to plan an efficient way to stream data in and out of the device. When working with multiple GPUs on a single host, the memory transfer between GPUs must also be factored in. Since the Nvidia Pascal processor, with the introduction of 49-bit virtual addressing and page migration, a unified virtual memory feature allows a block of memory larger than the GPU's RAM to be allocated. The memory is automatically synchronised between CPU and multiple GPU devices. When the GPU tries to fetch memory that's not on-board, it can cause a page fault in order to migrate data from the CPU automatically. While the unified memory feature has alleviated some memory management problems, it is still necessary to understand the underlying memory transfer patterns and apply techniques such as memory pre-fetching to make the most efficient use of the GPU. To help solve the bandwidth issue, it is possible to obtain servers like the DGX-1 [223] that's equipped with NVLink interconnect which is around an order of magnitude faster than the current PCIe bus and can connect between GPUs and CPUs.

Ever since the introduction of the programmable graphics pipeline, researchers have been using the GPU for general purpose computing. It was possible to input and output data by way of geometry or texture and programming was done using the graphics shader. The process for converting existing algorithms to this workflow was complex and un-intuitive. The introduction of Compute Unified Device Architecture (CUDA), a C based programming language (also C++ later on), provided a way to do programming directly on the GPU. Along with CUDA, Nvidia also provided parallelised versions of commonly used libraries like Basic Linear Algebra Subroutines (BLAS) or Fast Fourier Transform (FFT) and, as of today, includes others such as parallel data structure primitives (Thrust) or neural network primitives (cuDNN). In addition, the ability to share CUDA memory with 3D graphics libraries (e.g. binding data to a texture) allows results to be visualised in real-time at little additional computational cost.

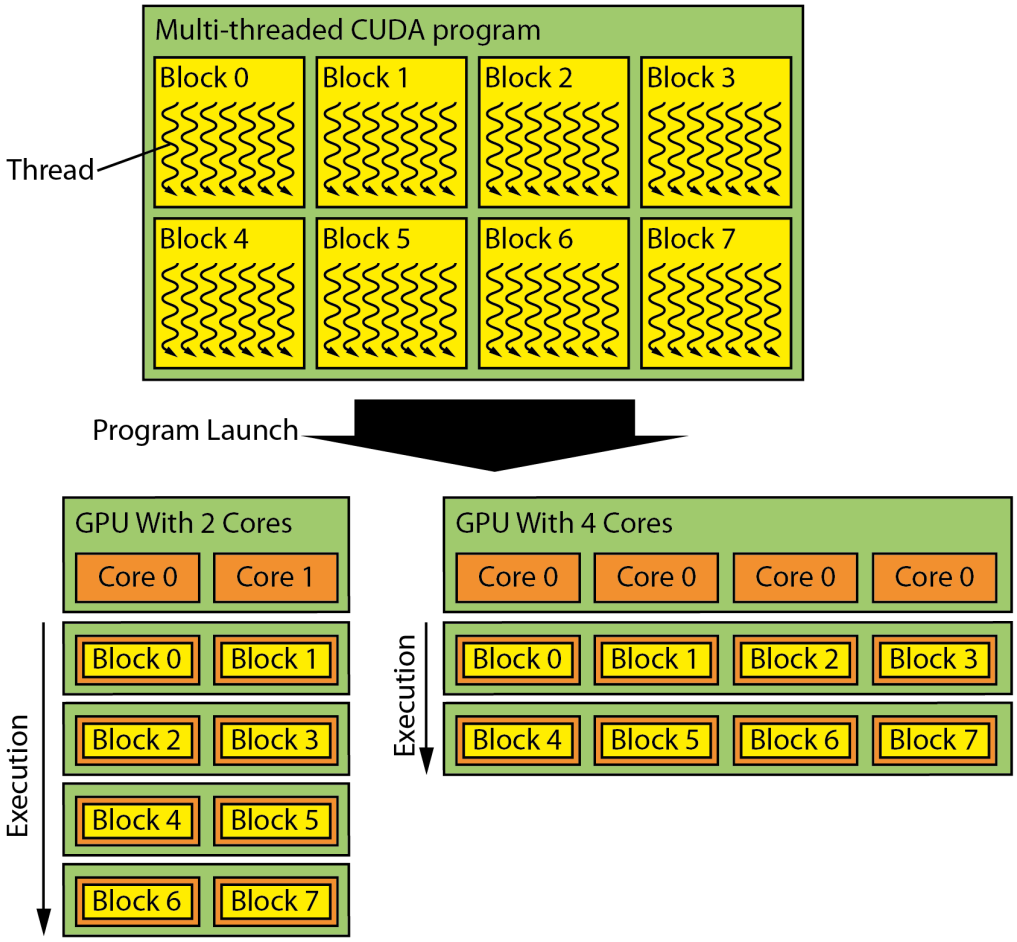

Figure 2.1 CUDA programs are launched as grids of blocks. Each block contain multiple worker threads. Threads in the same block are guaranteed to run on the same GPU core.

CUDA presents parallelism to users with the concept of launching a single program, a kernel, multiple times concurrently on a number of threads (Fig. 2.1). It uses the concept of blocks which contain multiple threads and the threads within each block are ensured to run on the same SM with shared access to the cache memory. This way, the execution of blocks can be divided up between all the available SMs and run concurrently (Fig. 2.1). Threads within a block have access to small amounts (less than 100KB) of cache memory which are the L1 cache, texture cache, and shared memory. The shared memory is the only cache that is directly programmable from within the kernel. Each thread, in addition, has access to local memory of 512KB, a global constant memory cache of 64KB, and global memory, which is the data that resides on the device's RAM, although this data access is much slower than shared memory.

The parallel memory access pattern on the GPU is different from a serial access pattern and this should be reflected in the structure of data that will be used on the device. When reading from memory, the GPU does not only read a single variable but loads a chunk of memory (128 bytes) to be cached. For serial data access, the Array of Structures (AoS) pattern is followed (Fig. 2.2 left), where the variables of the data structure are stored next to each other in memory. When accessing data in parallel, the use of the AoS pattern causes an interleaved memory access pattern as can be seen on the left of Fig. 2.2. This interleaved memory access means that the chunk loading strategy employed by the GPU is less effective as unnecessary data is read into the cache leading to more memory reads overall. The proper way to structure the data is to lay it out as Structure of Arrays (SoA) where there is only one object and the variables are instead laid out as an array. As can be seen on the right of Fig. 2.2, the memory read is coalesed. A single memory read will contain data from the same variable so there is minimal redundant memory reads.

As memory allocation on the GPU is relatively expensive, it is good practice to pre-allocate and re-use this memory. Also, as for CPU memory, it is more efficient to allocate single large blocks of memory at a time instead of many small blocks multiple times. The same applies when allocating memory in-kernel and by doing so (e.g. for the purpose of implementing dynamic arrays) each thread essentially makes a small memory allocation which can cause severe reduction in performance. This limitation essentially means that algorithms that require dynamic memory should be modified so that allocation and management of memory is performed outside of the kernel.

The ABM approach in simulating many heterogeneous individuals naturally fits well with the parallel processing nature of the GPU and so has been used in simulation such as representation of cancer growth [224, 225], tuberculosis [226], consumer behaviour in the insurance industry [227], road networks [228] and, of course, pedes-

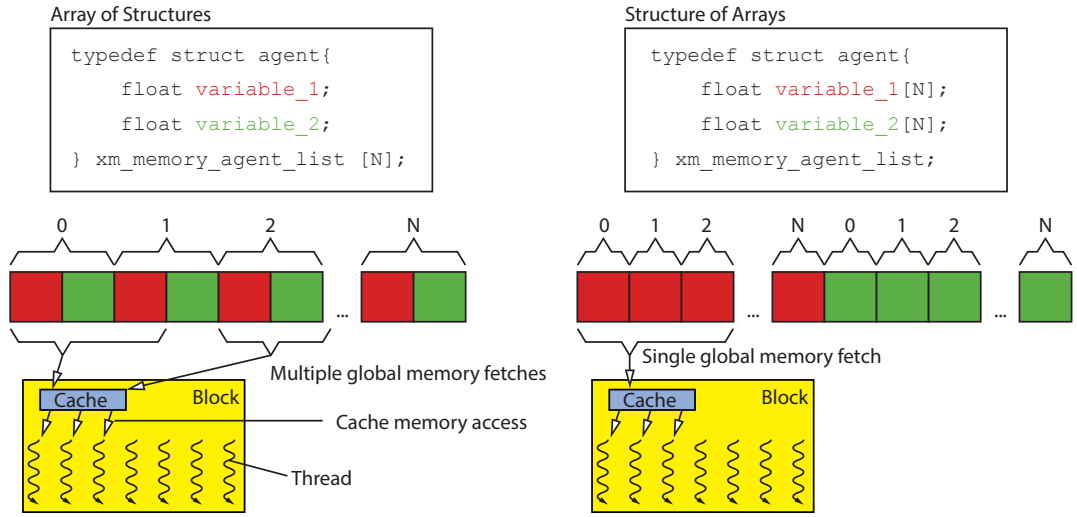

Figure 2.2 The diagram shows the difference in GPU memory access patterns for internal AoS and SoA. SoA leads to fewer memory fetches overall as one request obtains a chunk of memory that can be used by multiple threads.

trians [46, 68, 70, 229–234]. By using GPU to accelerate the processing of simulation in parallel, a gain in an order of magnitude is possible [218, 224].

An attempt has been made previously to create an ABM framework for the GPU using the fixed rendering pipeline [235] but to date the only modern general ABM framework that uses the GPU's general computing capabilities by taking advantage of CUDA is FLAME GPU [236, 237], which is further explored in Section 2.6.3.

2.5.2 HPC Architecture for ABM

Some computational tasks are infeasible to solve within acceptable time on a single processor or are simply too large to fit on system memory for efficient processing. HPC is a concept of splitting up the large task into smaller subtasks and distributing them across multiple processors to be run concurrently. Processors can be connected up through shared memory, i.e. by sharing the same physical RAM and bus. The shared memory paradigm can in fact be seen in modern computers as they often have more than one processor core. The advantage of shared memory is the fact that it is not necessary to synchronise between different memory spaces and therefore there is no communication cost involved with the synchronisation. As the number of processors increases, however, the design of high speed RAM and a bus with large enough bandwidth becomes an issue.

The other approach is the use of distributed memory where each processor has its own physical RAM and communicates through the network connection. The

distributed memory approach is harder to program for as the inter-processor communication needs to be taken into account. The approach is, however, more modular in the fact that each processor can be hosted within a separate machine, commonly referred to as a node, and nodes can be added or removed from the network more easily.

In modern HPC it is common to find a mixture of this where there are many nodes in the network with their memory distributed. Inside each node can be a number of processor cores with shared memory that can perform further parallel processing at the node level. Good examples of modern HPC with a mixed memory architecture are China's Sunway TaihuLight [238], that has 40,960 nodes with SW26010 RISC processors containing 256 computing cores, and IBM's Sequoia [239], which has 98,304 nodes equipped with 16-core PowerPC AC2 processors. Many researchers have used HPC systems for simulating models with very large numbers of pedestrian agents [240–245].

It is important to decide how the work is divided up across the nodes. While the ideal is to have each node doing a similar amount of work, in reality it may not be simple to do this division of labour. Large-scale pedestrian simulation, e.g. city scale with millions of pedestrians, presents a good example where certain parts of the environment may be busy, whereas other parts are less dense and hence require a different amount of computation. If the domain, in this case the physical environment, is decomposed into an even grid of subdomains and each subdomain distributed to a node it can be easily seen that some nodes where there's pedestrian congestion will have more processing work than others. Performing domain partitioning and decomposition in a way that takes into account the processing (e.g. split the congested regions into smaller domains than non-congested regions) that will be performed in the domain is necessary for running an efficient simulation on the HPC [41, 246, 247]. Another factor can include memory available in each node in that the size of any subdomain should not have memory requirements larger than a single node in the cluster.

Tasks distributed between nodes are often not independent from each other. With distributed memory, the synchronisation of data between nodes becomes an issue. In pedestrian simulation for example, a pedestrian agent requires local knowledge of their pedestrian neighbours. If the pedestrian should happen to be on the edge of a subdomain, its perception range can cross over to another subdomain. This means the node must obtain information from other nodes that are responsible for the neighbouring domains. More data sent across the network means extra cycles are needed for the serialisation, deserialisation, sending and receiving of the data. It is then necessary to ensure that only the necessary data is sent across. This can, for example,

be that only the pedestrian information at the edge of a domain is sent. In fact, it is this saturation in network bandwidth that causes bottlenecks in large simulations [39, 248].

HPCs are not limited to nodes with CPUs. Clusters such as Oak Ridge's Titan [208] or Swiss National Computing Centre's Piz Daint [209] are equipped with GPUs to take advantage of both architectures. They are being used in many fields such as fluid dynamics for solving incompressible flow [249], seismic modelling [250], a billion agent boids model [248] and pedestrian simulation [234, 251]. With GPU clusters, the problem of network bandwidth becomes a much bigger issue as the processing throughput of the GPUs means quicker simulation iterations and so requires much more communication between nodes in the same period of time. In order to remedy this, latency hiding techniques can be used such as overlapping data movement with compute, or fast interconnects such as Infiniband of Intel Omnipath must be used.

It is important to note that multi-simulation, as used in this thesis, refers to the running of small simulations that does not require the entire capacity of the GPU. Although a lot of pedestrians may be simulated at the same time with multi-simulation, unlike large-scale simulations [234, 251] where the pedestrians exist in the same simulation domain, these smaller simulations run independently and hence does not require communication between GPUs or different nodes.

2.5.3 A Summary of GPU and HPC Architectures

The GPU and HPC are prominent platforms for parallelising and accelerating scientific computation. It has been shown in many cases that ABM models can be parallelised for simulation on both GPU [46, 68, 70, 218, 224–234] and HPC [39, 234, 240–245, 248, 251] platforms. Each platform, however, provides different advantages for the running of ABM simulations.

For models that can run within GPU memory, GPU simulation can outperform serial simulation on the CPU by an order of magnitude. As memory transfer to and from the CPU is slow compared to the simulation process, it is recommended for programs to limit communication between CPU and GPU. The GPU allows for efficient 3D visualisation as agent memory is located on the same device that is used for 3D rendering. The inefficiency in allocating memory dynamically within agent functions can, however, make some models harder to implement.

Alternatively, HPC systems are suitable for extremely large models where the memory requirements exceeds that of a single machine, for models with many parameters and for models that require dynamic memory allocation. Models that perform less inter-agent communication or ones that can apply spatial partitioning to their messages

will perform better across a larger number of nodes due to the low communication overhead.

2.6 The FLAME Framework

The FLAME GPU framework was used extensively for the work in this thesis as it simplifies the development of pedestrian models optimised for Graphics Processing Unit (GPU) hardware. This section details the concept of the FLAME system (Section 2.6.1) and explains the two variants of the FLAME framework for High Performance Computing (HPC) (Section 2.6.2) and GPU (Section 2.6.3) platform and provides benchmarking using the Circles Model (Section 2.6.4).

The Flexible Large-Scale Agent Modelling Environment (FLAME) is an Agent-based Modelling (ABM) framework designed for efficient multi agent simulation using parallel computing architectures. It is designed so that architectural details are abstracted away from end users who specify models using a high level description of agents and their behaviours. Models are specified as a combination of an XML model description file and functionality code written in C (Section 2.6.1). The framework is robust and has been used in a wide range of agent modelling areas from biological science [69] and biological systems [252–255] through to modelling of the European economy [73].

Versions of FLAME for HPC and for the GPU utilise the same modelling format but target different architectures by specifying different templates for code generation. While both systems offer parallel computation, each has different advantages and limitations which affect the type of simulations that are optimal to run on each platform (as discussed in Section 2.5).

2.6.1 FLAME and FLAME GPU System Overview

FLAME agents are described as communicating stream X-machines [256, 257]. Similar to finite state machines, an X-machine has the addition of memory which is updated according to some function during state transitions. Communication between agents is performed by sending and receiving messages from each other through the use of message boards. Each message board handles a single type of message and is responsible for the efficient storing and retrieving of messages. Message boards are only allowed to be written to or read from once within a transition function. This is in order to ensure a stream like paradigm which prevents race conditions due to global synchronisation being ensured after each transition function. The use of message boards as an intermediary allows agents to be executed concurrently as they are not

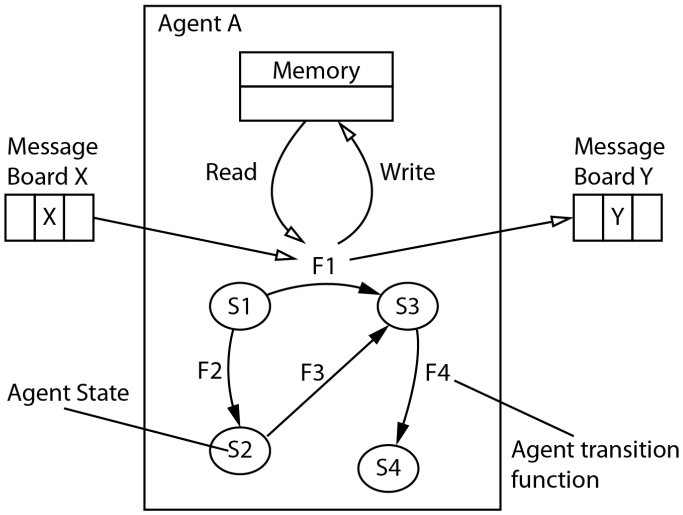

Figure 2.3 Process of transition between agent states and its use of the message board.

dependent on each other's memory. Communication overhead for the simulation is reduced for distributive systems as it is not necessary to communicate entire agents across a network, instead only message boards require distribution. In cases where agent interactions are spatially local, messages can be spatially partitioned to further decrease communication of messages between computational nodes. The processes of agent transitions between states and message board usage is shown in Fig. 2.3. An example of a complete model with two agents is shown in Fig. 2.4. This illustrates how all communication between agents is handled by the message boards and hence shows that only message boards require synchronisation after transition functions that write to them.

A FLAME model is specified using XML, governed by a schema in much the same way as other common XML specification techniques such as SBML [258] or CellML [259]. Model specification includes the definition of items such as global constants, agents and memory, agent state transition functions and messages. In order to generate simulation code, FLAME uses a parser and set of architecture specific template files to generate compilable C code (Fig. 2.5). Agent transition functions defined in the model are implemented as C scripts which link with simulation code at compile time. Running a simulation requires the additional specification of an XML seed file containing initial variables and agent instances. An XML model file is comprised of the following information:

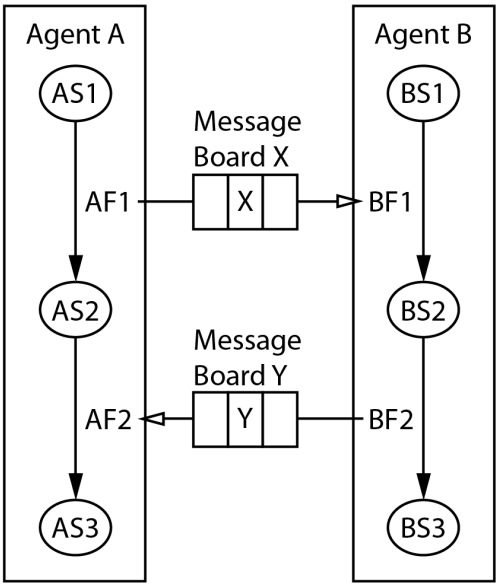

Figure 2.4 Example agent model definition with two agents. Each agent has 3 states (AS# and BS#) and 2 transition functions (AF# and BF#) that either send or receive a message.

1. A set of environment, or constant variables, which may be set between simulation steps. These are useful in controlling aspects of the model within a real time simulation.

2. A set of X-Machine agents. As with the formal definition of an X-Machine [260], each agent definition consists of a description of the agent's internal memory (the agent variables), a set of internal states (which may add a degree of diversity to an agent population) and a set of function definitions (which define the transition between any two states including any agent or message input or output). Each agent may be either discrete, in which case it is non mobile and part of a cellular automaton, or continuous, in which case it may be spatially distributed or represent a more abstract non spatial entity.

3. A set of messages. Each message definition has a set of communicated information (or variables) which may be input or output from within the agent function scripts allowing indirect communication between agents. A message definition also defines any restriction on message range or the type of agent which may use it (discrete or continuous). This ensures the back end simulation uses the most efficient communication algorithm for message iteration within the agent functions.

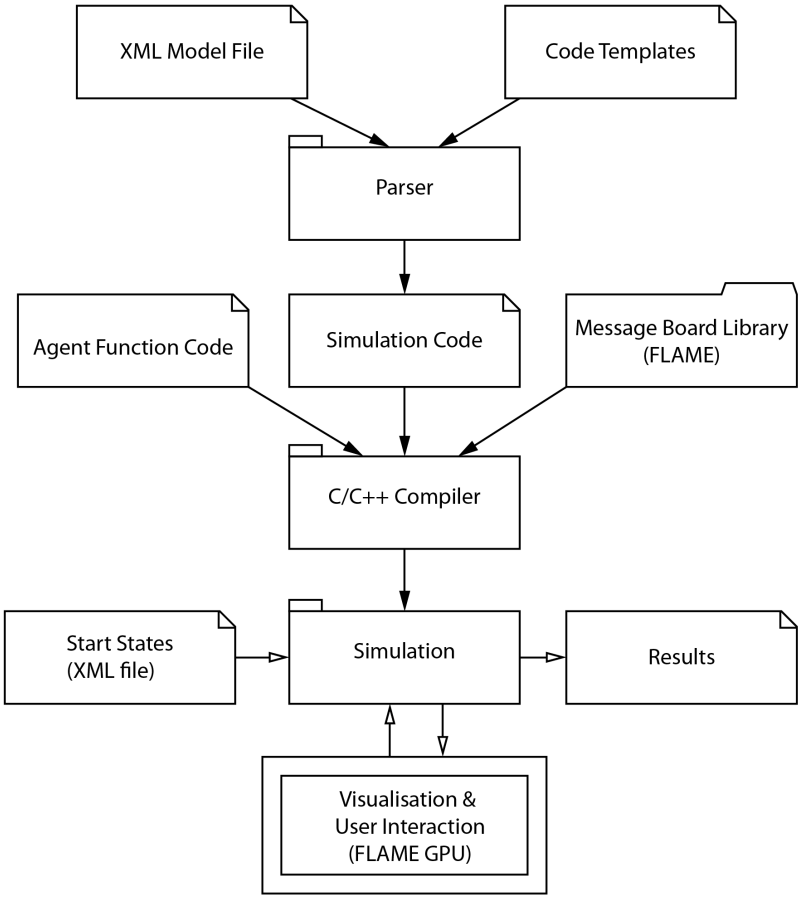

Figure 2.5 Architecture of the FLAME and FLAME GPU frameworks.

4. A set of function layers. This describes the sequential stepwise order of execution of the agent functions within a single simulation step. A single layer may execute more than one agent function in parallel if they share no dependencies, i.e. do not share any common message input or output. While automatically defined in the CPU version using dependency checking, the layers must be defined manually for the GPU version.

2.6.2 FLAME for the HPC Architecture

Although the FLAME framework [261] is capable of running on single CPU systems, it is optimised for running on HPC architectures. The HPC implementation of FLAME is able to automatically distribute agent data across multiple nodes in order to run extremely large models restricted only by the size of the memory of the nodes used and type of partitioning employed. The message board approach is combined with par-

titioning and filtering strategies in order to reduce the cost of network communication. This avoids a bottleneck due to heavy inter-agent communication.

Internally within HPC FLAME, agent memory is implemented as a list of structures which is made fully accessible to the current transition function. The message board is implemented as a library for handling all inter-agent communication between different network nodes. Simulation space can be internally partitioned for agents operating in continuous (Cartesian) space. This partitioning is performed according to the communication radius defined by the agent. Agent distribution across nodes complies with this partitioning regime. As each node knows what space partition others are responsible for, it is possible to filter messages by location and send them to the correct node. Partitioning becomes non-trivial in the case where agents are location agnostic, i.e. when the agent does not have spatial coordinates to limit potential interaction range, the filtering of messages has to depend on a variety of factors. In this case, for each message board, agent filter data is created for each node, i.e. the list of agents in this particular node and the type of messages it receives. This data is amalgamated and distributed to all the nodes. This filter data is then used within each node to filter the number of outgoing messages across the network.

The execution order of agent transition functions and message synchronisation are static and based on an automatically generated function dependency graph. The function dependency graph, in the form of a directed acyclic graph, for all agents in the model is generated by analysing the messages sent and received between agents. As an example, consider the model in Fig. 2.4. A function dependency graph for this is shown in Fig. 2.6. It can be seen that the function BF2 depends on the Y variable output from the AF2 function and that the function AF1 depends on the X variable output from the BF1 function.

Work is currently underway to create an updated version of FLAME which expands upon the issue of data granularity in order to improve parallisation [39]. By extending the dependency graph to account for agent memory variables, granularity can be reduced from agent entity level to variable level. This means FLAME can partition subsets of agent memory for each agent function and it is the execution of functions that can be partitioned instead of whole agents. The execution of these functions can then be best fit to any underlying architecture involving multi-core, multi-node CPUs or multiple GPUs. The ability to perform dependency analysis at the variable level will also help to create a broad dependency graph where functions which are not dependent on the same variables can execute in parallel (Fig. 2.7) as opposed to the current tall and narrow graph (Fig. 2.6) which leads to sequential execution. The variable-level dependency graph will make it possible to implement a dynamic task scheduler which can adapt to the different run-time conditions and communication loads.

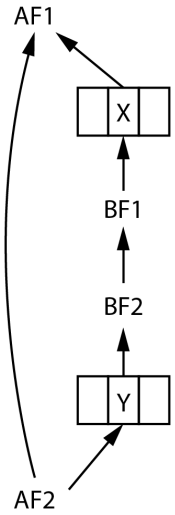

Figure 2.6 Dependency graph generated for the example model in Fig. 2.4.

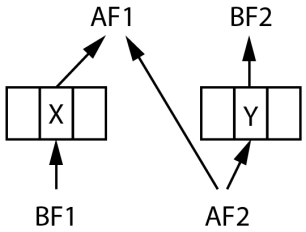

Figure 2.7 An updated dependency graph with data granularity defined. In this example, the transition functions AF1 and BF2 are independent and hence can be executed concurrently.

2.6.3 FLAME for the GPU Architecture

FLAME for the GPU (or the FLAME GPU framework) [40, 261] is an implementation of FLAME which utilises GPU hardware. Rather than use the traditional FLAME parser, FLAME XML models are translated to NVIDIA CUDA C code using an XSLT parser and a set of XSLT templates. Agent functionalities for FLAME GPU are written as CUDA C scripts. Dependency graphs are not generated to automatically schedule agents. It is instead up to the modeller to specify the sequential order in which transition functions are executed. This execution sequence is specified within the XML model files and each entry in the sequence is referred to as a layer.

Internal agent lists and message boards are stored as Structure of Arrays (SoA) (as opposed to Array of Structures (AoS)) to ensure memory access is coalesced and reduce the number of required memory fetch operations (Fig. 2.2). Message boards are bound to texture memory to provide locality based caching. User-defined agent transition functions are called from a wrapper kernel which hides this memory management process. Adding and iterating through messages is performed through a set of device function calls.

There are two different types of agents in FLAME GPU. The first is discrete agents, which are best used to represent a cellular automata. They are represented in memory as a static list and their position in space corresponds to their position in the list. The second is continuous spaced agents, used to represent agents in Cartesian space. Their locations within the agent list do not correspond with their locations in space. New agents can be created and old ones killed during simulation. However, pre-allocated maximum number of agents must be specified. Agents are stored and executed according to their current state. This ensures little divergence between code executed on the GPU, a performance strategy which is particularly beneficial. The agents can be created from within a transition function by calling the framework's agent creation device function specific to the agent name. These calls result in an entry written to a sparse agent birth list which is then compacted and added to the actual agent list. An agent can be killed from within its own transitional function by returning a kill flag. This array of kill flags is used to filter and compact the current list of agents which are still alive.

In order to optimise performance for common communication strategies a distinction is made between three common types of message communication: discrete, brute-force and spatial partitioning. All types of messages are stored as a 1-dimensional list in memory. Discrete messages are only available for 2D space and are stored column-wise in the 1-dimensional message list. The descrete message's location in the list corresponds to the agent's location in the 2D discrete grid. Due to this link

between the agent and message's location, only discrete agents can send discrete messages. Brute-force messages are simply un-partitioned messages and the agent that receives such a message must iterate through the entire message board (i.e. $O(n^2)$ communication). Spatial partitioning applies 3D grid partitioning to messages based on their x, y and z coordinates. After the transition function that outputs this message type, the messages are sorted and hashed to a particular cell in the grid for efficient access.

The simulation can be efficiently visualised with the use of compatible 3D graphics libraries such as OpenGL or Direct3D. GPU data used by CUDA can be bound to texture, vertex or other buffer memory to be used directly by these 3D libraries. OpenGL is used as a standard in FLAME GPU and basic 3D visualisation is provided by default. After every iteration, the x, y and z coordinates of all agents is copied to a texture buffer object (TBO) where they are stored within the red, green and blue colour components, respectively. Agents are rendered as spheres using a GLSL vertex shader which uses an instancing technique to displace agents according to the positions specified within the TBO. Global simulation variables can be defined within the model file and can be manipulated during the running of the simulation. This provides a way to interactively change parameters of the running model (e.g. changing a pedestrian's average speed distribution) or changing the state of the simulation (e.g. specifying that a fire has occurred) or related to rendering (e.g. changing the Level of Detail (LOD) distance).

2.6.4 FLAME HPC and GPU Performance Benchmarking

Two key measures of the performance of any computational simulation are time taken in the modelling process and the quantitative measure of simulation performance (e.g. simulation speed). A force-based model, the Circles model, has been used as a basis for benchmarking performance across the different platforms. A single Circle agent type has a position in a two-dimensional space with a radius of influence. Each agent reacts to their neighbours by exerting a repulsive force. Given sufficient simulation time, the agents will distribute themselves evenly within the environment. Each agent has coordinates x, y and velocity (fx and fy) in its memory. Three state transition functions are defined, of which the first outputs location information, the second cycles the location information of other agents to influence its own velocity and the third performs a movement operation by translation of its own location coordinates. The agents use only location messages which contain x and y coordinates. Both FLAME HPC and GPU utilise their respective message spatial partitioning techniques. The reduction in time required for the modelling process can be difficult to quantify but

the modelling code required to create a complete simulation can provide an indication. The Circles model can be described within 90 XML elements and fewer than 62 lines of agent function script.

Coakley et al. [39] demonstrated the model on a 432 core Fujitsu PRIMEGERY blade server with 36 dual-Intel Xeon X5670 nodes. The simulation uses from 10,000 to 500,000 agents and experiments were performed utilising up to 432 processors. The results indicated that parallel efficiency increases following the increase of agent population. Iteration time does, however, flatten out at around 120 cores whatever the population size. This is the bottleneck that is incurred by the synchronisation of the message boards.

Richmond et al. [262] tested the FLAME GPU version of the model (without visualisation) on a single GPU core of the 9800GX2 and GTX480 graphics cards. Single precision floating point was used for agent memory variables and arithmetic operations throughout the 9800GX2 while double precision was used for the GTX 480. Additionally, further benchmarking has been performed using the GTX 590 with both single and double precision on a single GPU core. The result of the benchmarking is shown in Fig. 2.8. One thing to note is that the simulation times for single and double precision showed no remarkable difference as the model is relatively communication heavy. For 1,048,576 agents a single iteration can be simulated (double precision) in only 27ms on the GTX590.

When comparing the GPUs to the HPC architecture, the HAPU with 100 cores is able to simulate 1 million agents at around 6.641 seconds per iteration and the GTX590 uses 0.027 seconds giving a speed up of around 245 times. It shows the graphics hardware can compete directly with HPC architectures when simulating large number of agents due to the fact that it is a high-bandwidth architecture and as memory is not distributed across multiple devices there are no issues with network latency.

2.7 Summary

This chapter has outlined the components of pedestrian modelling and the importance of local motion and navigation for creating a complex simulation that behaves closely to what is observed in reality. While computability presents a challenge to pedestrian ABM, parallel hardware architectures such as the GPU or HPC can be used to speed up simulation times or simulate larger models. Pedestrian simulation has been applied to help solve many real-world design, crowd flow shaping and evacuation problems and many tools and frameworks exist to help support this workflow. The implementation of pedestrian simulation into a fully integrated Decision Support System (DSS) still

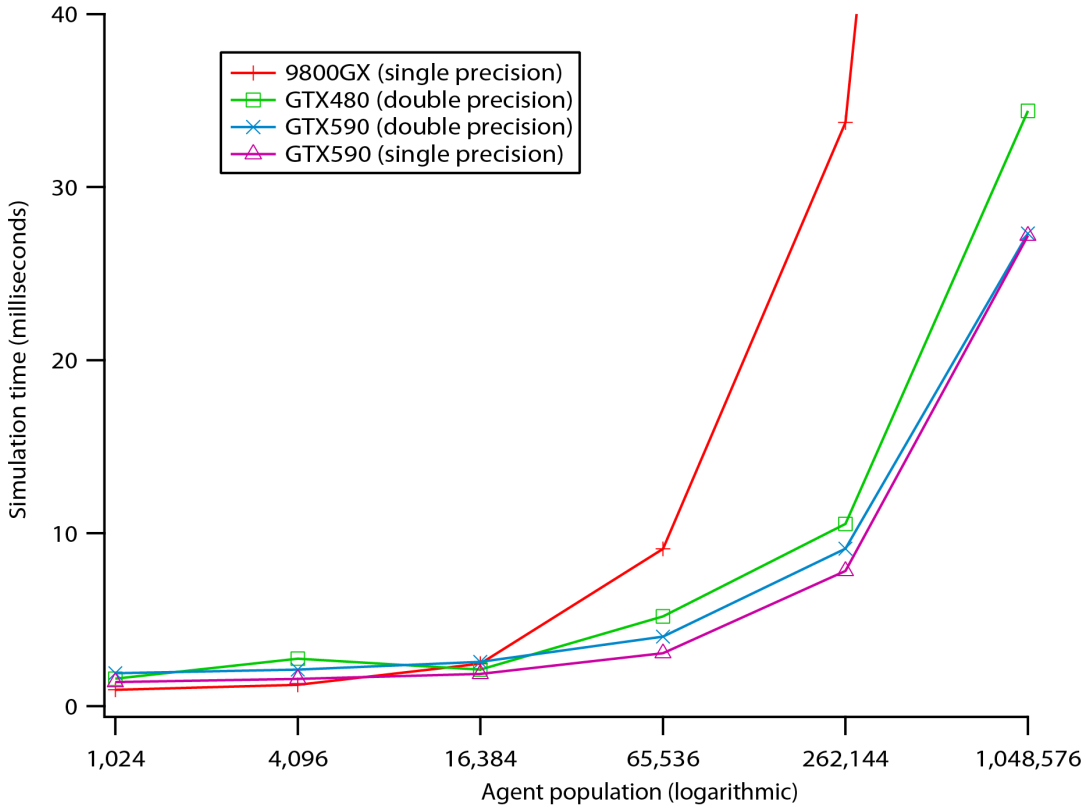

Figure 2.8 Simulation times for various agent populations on different GPU hardware.

faces many challenges. There is the need for optimisation algorithms for ranking and suggesting an optimal scenario, computability needed to explore the solution space proposed by the optimisation algorithm and tracking systems needed for obtaining the state of the current environment which can also be used to provide historical data.

In this thesis, the computability issue is addressed in Chapter 3 with the use of a fast and scalable pedestrian navigation system in combination with models optimised for the GPU. In Chapter 4, the results of the searchable navigation graph system Section 3.2 are demonstrated using a working prototype pedestrian simulation tool named Concoursia. Finally, Chapter 5 details a prototype multi-simulation tool created for running multiple simulation scenarios concurrently across multiple GPUs and computer nodes.

Chapter 3

Navigation for Pedestrian ABM

Whilst the Agent-based Modelling (ABM) approach to pedestrian simulation using fine or continuous networks allows for a more detailed and heterogeneous representation of pedestrian behaviour, it comes at a much higher computational cost when compared to pure macroscopic (coarse network) models. High Performance Computing (HPC) has been proven to be a viable platform for running large pedestrian ABM. By splitting up the simulation domain and distributing the process across multiple HPC nodes, Grandison et al. [41] was able to achieve up to 13.9 times speed up compared to a single CPU. However, as discussed in Sections 2.5 and 2.6.4, the use of a Graphics Processing Unit (GPU) can provide a more cost-efficient way to accelerate the simulation of ABM, as there is a large investment difference in a single GPU system and a HPC cluster. The caveat is that the benefit can only be obtained for models where both the representation of the agents and the environment are small enough to fit within the memory of a single GPU.

By keeping the model small enough to be simulated on a single GPU device, the extra complexity and processing required to synchronise memory between multiple nodes can be avoided. In addition, there is no need to manage the latency that results from the difference between the higher bandwidth internal memory and the lower bandwidth cross-node communication. The bandwidth difference is especially pronounced when each node in the HPC is equipped with GPUs since they are high throughput devices. Keeping the model small enough means keeping in mind the representation of pedestrian agents and the environment. A good example of this can be seen when comparing the representation of the environment as a uniform grid (Section 3.1) and as a graph (Section 3.2) where the latter's memory requirement does not increase exponentially with the environment size.

The use of local perception alone for navigation is insufficient in all but the simplest of environments. In many cases, a pedestrian's destination extends much further than

their local perception range with obstacles such as corners, walls and stairs in between. When many alternative routes to a destination exist, many heuristics can apply to the choice [100, 160–162], such as a pedestrian's familiarity with the environment, route chosen by other pedestrians, signage or congestion. It is possible that pedestrians need to navigate to multiple different locations to achieve their overall goal, for example catching a specific train would require one to buy a ticket and navigate to the correct platform. In terms of the implementation of a navigation system, it must be optimised to the GPU platform and be scalable in terms of the environment size and has high performance even when supporting large number of pedestrian agents.

This chapter details the research into two types of navigation system suitable for use with pedestrian ABM systems, agent-based navigation grid (Section 3.1) and searchable navigation graph (Section 3.2). A comparison of the two navigation systems is provided in Section 3.3. This includes a comparison of their memory use and a discussion on when best to utilise each system.

All of the work in this chapter uses the FLAME GPU framework (Section 2.6.3) to implement the pedestrian agent model on the GPU in order to gain the processing advantages provided by the parallel architecture.

3.1 Agent-based Navigation Grid

There are many global navigation techniques that are suitable for use with pedestrian ABM, as discussed in Section 2.2.2. These techniques however, are implemented separate to the pedestrian model.

This section proposes a novel navigation system that fully integrates with the ABM approach. It is a system where both pedestrians and the environment are modelled as agents. It uses a grid of discrete navigation agents to provide grid-based vector field navigation for pedestrian simulation. This means the system can be implemented in any generic ABM framework, allowing it to take advantage of any support for parallel architecture. In addition, a novel context-aware smoothing technique termed "backflow smoothing" is introduced in Section 3.1.2 to deal with the problem of diagonal convergence that can appear when generating vector fields. This work has been published at TPCG 2010 [70].

The FLAME GPU framework (Section 2.6) is used as a base for implementing the grid-based navigation system and pedestrian model. In accordance with the FLAME GPU model specification, two distinct types of agents are used: the pedestrian agents (Section 3.1.1) and the navigation agents (Section 3.1.2). Fig. 3.2 illustrates, in chronological order, the sequence of functions that each agent performs and the messages

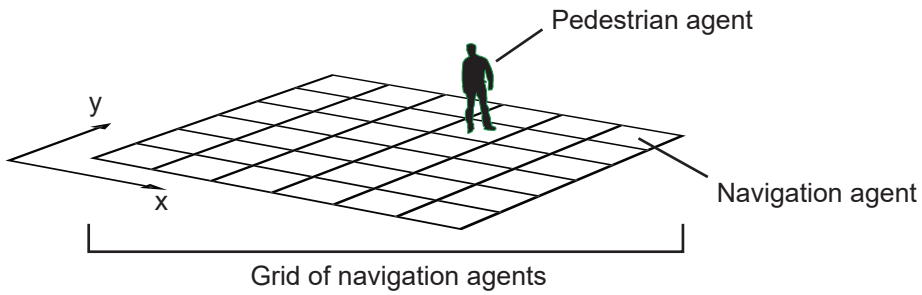

Figure 3.1 Pedestrian agents exist on top of a grid of discrete space navigation agents and their position can be transformed to a relevant navigation agent cell.

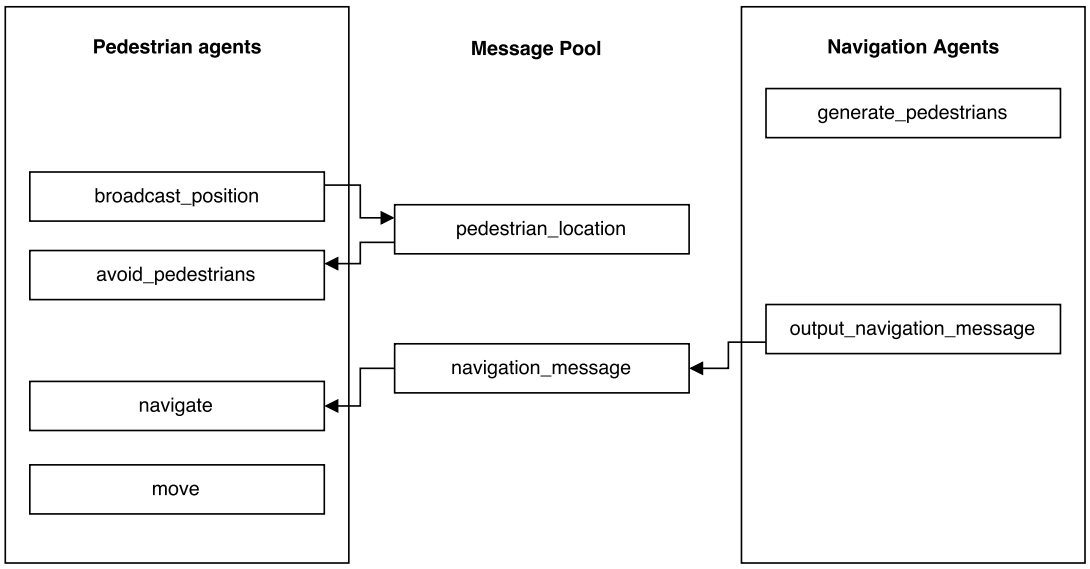

Figure 3.2 Sequence diagram for function call of both agents and their utilization of the message pool.

used in their communication. In the model, the environment that the pedestrian agents interact with consists of navigation agents arranged in a discrete grid (Fig. 3.1). Each navigation agent contains a set of vectors specific to their location that are used to guide the pedestrian agents to their destination and for avoiding static obstacles such as walls. They also act as entrances and exits to the simulation.

In order to help with the authoring of the environment, a GUI tool (Section 3.1.3) was created that allow users to create and edit the physical environment. The tool is then able to create the required vector fields and export the data to be used as the initial states of the navigation agents for use in the simulation.

An evaluation using a simulated scenario based on a real world environment to test the performance and behaviour of the system is presented in Section 3.1.4.

3.1.1 Pedestrian Agents

Pedestrian agents are continuous space mobile agents, each one representing a single embodied pedestrian entity within the simulation. The functions of the pedestrians and how they interact with navigation agents are outlined chronologically from top to bottom in Fig. 3.2 and the parameters of each agent are shown in Listing 3.1. For this particular case, the standard social force model with contact force [22] was chosen as it is a popular model used within research. However, other variations of continuous spaced pedestrian models, as mentioned in Section 2.2, can be used interchangeably.

The motion of pedestrian agent is calculated using a social and contact force model by Helbing et al. [22], where large contact forces are applied between pedestrians when they are touching each other, which helps to more accurately simulate areas of higher congestion. For N number of pedestrians, each pedestrian i with mass m_i and desired speed v_i^0 and their neighbouring pedestrian j has a force equation (3.1) where all forces exerted on and by the pedestrian $\vec{f}_{All}$ is composed of the navigation force $\vec{f}_{Nav}$, a sum of inter-pedestrian interaction forces $\vec{f}_{ij}$ and a sum of static obstacle interaction forces $\vec{f}_{iw}$.

$$\vec{f}_{All} = \vec{f}_{Nav} + \sum_{j(\neq i)} \vec{f}_{ij} + \sum_{W} \vec{f}_{iW} \tag{3.1}$$

The navigation force $\vec{f}_{Nav}$, which can be found using Equation (3.2), is the force exerted to reach a desired speed v_i^0 in the desired direction $\vec{e}_i^0$ where v_i is the current speed and τ_i time characteristic.

$$\vec{f}_{Nav} = m_i \frac{v_i^0(t)\vec{e}_i^0(t) - v_i(t)}{\tau_i} \tag{3.2}$$

In the inter-pedestrian interaction equation (3.3) the first term $A_i e^{(r_{ij}-d_{ij})/B_i}$ is composed of repulsive force from other pedestrians that exponentially increases the nearer the distance between pedestrians. A_i and B_i are weighting constants, $r_i j$ is the sum of the two pedestrian's radii $r_i + r_j$, and d_{ij} is the distance between the centre of mass of two pedestrians. The second term $kg(r_{ij} - d_{ij})\}\vec{n}_{ij}$ is the 'body force' counteracting body compression and the third term $\kappa g(r_{ij} - d_{ij})\Delta v^t_{ji}\vec{t}_{ij}$ is the 'sliding friction force' which impedes tangential motion, both only apply when pedestrians are touching each other and $d_i j$ is less than $r_i + r_j$. The parameter $n_{ij} = (n^1_{ij}, n^2_{ij}) = (r_i - r_j)/d_{ij}$ is a normalised vector pointing from pedestrian j to i. Parameters $k = 1.2 \times 10^5 kgs^{-2}$ and $\kappa = 2.4 \times 10^5 kgm^{-1}s^{-1}$ are obstruction effects. Parameter $t_{ij} = (-n^2_{ij}, n^1_{ij})$ is the tangential direction and Δv^t_{ji} is the tangential velocity difference. $g(x)$ is a function that outputs zero when pedestrians are not touching.

$$\vec{f}_{ij} = \{A_i e^{(r_{ij}-d_{ij})/B_i} + kg(r_{ij}-d_{ij})\}\vec{n}_{ij} + \kappa g(r_{ij}-d_{ij})\Delta v^t_{ji}\vec{t}_{ij} \tag{3.3}$$

The static obstacle interaction equation (3.4) is composed of the wall avoidance term $A_i e^{(r_i-d_{iW})/B_i}$, compression term $kg(r_i - d_{iw})$ and the friction force term $\kappa g(r_i - d_{iW})(\vec{v}_i \cdot \vec{t}_{iW})\vec{t}_{iW}$. The parameter d_{iW} is the distance to the obstacle W, n_{iW} is the direction perpendicular to the obstacle and t_{iW} is the tangential direction.

$$\vec{f}_{iw} = \{A_i e^{(r_i-d_{iW})/B_i} + kg(r_i - d_{iw})\}\vec{n}_{iW} - \kappa g(r_i - d_{iW})(\vec{v}_i \cdot \vec{t}_{iW})\vec{t}_{iW} \tag{3.4}$$

For this model while the desired speed is kept constant at standard walking speed of $v_i^0 = 1.4ms^{-1}$ other variables are kept the same as the original paper [22] where the mass is $m = 80kg$, acceleration time is $\tau_i = 0.5s$, constant $A_i = 2 \times 10^3$ and constant $B_i = 0.08m$.

Each pedestrian agent first broadcasts their locality and velocity in the `broadcast_position` function. The broadcast messages `pedestrian_location` (Listing 3.2) are stored in a 3D spatial hash with pedestrian's position as key. In the `avoid_pedestrian` function pedestrians iterate through other pedestrian location messages within a radius of $r_{msg} = 5m$. The repulsive force $\vec{f}_{ij}$ is calculated for each pedestrian in range and stored in the variables `force_x` and `force_y`. The `navigate` function translates the pedestrian agent's current coordinates into a discrete grid coordinate and obtains the `navigation_message` at that location. The pedestrian uses its `exit` variable to determine the correct navigation vector parameter to use as the desired direction $\vec{e}_i^0$ and when the pedestrian's `exit` variable matches the `navigation_message exit` variable the pedestrian knows it has reached the correct exit. The `collision_x` and `collision_y` variables contain the location of the nearest static obstacle and are used to calculate the wall interaction force $\vec{f}_{iw}$ which is used to set the agent's `force_x` and `force_y` variables. Navigation and static obstacle avoidance is further explored in Sections 3.1.2 and 3.1.2. The change in velocity $\Delta\vec{v}_i$ of the pedestrian is calculated using Equation (3.5) where the current time step is normally set at $\Delta t = 0.2s$.

$$\Delta\vec{v}_i = \frac{\vec{f}_{All}\Delta t}{m_i} \tag{3.5}$$

To prevent numerical instability, where $\Delta\vec{v}_i$ is above the speed threshold $v_{\Delta max} = 0.2ms^{-1}$, Δt is scaled according to Equation (3.6). As the time step applies to all pedestrians, the threshold is applied to the one with the greatest $\|\Delta\vec{v}_i\|$ by having all pedestrians store the value in the `deltaSpeed` parameter and the scaling is performed after the `navigate` function has completed.

$$\Delta t_{final} := \frac{v_{\Delta max}}{\|\Delta \vec{v}_i\|} \Delta t \tag{3.6}$$

Finally, with the scaled time step Δt_{final} the `move` function re-calculates the velocity change $\Delta \vec{v}_i$. The change in velocity is then added to the current velocity $\vec{v}_i$ with equation (3.7) and stored in pedestrian variables `vel_x` and `vel_y`.

$$\vec{v}_i := \Delta \vec{v}_i + \vec{v}_i \tag{3.7}$$

The position $\vec{p}_i$ can then be calculated with Equation (3.8) which is used to set the pedestrian's position variables `x` and `y`.

$$\vec{p}_i := \vec{v}_i \Delta t_{final} + \vec{p}_i \tag{3.8}$$

```
pedestrian_agent:agent{
    x:float32
    y:float32
    height:float32
    deltaSpeed:float32
    vel_x:float32
    vel_y:float32
    force_x:float32
    force_y:float32
    exit:int
}
```

Listing 3.1 Pseudocode of a single pedestrian agent's parameters.

```
pedestrian_location:message{
    x:float32
    y:float32
}
```

Listing 3.2 Pseudocode of a single pedestrian_location message's parameters.

3.1.2 Navigation Agents

Navigation agents are discrete space agents arranged on a two dimensional grid. As each agent contains a set of vectors used to assist the pedestrian agents in their global navigation, each agent can be thought of as a cell and a grid of them makes up a set of overlapping vector fields. Fig. 3.3 illustrates an example of a simple environment with

a wall to the north and an obstacle in the middle (black squares). In the example the pedestrian has two destination locations, either navigating to the exit to the west or to the north-east (red squares). Two types of vector fields are used. The vector field used for the avoidance of static obstacles such as walls is called the Collision Vector Field (CVF). The vector field used for guiding agents to their intended goals is referred to as the Navigation Vector Field (NVF). In this example model, all goals are associated with an entrance/exit to the environment and the number of NVFs corresponds to the number of exits. Each cell in the NVF is encoded with a vector that points to the shortest non-obstructed direction to the goal. Each agent can also be marked as an entrance and an exit associated with a single destination. An agent marked as an entrance is able to emit pedestrians. An agent marked as an exit allows pedestrians to exit the environment once they have reached the location of the agent and if they share the same destination id.

The Navigation agents have two functions, `generate_pedestrians` and `output_navigation_message` (Fig. 3.2). When the `generate_pedestrians` function is called, if the navigation agent is deemed to be an entrance it will randomly generate pedestrian agents at that location at a configurable (people per minute) rate. Newly generated pedestrian agents are then assigned a random exit goal according to a probabilistic function. Since all vector fields were generated before the start of the simulation, at run-time navigation-agents need only to broadcast information of their respective cell in the vector field. This is handled by the `output_navigation_message` function.

The structure of a `navigation_message` can be found in Listing 3.3. The `height` parameter is used to encode a discrete height map of the simulated environment and is used within the visualisation to displace pedestrian agents across differing physical levels in the simulated space.

For a CVF cell, the variables `collision_x` and `collision_y` encode a normalised repulsion vector pointing away from the obstacle $\vec{\boldsymbol{n}}_{iW}$. The variable `collision_d` is distance away from the pedestrian d_{iW} that can be used directly in the static obstacle interaction force equation (3.4). Only one CVF map is required as pedestrians still have to avoid the same static obstacles whatever their destination.

The variables `exit#_x` and `exit#_y` describe a normalised navigation vector with the direction leading agents towards the nearest exit cell and are used as the $\vec{\boldsymbol{e}}_i^0$ variable in the agent navigation force equation (3.2). The number in the `exit#_x` or `exit#_y` variables indicates which NVF map it belongs to and therefore the number of NVF maps used is dependent on the number of destinations available within the model. For the example in Fig. 3.3, there are two destinations.

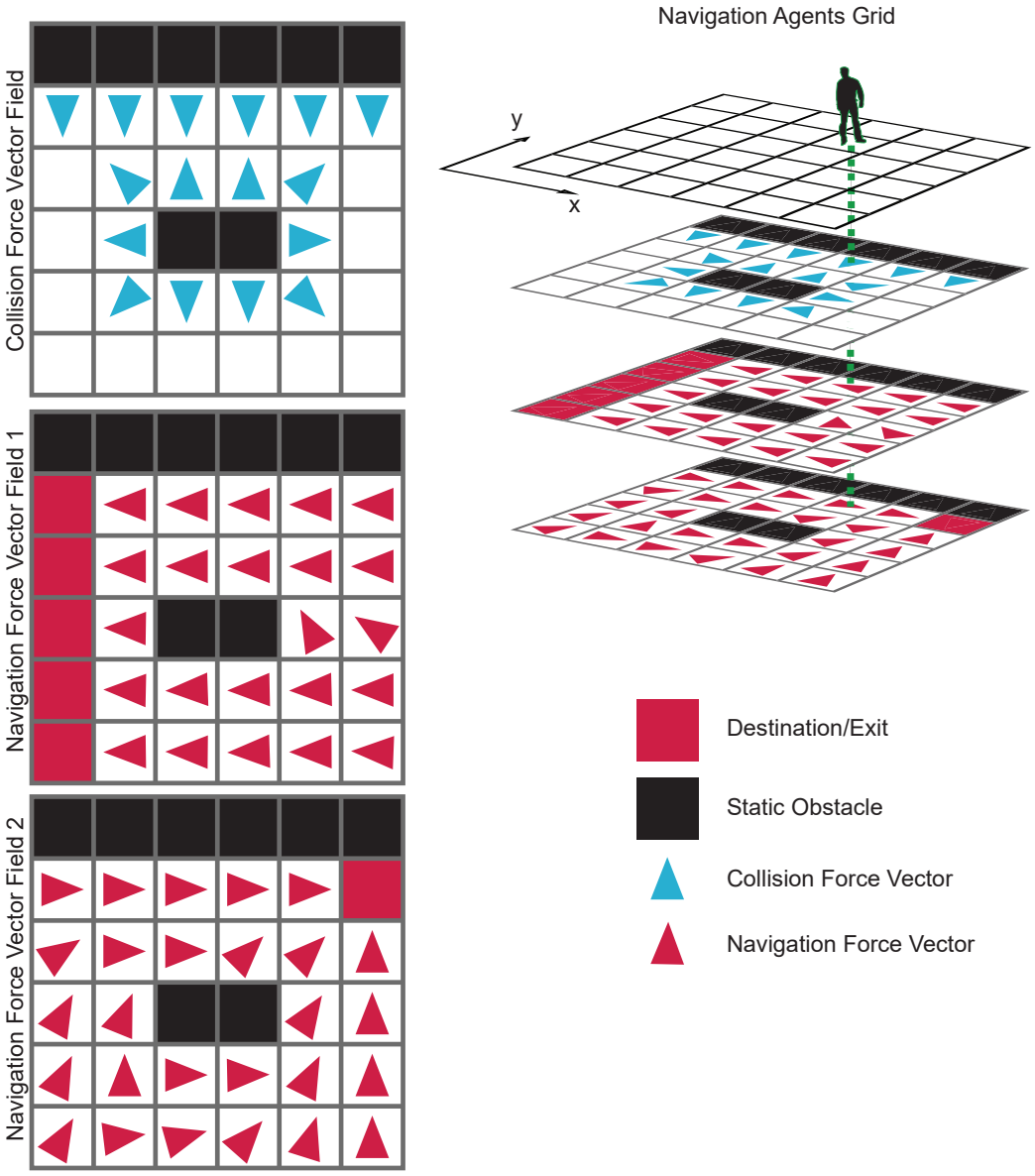

Figure 3.3 An example of a 6x6 environment with 2 exits and obstacles. Navigation agents are arranged in a grid as shown (top right) and the position of the agent relates to a particular cell of the vector fields.

The `entrance` variable has an associated id that can be used to determine it as part of a single entrance point and allows emission rates to be set according to entrance. Finally the variable `exit` is used to determine if the force cell indicates the presence of an actual exit position within the simulation. Exit positions play two important roles within the simulation. As the exits also double as entrances, the exit position acts as an ID allowing the navigation agent to know which entrance it is. This allows the navigation agent to get the emission rate associated to the specific entrance as well as setting the emitted pedestrian's destination to be different to the current exit. In addition, pedestrian agents use the exit position to identify (through message communication of the cell information) that they have successfully reached their destination point.

```
navigation_message:message{
    x:float32
    y:float32
    exit:int
    entrance:int
    height:float32
    collision_x:float32
    collision_y:float32
    collision_d:float32
    exit0_x:float32
    exit0_y:float32
    exit1_x:float32
    exit1_y:float32
    exit2_x:float32
    exit2_y:float32
    ...
    exitN_x:float32
    exitN_y:float32
}
```

Listing 3.3 Pseudocode of a single navigation_message parameters. Each `exit#_x` and `exit#_y` variable belong to a different set of NVF.

Collision Vector Fields

Navigation agents are generated by rasterising a vector-based representation of the environment, as shown in Fig. 3.4, with each cell $0.25m$ in width in accordance with the smallest obstacle size. Each destination, which can consist of multiple exit areas, has the same exit id. Each cell is either determined to be a walkable area or an obstacle.

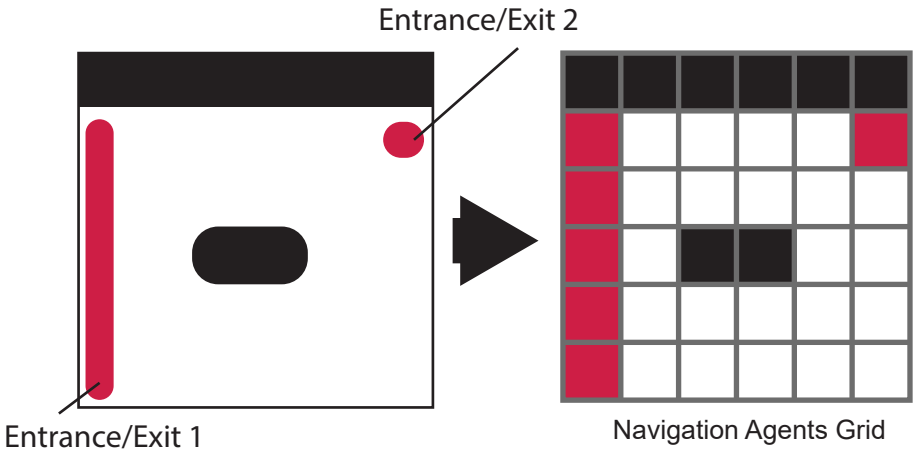

Figure 3.4 A vector representation of the environment is rasterised into a grid of navigation agents. White cells are walkable terrain, black cells are static obstacles and red cells are exits which also double as entrances.

A wavefront propagation algorithm ([147] p.378) is then used with the origin of the propagation being the list of cells that have been marked as an obstacle. The algorithm proceeds to collect all adjacent cells following the visibility rule shown in Fig. 3.6 that are not already in the list (Fig. 3.5). This produces a list of valid and not already visited adjacent cells. For each of these valid cells, the collision vector is the normalized sum of the vectors pointing away from its neighbouring cells that were visited last iteration (so for the first iteration, it would be the vectors pointing away from obstacle cells) and is used to represent the term $\vec{\boldsymbol{n}}_{iW}$ in the Equation (3.4). The Euclidean distance to the nearest obstacle is also stored which represents the term d_{iW} in Equation (3.4). The adjacent cells list is used as the origin for the next iteration and also added to a list of all visited cells. The process repeats until there are no more cells to visit or by reaching a fixed maximum number of iterations (the wall influence distance d_{Wmax}). An example of a calculated CVF map is shown in Fig. 3.3 (top left) where d_{Wmax} is set to be 1.

As interaction force with a static obstacle is sensitive to exponential increase, it is important to obtain an accurate distance from the static obstacle d_{iW} especially when a pedestrian is near to it. To obtain a continuous d_{iW} from discrete cell values, the pedestrian agent's position within the cell is used. First a vector of a pedestrian's position $\vec{\boldsymbol{p}}_i$ relative to the cell C's centre location $\vec{\boldsymbol{p}}_C$ is using Equation (3.9).

$$\vec{\boldsymbol{d}}_{iC} = \vec{\boldsymbol{p}}_i - \vec{\boldsymbol{p}}_C \tag{3.9}$$

The vector $\vec{\boldsymbol{d}}_{iC}$ is then rotated with the function $rot(\theta, \vec{\boldsymbol{x}})$ by $\vec{\boldsymbol{n}}_{iW}$'s angle $\theta_{\vec{\boldsymbol{n}}_{iW}}$ away from the x axis found by using the $atan2(\vec{\boldsymbol{x}})$ function (Equation (3.10)). This rotation

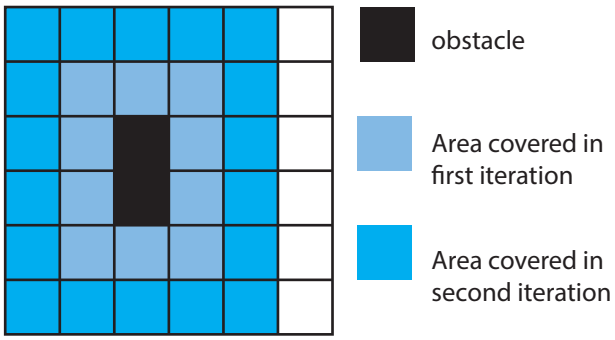

Figure 3.5 Wavefront propagation performed on the obstacle areas. Black areas are obstacles, white areas are walkable, light blue is the area covered in the first iteration and dark blue areas are covered in the second iteration.

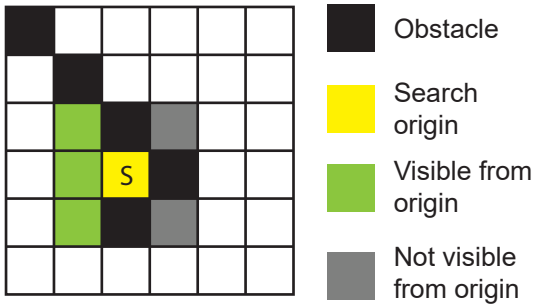

Figure 3.6 Visibility rules for the wave propagation. Starting from origin (yellow), the green areas will be included while the grey areas are not due to it being blocked off by obstacles.

obtains $\vec{d}_{iC\tau}$ which is the vector $\vec{d}_{iC}$ aligned to the frame of reference $\vec{n}_{iW}$. The process is illustrated in Fig. 3.7. The x component of the vector $\vec{d}_{iC\tau}$ can then added to d_{iW} to achieve a continuous distance value (Equation (3.11)).

$$\vec{d}_{iC\tau} = rot(-atan2(\vec{n}_{iW}), \vec{d}_{iC}) \tag{3.10}$$

$$d_{iW} := d_{iW} + \vec{d}_{iC\tau_x} \tag{3.11}$$

Navigation Vector Fields

The NVF map for each destination is generated in a similar way to the CVF with the difference that the origins of propagation are pre-defined exit points rather than obstacles (Fig. 3.8). Also, each vector points towards the cells of previous iterations and after normalisation can be used directly as the parameter $\vec{e}_i^0$ in the navigation force

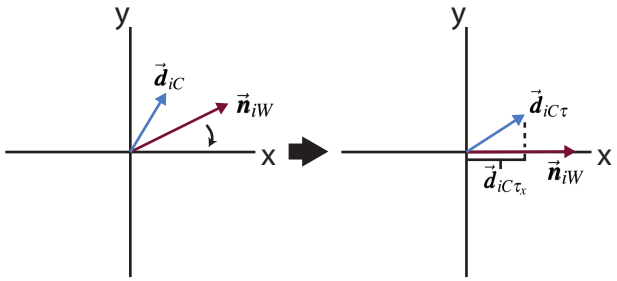

Figure 3.7 Rotation of navigation agent cell to pedestrian distance vector $\vec{d}_{iC}$ to align with the the frame of reference of vector $\vec{n}_{iW}$.

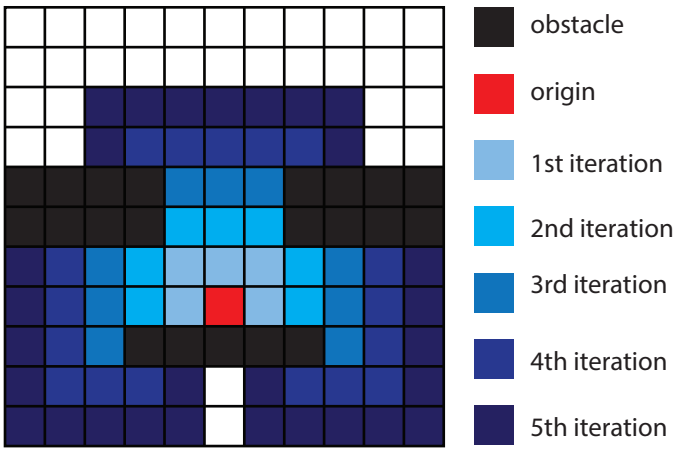

Figure 3.8 Wavefront propgation performed on a destination cell (red) over 5 iterations.

equation (3.2). Propagation also follows a visibility rule (Fig. 3.6) so that it does not go through walls or obstacles. The generated NVF is shown in Fig. 3.9(b).

Smoothing the Navigation Vector Field

Although the generated NVF will guarantee to find the shortest path to the destination from anywhere in the environment, a straightforward implementation of the algorithm results in a common issue which can be described as a "diagonal convergence" of vectors (Fig. 3.9(b)). This is a result of the diagonal lines which propagate from corners representing the shortest path with surrounding flow vectors attempting to converge towards them. This becomes even more apparent as the grid resolution is increased around larger areas of open space and results in squashed and unnatural pedestrian flows (Fig. 3.10(a)). In order to tackle the problem of diagonal convergence, it is necessary to smooth out the vector flows.

A novel process of backflow smoothing can be used to create a natural looking navigation flow for pedestrian agents successfully avoiding convergence issues. Although

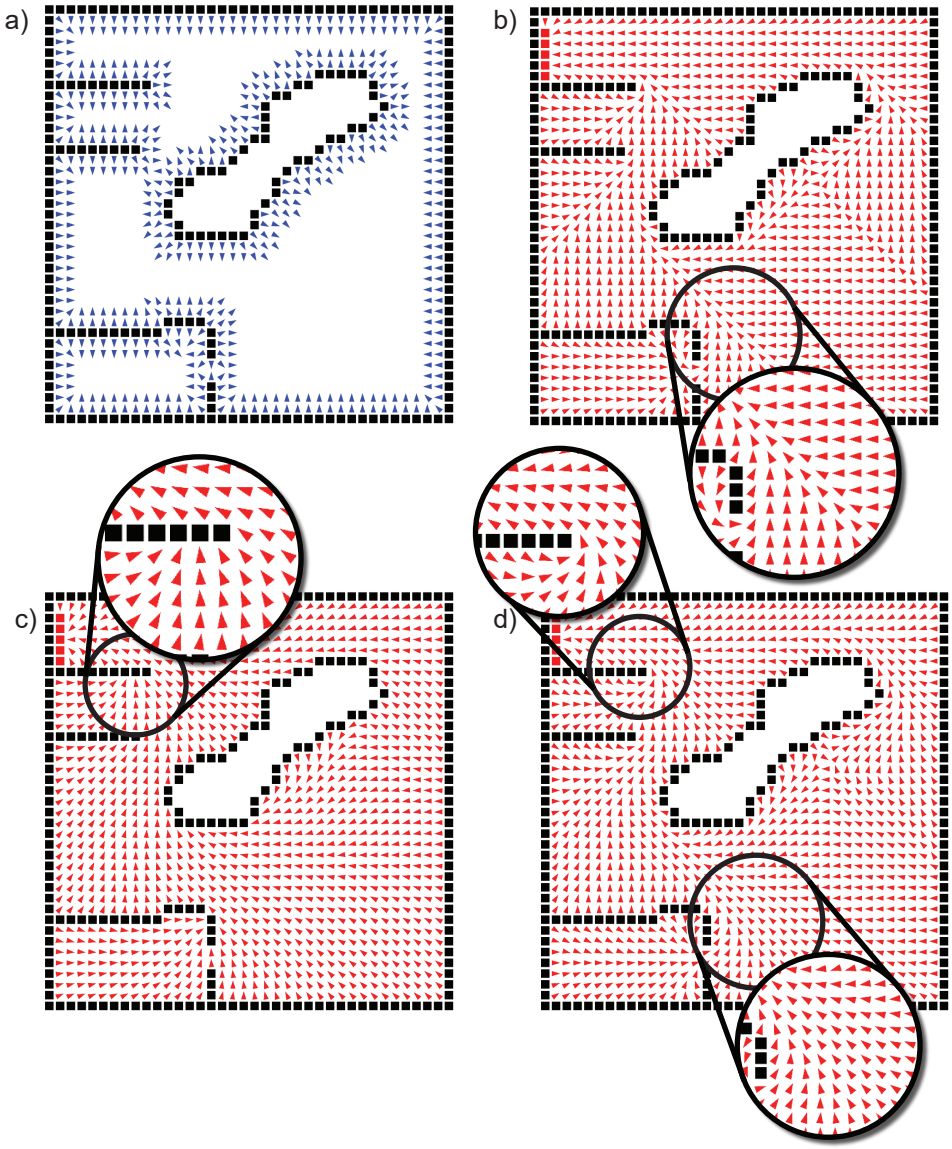

Figure 3.9 Comparison of the generated force vector fields. (a) A CVF with force influence distance of 2. (b) Shows an NVFs with simple propagation , with backflow propagation but no limit (c) and the result of the final algorithm (d).

one potential solution is to use a simple neighbourhood averaging of vectors. The progressive nature of the backflow smoothing however, results in an even smoother field especially in large open areas.

The backflow smoothing approach works by propagating a flow in the reverse direction for each cell of the current iteration and the navigation vector is generated by summing up the vectors pointing from the current cell to the cells in the reverse flow. This reverse propagation also follows the same visibility rules of the main wave propagation (Fig. 3.6) and flows around obstacles. The influence of reverse flow vectors is decreased in proportion to the increase in distance of the backflow. The final navigation vector is then normalised after the backflow reaches its maximum iteration distance.

As the resolution of the grid increases it then becomes necessary to increase the number of iterations to generate smooth flow, especially if there are large open areas in the scene. Using more iterations, however, can cause problems in the rest of the scene especially around tight corners where vectors run into obstacles (Fig. 3.9(c)). This results in pedestrians getting stuck to walls and obstacles then the NVF is used.

The problem is solved by placing a limit on the backflow when an obstacle is encountered. Fig. 3.11 visually illustrates how the final algorithm works. Where the backflow is limited to three iterations, it can be seen that at location $B1$ it is able to apply a backflow propagation for the full three iterations. The backflow only propagates backwards into the previous wavefront iteration. This means for $B1$ that starts at the 5th propagation iteration, the first backflow iteration propagates back to the 4th wavefront propagation iteration (green), the second on the 3rd (light brown) and third on the 2nd (dark brown). $B2$ is stopped at a single iteration as there is an obstacle next to the backflow. The same occurs for $B3$ where no backflow propagation occurs as the cell is already next to an obstacle. The final algorithm generates a vector field that is both smooth in open areas but is also correct around corners and narrow corridors (Fig. 3.9(d)). The effect of this algorithm can be seen in Fig. 3.10(b) where pedestrians (in blue) can spread out to take up more space thus making the simulation look more natural.

3.1.3 Environment Editor

In order to facilitate the rapid generation of the environment, a GUI tool was created using C# and WPF (Fig. 3.12).The tool allows the user to sketch or trace over a plan of an area and mark it as an obstacle, entrance or exit. The program can then generate a CVF and individual NVFs for each destination according to user-specified parameters. The parameters are influence distance for CVF generation and maximum backflow

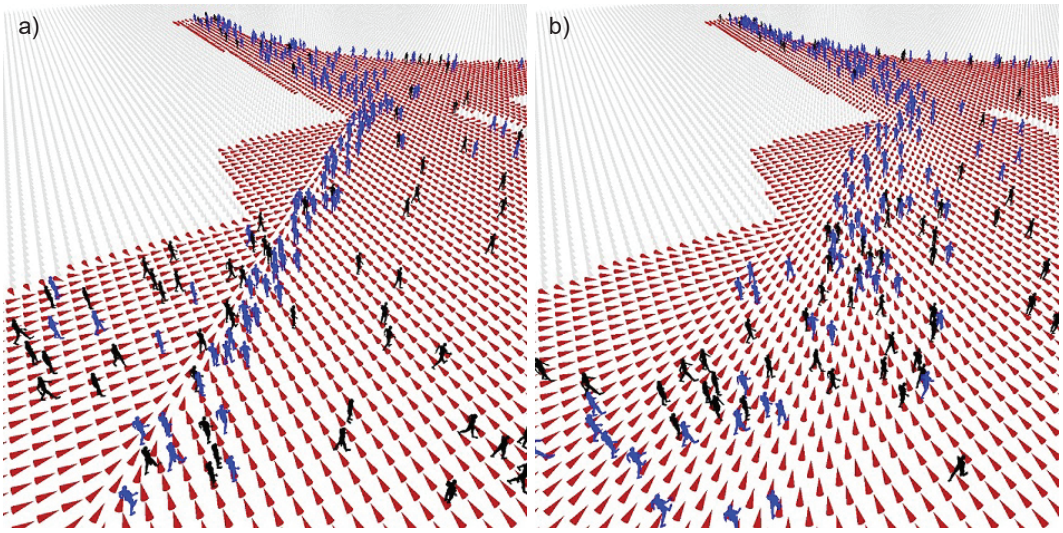

Figure 3.10 Comparison of the simulation using un-optimised (a) and with backflow smoothed (b) navigation vector fields.

iterations for NVF generation (refer to Sections 3.1.2 and 3.1.2). Fig. 3.12 shows an example generated environment with 2 exits. The environment size is set to be 64x64 and maximum backflow propagation iteration is 5. Finally an XML file is generated in the format compatible with the navigation agent definition for the FLAME GPU simulation framework.

McIlveen et al. [263] later extended on the concept with the creation of a Photoshop-like tool for authoring the vector fields. The approach in addition allowed users to annotate areas of attraction, areas of avoidance and areas of interest to provide a multi-objective navigation system.

3.1.4 Simulation & Results

This section presents two tests in order to evaluate the agent-based navigation approach. The first test is a performance benchmarking that varies the navigation agent grid size. Each grid size is tested for iteration time when there are various numbers of pedestrian agents in the environment. The second test uses a real urban environment modelled after a location in central London in order to evaluate the functionality of the navigation approach in a real scenario with a complex environment.

For both tests, the simulation was run on a 64-bit Windows 7 machine with an AMD Athlon 4800+ and on NVIDIA GeForce 9800GX2 with one core used for visualisation and one used for running the simulation.

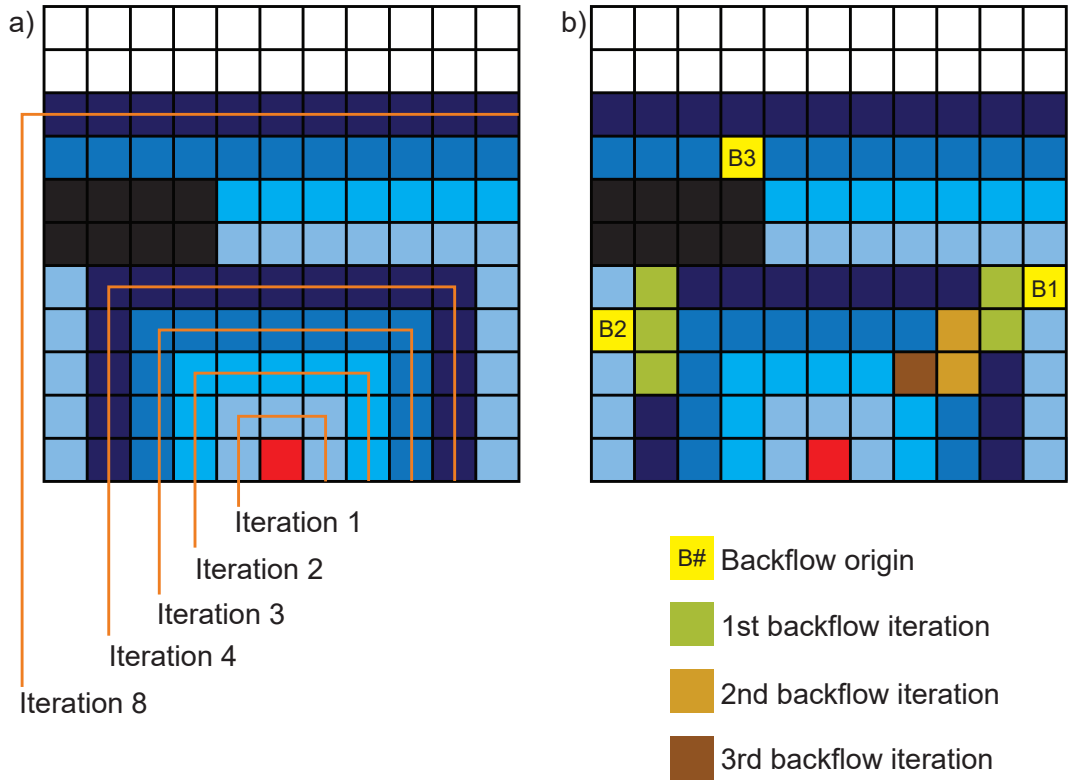

Figure 3.11 Illustration of backflow propagation in action. Take an environment that has 8 iterations of propagation (a), each blue colour signify a boundary of a wavefront propagation iteration. The colours are repeated at the 5th iteration. (b) Shows examples of backflow propagation three locations. Backflow propagation limit is three iterations. The backflow origin B1 is able to propagate the full three iterations while B2 is stopped at one iteration as it encountered an obstacle. As B3 is next to an obstacle the backflow algorithm does not run.

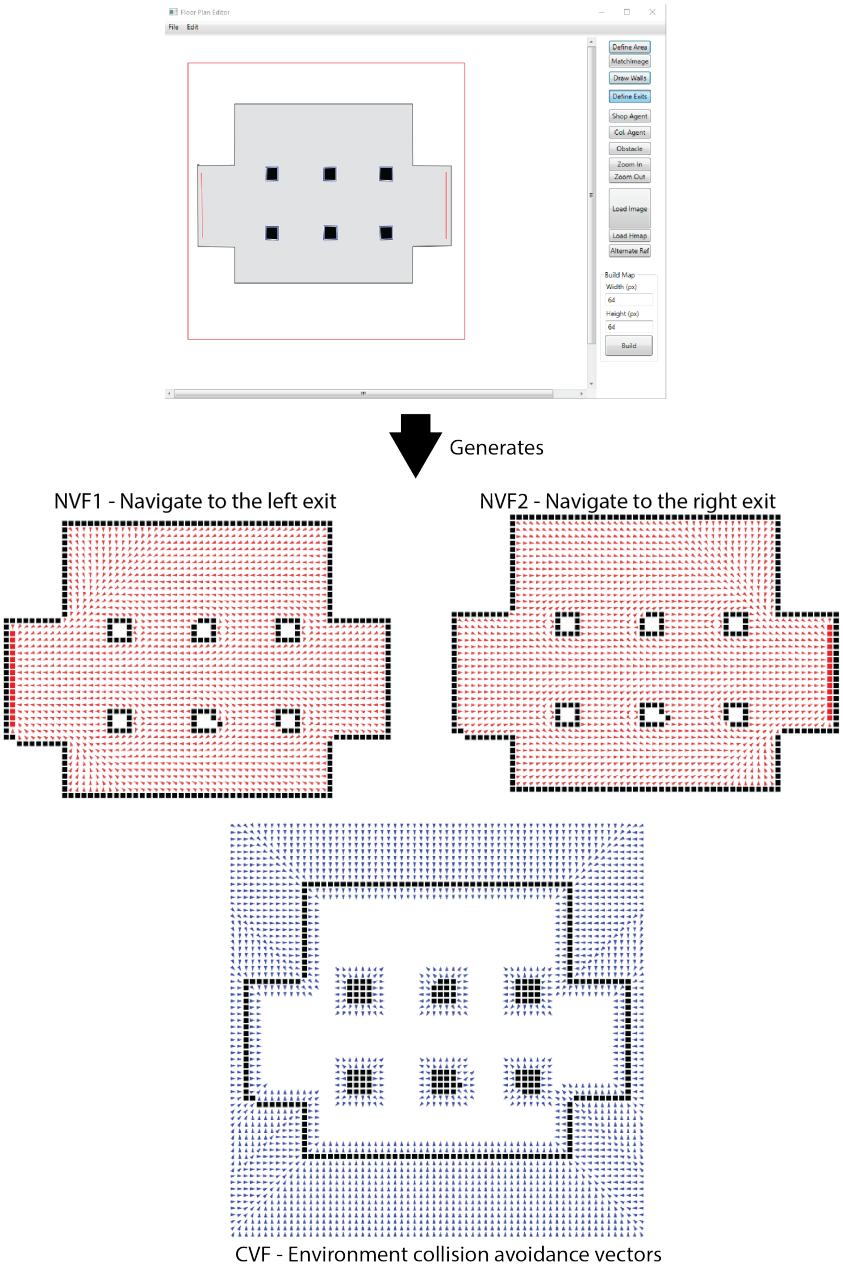

Figure 3.12 The plan editor software for generating the vector fields. The example shows a 2-exit environment that was set to be a dimension of 64x64 with maximum backflow propagation iteration of 5. Exits are shown as red squares and obstacles as black squares. The two NVFs and an CVF generated are shown where red arrows are NVF and blue arrows are CVF.

Simulation Performance Benchmarking

Navigation grids of sizes 64x64, 128x128, 256x256 and 512x512 were used for benchmarking. The environment used for testing was a blank space. Each navigation agent is configured to contain 7 NVF maps and a single CVF map to correspond with the 7 exits available in the urban test example (Fig. 3.14). Navigation agents always broadcast their `navigation_message` and pedestrian agents always receive and process them along with performing inter-pedestrian avoidance. For each grid size, 1024, 4096, 16384, 66536 and 262144 pedestrian agents were generated and placed at a constant density within the environment so that time spent processing inter-pedestrian communication for each pedestrian is roughly constant. The simulation was left to run for 200 iterations then the time of the next 100 iterations was averaged out to obtain an average time to perform a single iteration. The result of this test is shown in Fig. 3.13.

As all navigation agents need to broadcast their position every iteration, the time needed increases exponentially with the grid dimensions. The time for each pedestrian to access the navigation information takes roughly the same amount of time for each pedestrian agent. The same applies to inter-pedestrian interaction time as the density is kept constant. The graph in Fig. 3.13 shows a logarithmic increase in the iteration time is due to the fact that parallel computing efficiency is not reached until after a certain point, which is around 60,000 pedestrians in this case. At 262,144 pedestrian agents and 262,144 navigation agents (512x512 grid) the simulation is able to run at an average of 34.9ms per iteration.

From Fig. 3.13, we can see that for smaller numbers of pedestrians the difference in grid size makes a big difference to the simulation time, as the number of pedestrians increases, the size of the grid becomes less consequential. This would be especially true in environments with dense areas of interacting agents as the number of `pedestrian_location` messages can increase rapidly.

Navigation maps beyond the size of 512x512 cannot be run on larger pedestrian counts (more than 262,144 pedestrians) due to the lack of GPU memory on the test device as the list of pedestrian agents are double buffered in order to allow reallocation for agent births and deaths. The double buffering process is used in order to provide memory space for moving only alive agents to the buffered memory in parallel and then swapping this buffer memory location with the main list. Another factor that contributes to the lack of memory is the fact that as the pedestrian population gets larger, in order to keep the same density, the number of space partitions (defined by the communication radius of agents [40, 264]) gets larger and hence requires a lot more memory.

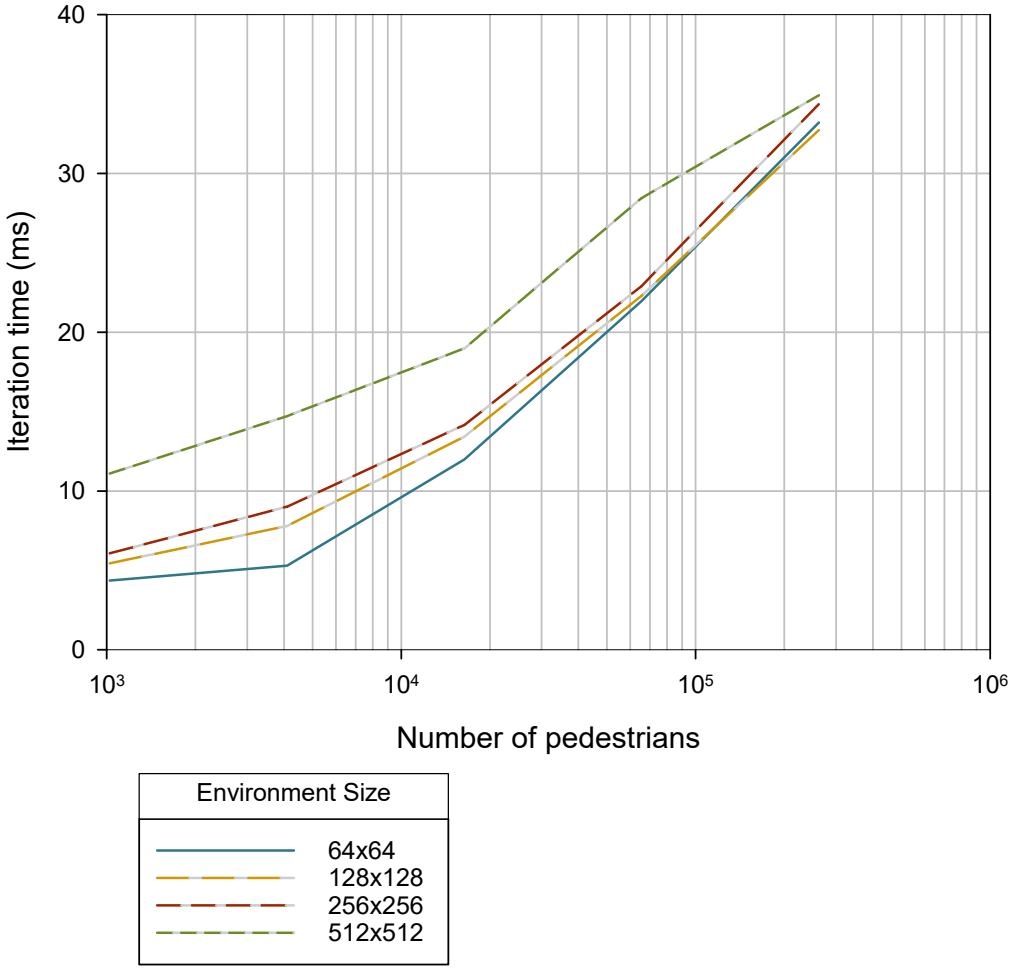

Figure 3.13 Performance benchmarking for square environments of width 64, 128, 256 and 512 measuring iteration time is measured when running with 1024, 4096, 16384, 65536 and 262144 pedestrian agents.

Simulation Based on a Real Urban Environment

The test environment was based on a busy pedestrianised area in central London. It contains 7 destinations with one being an access to the underground station (Fig. 3.14). For this simulation, the navigation agents highlighted by the red lines are marked as both entrances and exits. Pedestrians are generated at each entrance according to a specified emission rate in proportion to the size of the entrance and each has a specific destination based on a pre-determined probability. All exits can be in the state of either open or closed. When a pedestrian reaches a closed exit, it will randomly choose a different exit and start navigating to the alternate destination. All vector fields are pre-generated according to Section 3.1.2 with the whole environment of approximately $120m^2$ being represented as a 128x128 grid although walkable space occupies less than 70% of the total area. The resolution of 1 pixel $\approx 1m^2$ was chosen due to the fact that an outside environment is represented with a relatively small amount of small obstacles or narrow corridors. The scenario of the simulation shows the effects of areas that have been flooded with pedestrians possibly as a result of an evacuation or a large public event.

The model scenario has the following sequence. First, the simulation starts unpopulated and pedestrians are continually generated until they reach a steady state. The model is then flooded with pedestrians while an exit is closed. The inability of pedestrians to leave the environment in a timely fashion results in heavy congestion and potentially dangerous conditions.

Fig. 3.15 shows a breakdown of simulation time (in ms) during a real time simulation of the London model. More specifically it shows the timings (stacked) for individual or combinations of agent function calls (Fig. 3.2). Timings have been obtained by averaging the simulation time of each function call over the period of 200 simulation steps to avoid any small fluctuations. Over the simulation of roughly 9000 iterations the graph shows three distinct phases. The first occurs between step 0 and 800 and shows relative stability in simulation times. During this phase the emission rate for each exit in the model is constant and the total number of pedestrians remains at roughly 3000. The sharp rise in simulation time after this phase is a result of increasing the emission rate of all the exits. The second phase of relative stability occurs between steps 1400 and 4400. During this phase of the simulation the large increase in emission rates has reached a stable population of roughly 11,000 pedestrian agents. The final phase of the simulation shows the effect of closing one of the 7 major exits. Any agent that reaches a closed exit is forced navigate to a new exit. The effect of this is that first the population increases relatively slowly as a result of pedestrians failing to leave through their preferential exit (steps 4400 to 5600). After this the area

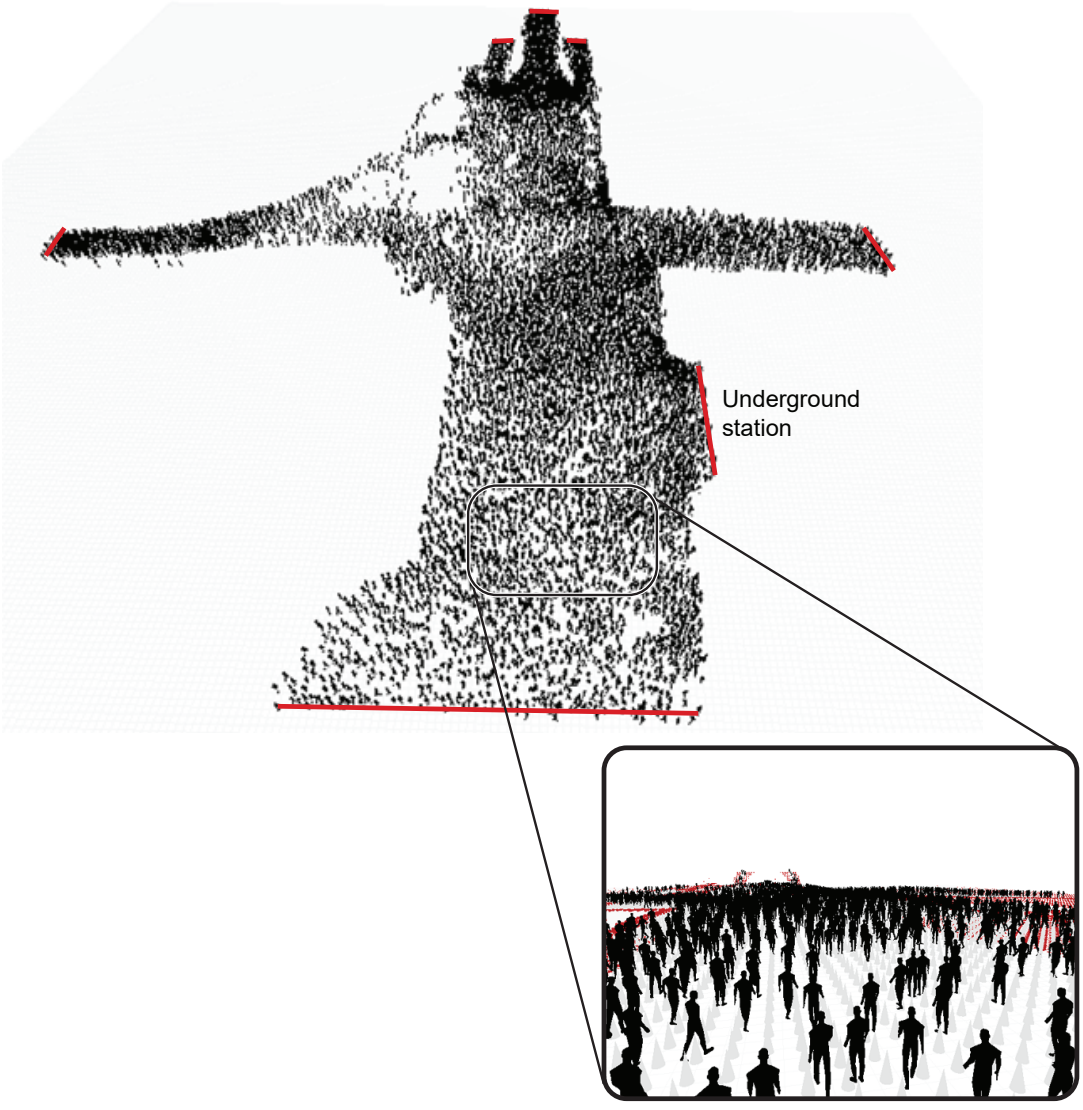

Figure 3.14 Screenshot of the urban environment simulation running with 10,000 pedestrians. Red lines signify locations of entrances and exits.

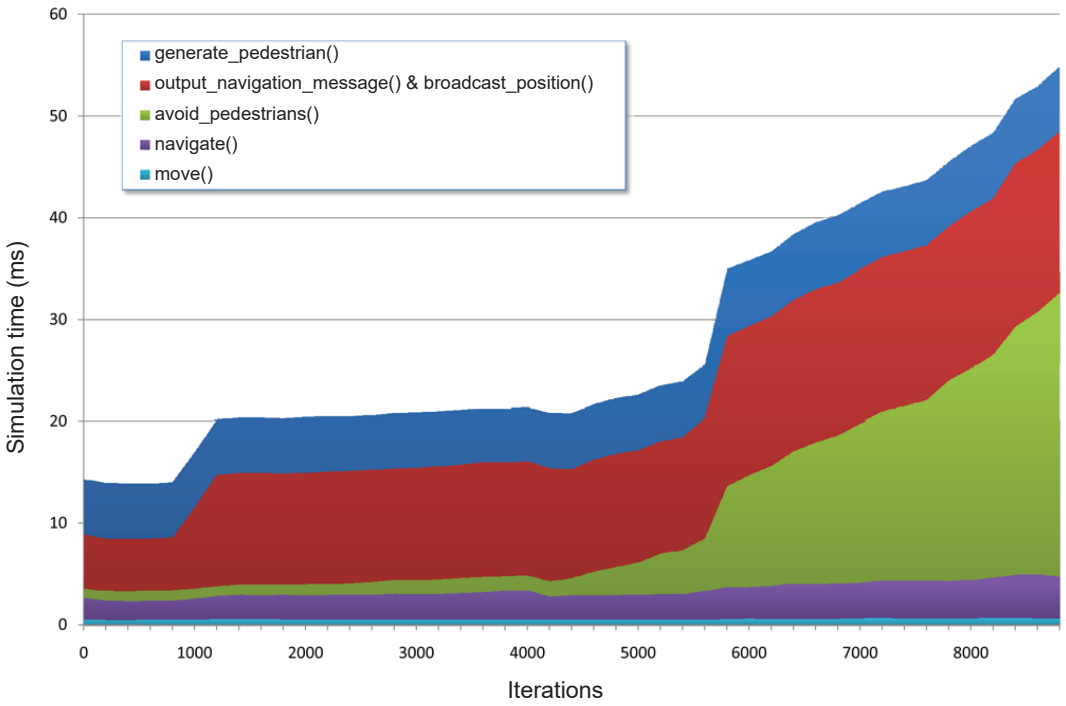

Figure 3.15 Time utilization by different agent functions with increasing pedestrian agent density

around the exit becomes very congested and the number of pedestrians continues to rise very sharply to just over 20,000 agents. The effect of closing the exit congests the model to the extent that a stable number of pedestrians cannot be reached.

It can be seen that after the first 1200 iterations, all agent function times remain relatively stable and that functions related to navigation form a higher percentage of the simulation time. As pedestrians start to congregate and the density increases from around iteration 6000, the cost of inter-pedestrian interaction rapidly increases and even overtakes the computation used for navigation.

The agent-based navigation grid approach provides many advantages, such as having pedestrian and navigation agents in the same model framework, pre-resolved paths for pedestrians from anywhere in the environment and high performance for very large numbers of pedestrians. However, there are also disadvantages. As the system uses a dense data structure, it will always be limited by exponential memory requirements with increasing environment size. The use of navigation agents also increase computational cost as they need to broadcast their location message at every iteration. These factors become a concern when we start to look at modelling very large public facilities or even an entire city. In order to deal with the problem of scalability, a graph-based approach that stores environment data more sparsely can be used which is discussed in the next section.

3.2 Searchable Navigation Graph

As previously mentioned in Section 2.2.2, using a graph-based navigation approach generally involves partitioning areas of the physical environment into a set of nodes containing bounded areas. These areas are connected by node edges. The area within a node can be of any arbitrary shape resulting in a graph that is invariant to the environment size. By using sparse data structures, there is no direct relation between dimension of the environment to the memory required. The low memory use makes the approach suitable for scenarios where many pedestrians have varying and multi-stage objectives.

There are many implementations of graph-based navigation (Section 2.2.2). However, none looks into the effects of using the approach during congestion where pedestrians may need to make a minor route changes implicitly to avoid getting stuck trying to follow a pre-assigned route. In addition, although there are various navigation and path planning research on the GPU that can be used for pedestrian simulation [265–269] it does not provide a complete system that allows for complex, multi-stage navigation.

The searchable navigation graph presented in this section is a novel graph-based navigation system that uses searchable navigation graphs to provide dynamic probabilistic navigation on pre-resolved routes. The work combines the advantages of both grid and graph-based methods. Sparse navigation graphs are constructed using a subdivision approach which improves coverage of walkable areas while using significantly less memory than the grid-based alternative. Routing to a destination is fully resolved for the whole environment so pedestrians are able to navigate to a desired destination from any point in the environment. This navigation information, also contained in a graph, is spatially partitioned and made searchable to prevent the possibility of pedestrian agents having to iterate through the entire graph. This searchable navigation graph also allows for dynamic re-routing in congested areas, which is applicable to congested crowds that do not follow strict corridors.

The searchable navigation graph is based on the work of Pettré et al. [47] where the environment is divided up into inter-connected circular areas forming a disk graph. While the approach by Pettré et al. [47] goes further to form corridors from the bisection of overlapping circular areas of the disk graph nodes. This work however, takes a different approach and uses the disk graph by itself as a means of navigation.

Given a scenario, a definition of the physical environment and a set of high-level navigation behaviours, the process for creating the navigation data structures takes place over three main stages. A simple T-shape environment with a scenario shown in Fig. 3.16 illustrates these different stages.

The first stage is the generation of the Environment Graph (Section 3.2.1) which is a graph-based representation of the physical environment and shows which areas are connected and traversable (Fig. 3.16(b)). The second stage is the creation of the Itinerary Graph (Section 3.2.2) that provides a high-level sequence of location-based objectives defined in the Scenario that pedestrians are to follow (Fig. 3.16(c)). Lastly, the final stage uses the Environment Graph and the Itinerary Graph to find the optimal routes between objective locations (e.g. from Entrance 1 to Exit 1) in order to generate the Navigation Graph (Section 3.2.4) that contains the concise paths that pedestrians follow to reach their destination (Fig. 3.16(d)). Additionally, static obstacles are arranged in a separate searchable tree to be used for collision avoidance during the simulation.

The work was implemented in CUDA as an extension to the FLAME GPU framework and provides an API which is accessible from agent functions within the framework. Chapter 4 gives further details of how this navigation system is integrated into the overall pedestrian simulation system. Examples are given in Section 3.2.8 to show that the approach is memory efficient compared to the grid-based approach, while still being able to provide dynamic routing for the entire environment.

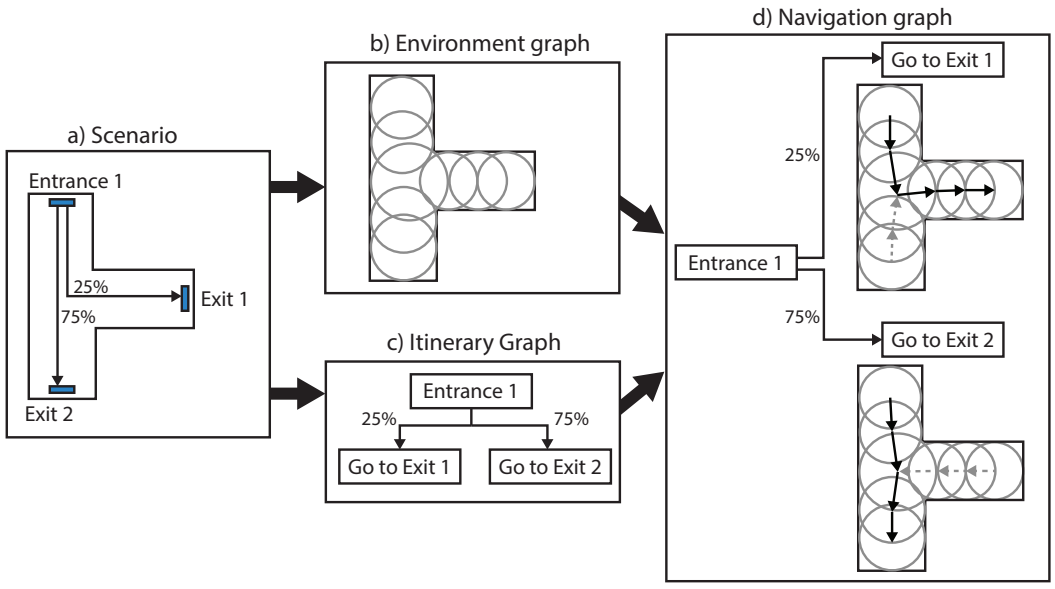

Figure 3.16 a) For a given scenario, graph data structures are created for use in the simulation. b) The Environment Graph is a representation of the environment and how each part is connected. c) The Itinerary Graph represents a high-level set of objectives. d) The Navigation Graph contains the precise sequence of graph nodes that must be traversed in order to reach an objective location.

3.2.1 The Environment Graph

The Environment Graph contains a representation of pedestrian walkable areas in the physical environment. Only circular areas are used to represent the walkable area as the shape fits well within abstract shapes and bounds can be detected quickly. An R-tree [270], a spatial search tree, is constructed for the Environment Graph to allow faster retrieval of navigation information during the simulation.

The environment is discretised into a binary grid where the cells are either walkable or non-walkable to a pedestrian. In Fig. 3.17(b) walkable areas are shown as white and non-walkable areas shown as grey. A Euclidean distance transform is then performed for all walkable cells to find the nearest distance to non-walkable cells. Fig. 3.17(c) illustrates the result of the distance transform with darker green areas indicating further distance from a wall or static obstacles. Each cell's distance transform value effectively becomes a "clearance radius" where it is guaranteed that there are no static obstacles within the radius.

To provide maximum coverage of the environment areas using the least number of cells, a cell with the highest clearance radius is chosen and any other cells within the radius are rejected. The selection continues until there are no more cells above a minimum size of $0.25m$. Each of the chosen cells becomes a node in the Environment Graph and a connection is made between all nodes overlapping radii. The result of this process is shown in Fig. 3.17(d) where the final set of chosen areas can be seen as overlapping circular shapes covering a majority of the walkable area.

To obtain navigation information during simulation, the Environment Graph is queried to find the node each pedestrian is in or nearest to. An R-tree is used to accelerate retrieval of node information. Each node in a tree can have a maximum of three child nodes and contains a bounding box that completely encapsulate its child nodes. The method for construction of the tree is outlined in the paper by Guttman [271]. The process is demonstrated visually in Fig. 3.18 where at the initialisation stage, a root node is created that has a bounding box covering the entire area (Fig. 3.18(b)). Note that each node in the R-tree can store a reference to three other nodes and a node in the R-tree can either be a bounding box node or an Environment Graph node. The first iteration divides the space along the x axis where we get the bounding box areas $B1$ and $B2$ (Fig. 3.18(c)). The second iteration divides the space long the y axis and $B1$ is further subdivided into $B3$ and $B4$. There is no need to further divide the space in $B2$ as there are exactly three nodes within the boundary (Fig. 3.18(d)). Should further subdivisions be needed, the division axis alternates every iteration until the partitioning is done. The final R-tree is shown in Fig. 3.18(e).

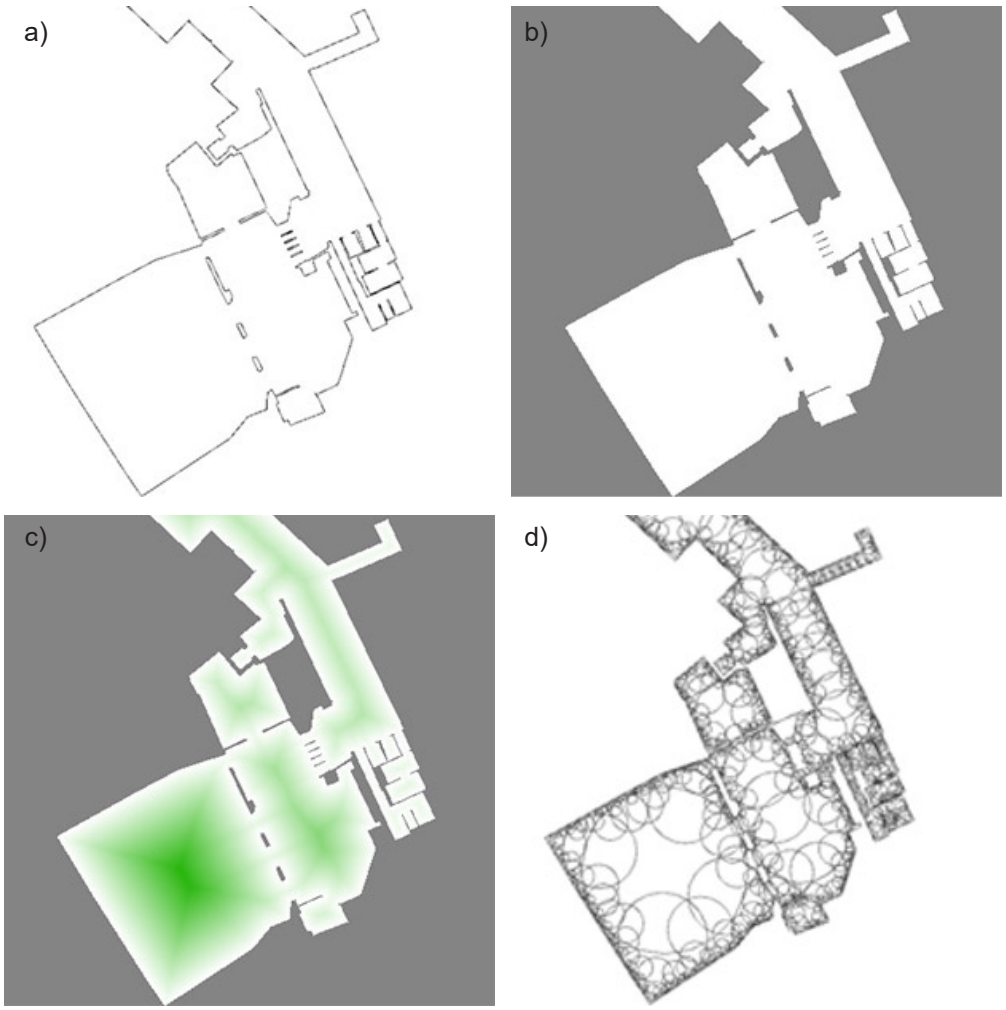

Figure 3.17 The figure shows the process of creating the Environment Graph. (a) The outline of the environment in black. (b) The environment discretised to a binary grid where the white colour is a walkable area and the gray colours are non walkable. (c) The distance transform where deeper green indicates higher distance transform value. (d) The nodes with cicular areas created for the Environment Graph.

3.2.2 The Itinerary Graph

The Itinerary Graph is a directed acyclic graph used to express each pedestrian's goals as a sequence of location-based objectives (Fig. 3.19(b)). Each node of the Itinerary Graph is either a single objective or a portal which can act as both entrance and exit to the simulation. All nodes are associated with a bounding geometry which is used to check when the pedestrian has arrived at the objective location. Although the bounding geometry could be any arbitrary shape, only circular, rectangular and capsule shapes (composed of two circles and a rectangle) were used in order to enforce simple convex geometry, thus limiting the time needed for bounds checking.

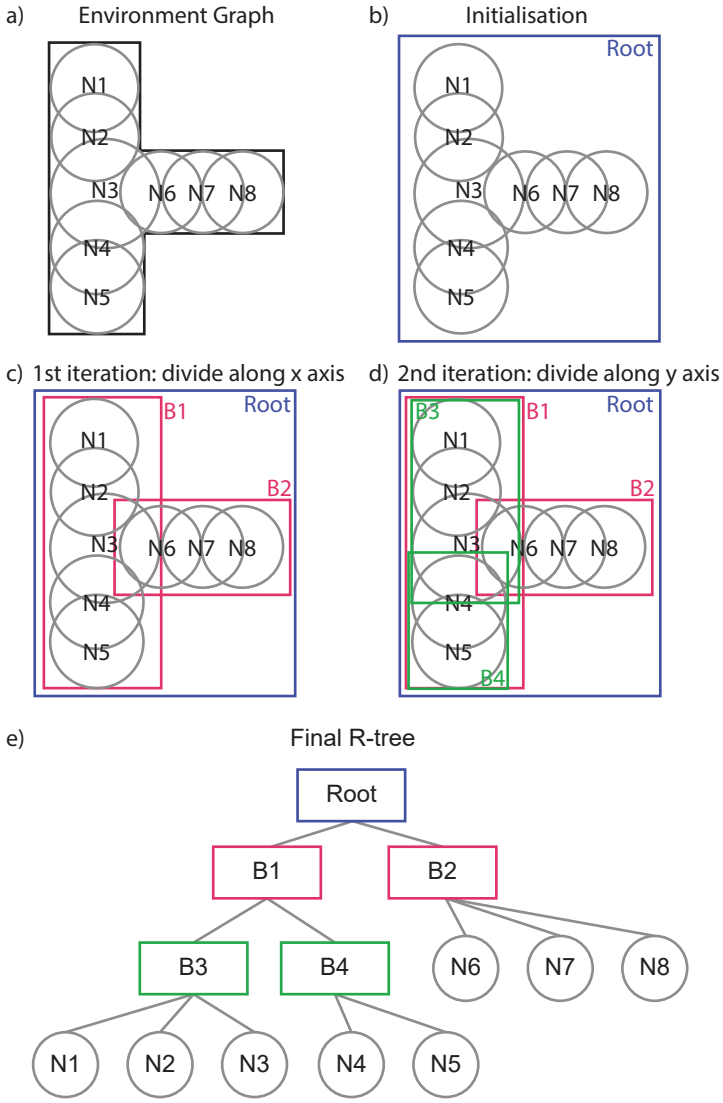

Figure 3.18 Generation of an R-tree from the Environment Graph in Fig. 3.16(b).

A connection between nodes within the Itinerary Graph signifies the travel between the objective locations but not the exact path. It is possible for objectives to diverge and this is represented as a branch in the graph. When branching occurs, the probability for selecting the next objective is assigned to each edge of the branch. The approach allow the itinerary to be shared for pedestrians coming from and leaving at different portals as long as they share the same objective selection probabilities when encountering the same branch points. When pedestrians' goals are too different such as "catch a train" or "leave the station", multiple Itinerary Graphs must be created for each of the goals.

Fig. 3.19(b) show an example Itinerary for the more complex environment used in Fig. 3.19(a), which has a shop and two exits. According to the itinerary, the pedestrian

will first go to the Shop and afterwards has a 50% probability to go to either Exit 1 or Exit 2.

3.2.3 Routes Generation

A Route refers to a connection between two objective locations, e.g. going from the Shop to Exit 1 in Fig. 3.19(b) is a single Route. Routes are defined during the creation of the Itinerary Graph and can also be considered as representing the edges of the graph. However, at this stage, they do not contain the exact sequence of nodes to allow navigation from one location to the other. Before the Navigation Graph can be generated, this exact sequence of nodes must first be found.

For every Route in the Itinerary Graph, an A* [144] search of the Environment Graph is performed. Inter-node distance is used as the heuristic in order to find the shortest path between objective locations. Fig. 3.19 illustrates this process. The Environment Graphs shown in Fig. 3.19(a), with an Itinerary Graph as in Fig. 3.19(b). This has three connections and three Routes are generated, as shown in Fig. 3.19(d, e and f). The sequence of Environment Graph nodes in a route is uni-directional in order to enforce a direction of travel.

3.2.4 The Navigation Graph

The Navigation Graph contains detailed routing information between objective locations specified in the Itinerary Graph and is used by pedestrians during simulation to obtain routing information (Fig. 3.19(g & h)). The Navigation Graph (Fig. 3.19(d & e)) shares the same number of nodes as the Environment Graph (Fig. 3.19(a)), also retaining the same index position, physical location and radius. The edges in Navigation Graphs are, however, uni-directional connections which are used to indicate the direction of travel rather than bi-directional connections as found in the Environment Graph. The edges retain their Route identification and their Route probability which makes it possible for multiple Routes to overlap over a single node. Having a Navigation Graph node inherit its physical location, radius and index from the Environment Graph node allows the Navigation Graph to be searched using the R-tree constructed for the Environment Graph. As with grid-based navigation (Section 3.1), all nodes within the Navigation Graph contain navigation information.

Generation of the Navigation Graph begins by taking the Route information generated previously, as described in Section 3.2.3. For each Route, the connections between the sequence of nodes that form the Route are added as edges to the Navigation Graph.

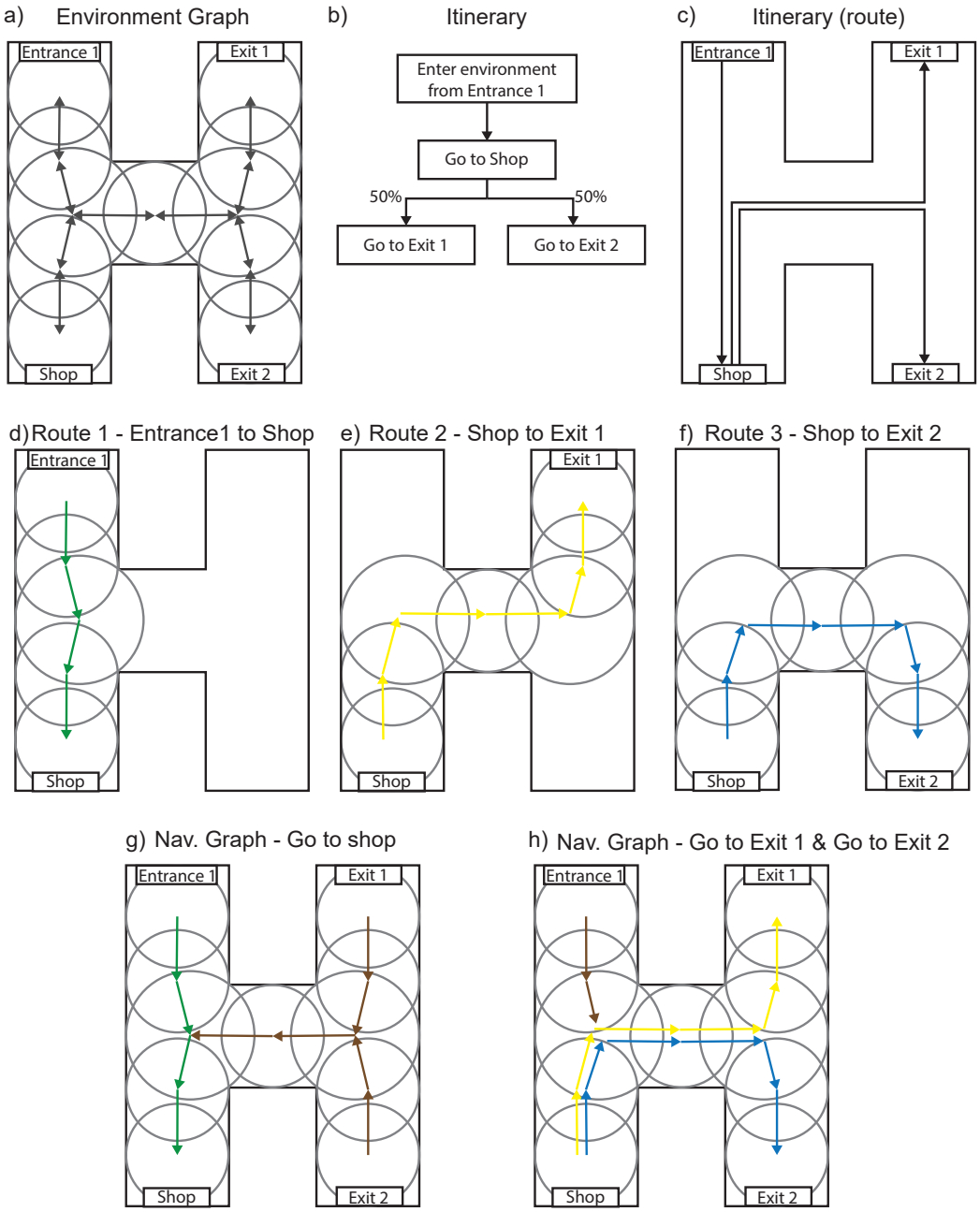

Figure 3.19 The process of creating the Navigation Graph. (a) The Environment graph for an example environment with an entrance, a shop and two exits. (b) The Itinerary Graph. (c) The routing in the physical environment. (d, e & f) The exact sequence of nodes is found for each Route in the Itinerary. Two Navigation Graphs are required for this Itinerary as the nodes overlap in opposite directions. (g) The Navigation Graph from Entrance 1 to the shop (green arrows). (h) The Navigation Graph from the Shop and branches to either Exit 1 (yellow arrows) or Exit 2 (blue arrows). For both Navigation Graphs the brown arrows guide pedestrians back to the itinerary routes.

For all nodes in a route that overlap the Itinerary Graph node's bounding geometry, a reference to the geometry is added to the node as a Navigation Target (Section 3.2.5).

The routes found in the previous step are unlikely to utilise all nodes in the Navigation Graph, e.g. in Fig. 3.19(g) the green arrow indicates the route found from Entrance 1 to the Shop. A second routing process is performed to provide navigation information that can lead pedestrians to these routes. A Dijkstra search is performed starting from the Navigation Graph nodes with routing information and spreads to the rest of the nodes. Optimal paths for returning to the routes are then resolved from all locations in the environment. The result of the second routing process is shown in Fig. 3.19(g) as the brown arrows, which provides navigation for any other nodes not covered by the green arrows.

Similar to grid-based navigation, multiple Navigation Graphs can be used in sequence to capture a more complex set of objectives or when routes are circuitous. An example can be seen in Fig. 3.19(c), where the itinerary involves going to a location which requires the routing to double back on itself, i.e. Route 1 (Fig. 3.19(d)) overlaps with Route 2 (Fig. 3.19(e)) and Route 3 (Fig. 3.19(f)). Two Navigation Graphs are generated, one for going to the first objective, the Shop (Fig. 3.19(g)), and the second for doubling back and going to the second or third objectives, Exit 1 or Exit 2 (Fig. 3.19(h)).

3.2.5 Other Properties of the Navigation Graph

There are three other properties associated with the Navigation Graph that help in the implementation of specific behaviours. The properties of Navigation Targets, Queues, and Waiting Areas are discussed in this section. Chapter 4 provides an example of a model and scenario which uses these three properties.

Navigation Targets

The location of a pedestrian's destination may not lie exactly at the centre of the Environment node. A Navigation Target is extra information that can be attached to nodes in the Navigation graph. It encapsulates the exact geometry of the destination within a convex shape and contains other information such as whether it is an exit, or a teleport, or that it has a delay time which causes pedestrians to be removed from the simulation, or directly moves a pedestrian from one location to another (e.g. moving between floors) or introduces a user-specified delay.

Queues

A Queue can be attached to each objective location and provides a location of where the queue starts and how the physical queue takes shape. The Queue is represented as a multi-section line (polyline) with one end indicating the start of a queue. A reference to the Queue is added as a property of a Route that has the associated objective location as a destination during the creation of the Navigation Graph.

Waiting Areas

A Waiting Area can be attached to each objective location. This indicates a location where pedestrians navigate to, in order to wait until they are allowed to reach the objective location the Waiting Area is attached to. The Waiting Area is represented as a 4-sided polygonal shape and its reference is added to a Route that has the associated location as a destination during the creation of the Navigation Graph.

3.2.6 Implementation as a Navigation Module in FLAME GPU

This section gives an overview of how the searchable navigation graph is implemented on the GPU platform. The FLAME GPU framework is used as a base for simulation of the pedestrian model but as it does not support the graph-based navigation approach by default, a Navigation Module extension is created in CUDA to provide function calls that pedestrians can make directly from within the FLAME GPU model. The main function of the module is to provide global navigation but it also integrates static obstacle avoidance, waiting and queueing. The difference in architecture between CPU and GPU means there were three major considerations that affected the final design of the module.

The first is related to the memory access pattern on the GPU. While it is normal on the CPU to lay out data structures following the Array of Structures (AoS) pattern, as discussed in Section 2.5.1, the data structure needs to be converted so that it follows the Structure of Arrays (SoA) pattern. By laying out the data structure this way, i.e. coalescing the data, the GPU will be able to cache more relevant variables in a single memory read. This ultimately leads to lower memory reads overall.

The second consideration is on how data structures link to each other and is closely related to the first consideration. On the CPU, it is common that links between objects are done using memory pointers. However, by following the SoA pattern, each variable in an object is an array. In addition, as the data structures needs to be serialised in order to be uploaded to the GPU, the memory address of the objects will change, causing

the memory pointers to fail. Because of this, objects are connected by indices rather than memory pointers.

The third consideration is the inefficiency of dynamically allocating global memory on the GPU device and the smaller size of GPU RAM in comparison to CPU systems. The system is designed so that pedestrians use a minimum and fixed number of variables from pre-allocated global memory to track their navigation progress.

The Navigation Data (Fig. 3.20) is a set of data structures that is designed with these considerations. Fig. 3.20 shows how some objects have variables that store indices that link it to another object. For the Navigation Graphs, many graphs are stacked into a single array, but the number of nodes in the graph is always the same. In this case, an offset is also used along with the index. Wherein, the index refers to a particular node in the graph, the offset is the number to be added to the index in order to get the correct node in the correct Navigation Graph. At the start of the simulation, the data structures are converted to the SoA pattern (Fig. 3.21) before they are uploaded to the GPU.

Once the Navigation Data is uploaded to the GPU, it is never modified from the device. It is the pedestrian agent and its agent variables that keep track of the progress of the current navigation. The pedestrian agent uses the `nm_navigate()` function that takes reference pointers of the input parameters so that these parameters can be modified inside the function and used as output. It is only essential for a pedestrian agent to store references to its itinerary, current route, current and next Environment Graph node indices, and its current status (e.g. the pedestrian is waiting or that a destination is reached). It is only when the pedestrian falls outside of the bounds of the referenced Environment Graph nodes that an Environment graph search operation has to be performed.

The R-tree created for searching the Environment Graph uses the stackless traversal approach by Popov et al. [272] in order to search the tree with constant memory. The searching of the Environment Graph returns the index to the Environment node that the pedestrian's inside (or closest), this index is combined with the offset previously mentioned in order to get the correct Navigation Graph Node to follow.

Each Navigation Graph Node can also be associated with Navigation Targets (Section 3.2.5). This forces the pedestrian to navigate to the `Shape`, a convex shape bounds, of the associated Target. After the bounds of the shape has been entered, the action associated to the Navigation Target is carried out such as exiting environment, teleport to a different floor, wait for a specified time or switch to another Navigation graph.

Each Navigation Graph Node can have multiple outbound connections belonging to multiple routes (Fig. 3.22), these are stored in a separate object called Navigation

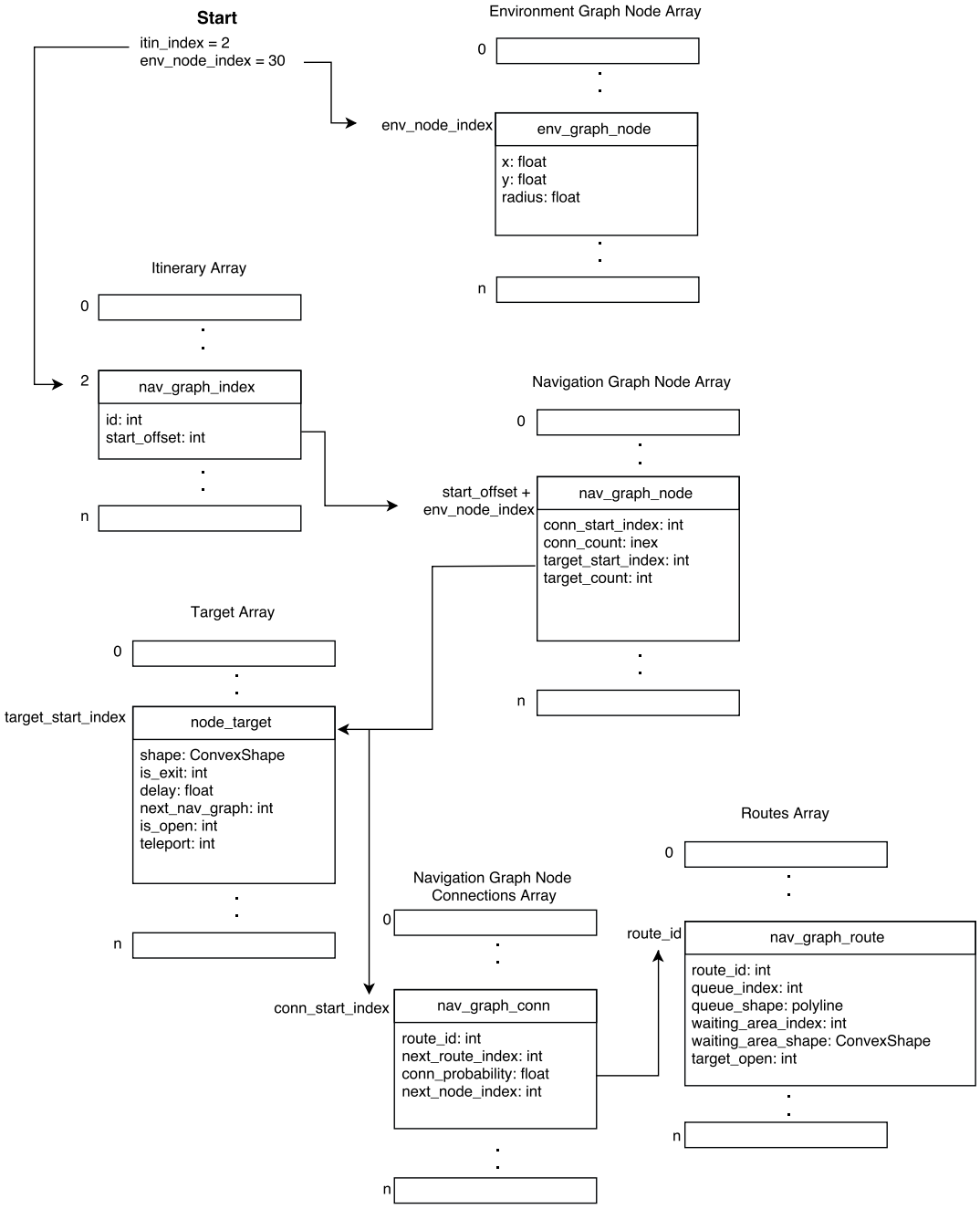

Figure 3.20 A simplified information flow diagram of the Navigation Data structure.

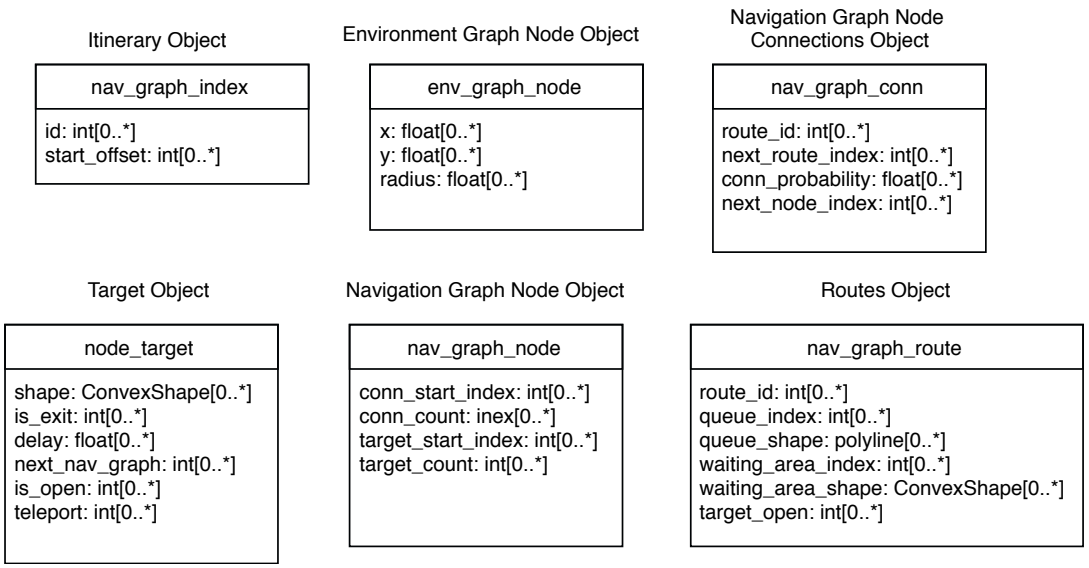

Figure 3.21 The Navigation Data Structure Fig. 3.20 laid out as SoA objects for use on the GPU.

Graph Node Connections. The Table 3.1 shows an example of the values that can be found for the configuration shown in Fig. 3.22. The `nm_navigate()` function iterates through these connections, first trying to find a connection with the same route id that the current pedestrian has, otherwise it will fall back on the last route id encountered. Having decided the route id to follow, if there is branching, a random number is cast. If the random number is lower than the probability value of the connection, the pedestrian will follow that connection. Having chosen a route id, the pedestrian also accesses the Routes Object to obtain information about possible queues within the Route or check if the Route's ultimate destination has an associated Waiting area.

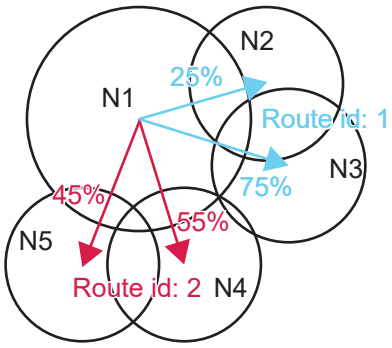

Figure 3.22 An example of branching in the Navigation Node Connections Array. The pedestrian is at node N1 and there are two routes with IDs 1 and 2 each having 2 branching possibilities.

array_index	route_id	next_route_index	conn_probability	next_node_index
0	1	2	0.25	N2
1	1	2	1.0	N3
2	2	-1	0.55	N4
3	2	-1	1.0	N5

Table 3.1 The values that would be found in the Navigation Node Connections Array for the example in Fig. 3.22.

Having chosen the next Navigation Graph Node to navigate to, the `nm_navigate()` function determines navigation direction using the process described in the next section. Finally, a normalised navigation vector is returned to the pedestrian that indicates the current direction in which the pedestrian should be heading as well as a flag for activating a Navigation Target event.

Determining Navigation Direction between two Navigation Graph Nodes

In order to determine a pedestrian's navigation direction once they have obtained a reference to a navigation node, a trapezoidal navigation area (red lines in Fig. 3.23) is generated from the two Navigation Graph nodes' diameter line that runs perpendicular to the line connecting the two nodes' centre points. The pedestrian's navigation goal is directed to the location on the next node's diameter line that is of proportional distance away between the centre connection line and the top or bottom edge of the navigation area, as shown by the blue dotted line in Fig. 3.23. Should the pedestrian be outside of the navigation area, they are redirected to the nearest point on the line of the side nearest to the pedestrian.

Adding Dynamic Objective Locations

The ability to add new objective locations dynamically is useful in certain scenarios, for example, when a model represents a fallen pedestrian that needs to be rescued. As long as the environment layout does not change, it is a simple case of generating a new Itinerary with the fallen pedestrian as a destination and generating a Navigation Graph from the itinerary. The generated Navigation Graph can then be appended to the existing Navigation Data arrays. As long as the pedestrian has the index to the newly created Navigation Graph, it will be able to navigate to the fallen pedestrian. In the case where dynamic events also cause the environment layout to change, currently all the Navigation Graphs must be generated anew and re-uploaded to the GPU.

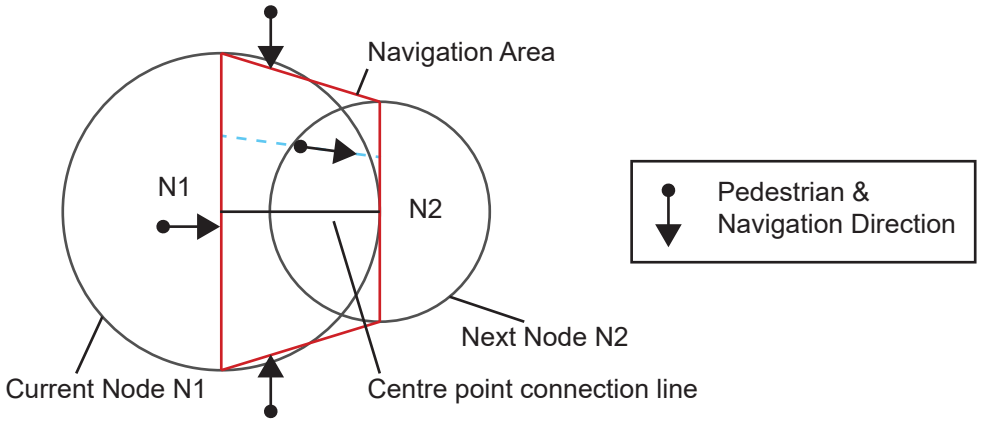

Figure 3.23 During the simulation, a trapezoidal shape navigation area (red line) is formed from the two diameter lines of the current and next node and used as a navigation guide. Pedestrians outside of the navigation area try to navigate towards it while ones inside the navigation area will try to keep a proportional distance to the centre line between the current and the next node (blue dotted line).

3.2.7 Static Obstacles & Environment Bounds Detection

For engineering applications, it is essential that environment bounds are represented as accurately as possible to allow pedestrians to utilise the entire space should congestion occur. In order not to mitigate the scale invariability of the navigation system, bounds are represented as lines in continuous space and are stored in a searchable K-d tree as used in the Optimal Reciprocal Collision Avoidance (ORCA) approach by Van Den Berg et al. [129].

3.2.8 Evaluation and Discussion

This section discusses the advantages of using the searchable navigation graph-based approach in combination with a graph-based obstacle avoidance system such as ORCA [129]. The section is divided into two parts. The first part investigates the effect of congestion in relation to the environment boundary representation and how it effects the pedestrian flow. The second part then discusses a case when pedestrians are pushed towards an unintended destination due to congestion and how the searchable navigation graph approach solves the problem.

Effects of environment shape during congestion

Full utilisation of the entire walkable area makes a substantial difference in the simulation results. Existing research [100, 273, 274] shows that the congestion pattern has an effect on flow rates and many pedestrians competing to get through a narrow exit at the

same time actually makes the flow slower. The congestion can be reduced by changing the shape of the corridor, such as making it a funnel or zig-zag shape, and slowing the pedestrians who are trying to push in from the sides [100, 273, 274]. Placing of pillars in front of exits also helps to slow down the flow and reduce the congestion.

To investigate the effect of corridor shape on the pedestrian flow and congestion pattern, a funnel shape corridor is used. Fig. 3.24(a) shows an environment consisting of a large corridor, 10m in width, connected to a narrower one that is 1m in width. Pedestrians are emitted from the left side and walk towards the exit on the right side in the narrow corridor. The emission rate is kept constant at 7 pedestrians per second which is enough to cause a steady build up of congestion. It can be seen that an arch congestion pattern (as observed by Helbing et al. [22]) is formed when too many pedestrians (solid circles) try to exit through a narrow corridor at the same time.

For the approach by Pettré et al. [47], bisection lines between two Environment Graph nodes (red lines in Fig. 3.24(a)) are used to also restrict the pedestrians' walkable space. A simulation was created to show the effect of this change in the pedestrian flow under congestion when the bounds becomes a funnel shape instead of utilising the entire walkable space (Fig. 3.24(b)). The narrow corridor width is kept the same size as the original environment to provide a fairer comparison. The funnel shape causes the congestion pattern to be very different, which in turn changes the flow rate of the pedestrians themselves, as observed by Helbing et al. [118], where it is shown that a funnel shape is an optimal environment design for increasing pedestrian flow and reducing congestion.

To confirm the change in pedestrian flow between the two corridor shapes, a comparison is made between the two environments by measuring the number of pedestrians that have exited the environment over time. This is shown in Fig. 3.25. After peak congestion near the narrow corridor is reached (approximately 50 seconds, Fig. 3.24) the exit rate is shown to be roughly stable for both environments. From the graph in Fig. 3.25 it can be seen that the exit rate in the Funnel environment (dashed line) averages at 2.89 pedestrian exiting per second while the original environment (solid line) has only 1.18 pedestrians exiting per second. This shows that environment shape can greatly affect flow during congestion.

Minor route change during congestion

In locations with multiple narrow corridors side-by-side, e.g. turnstiles or ticket barriers (Fig. 3.26(a)), under congestion it is possible for pedestrians to be pushed to the side and outside of all areas within the route. It is also possible for them to get pushed into a different corridor as more pedestrians arrive from behind and restrict

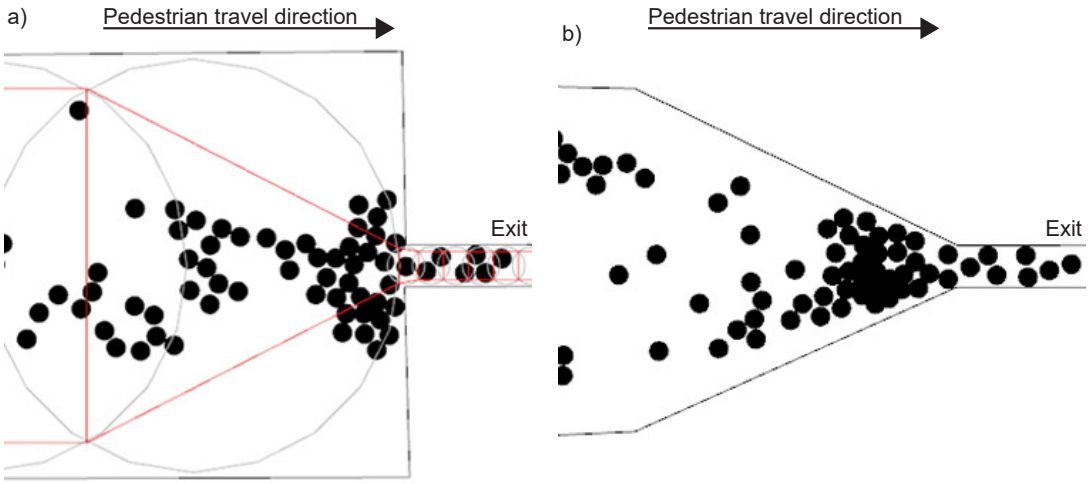

Figure 3.24 Comparison between the shape of corridors on pedestrian flow during congestion. (a) During congestion, formation of an arch pattern when moving from a wide passage to a narrow corridor or doorway is shown which matches the results of Helbing et al. [22]. Solid circles are pedestrians, black lines mark the environment boundary, grey lines represent the Environment nodes used in the route, red lines marks the gate/corridor and shows how the environment is bounded if following the approach by Pettre et al. [47]. (b) A simulation when the environment is bounded by corridors creating a funnel shape. The exit corridor size is adjusted to be the same as the first case to provide a fair comparison of the exit flow rate.

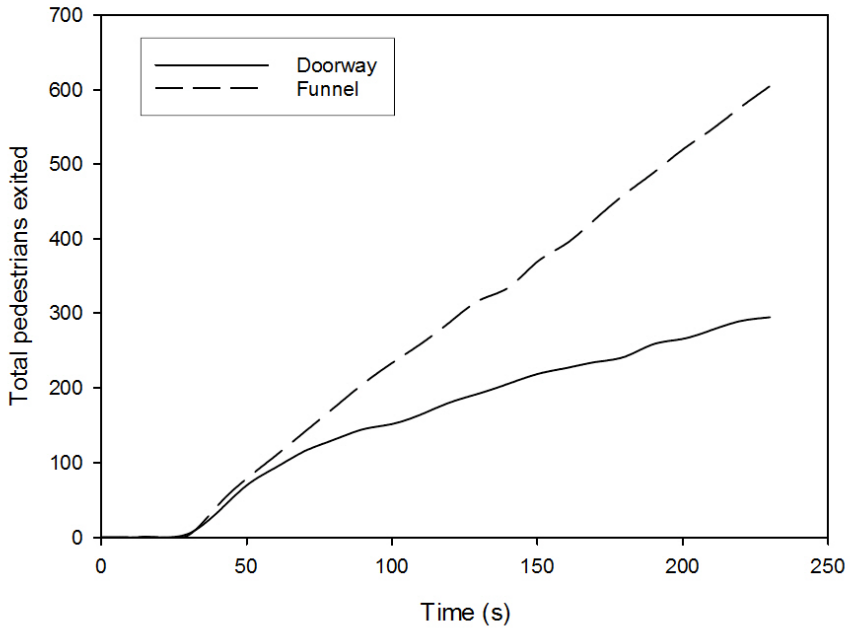

Figure 3.25 The shape of the corridor causes a large difference in flow rate. It can be seen that the funnel shape allows for pedestrians to exit from the environment much faster.

the ability to navigate back to the desired route (Fig. 3.26(b)). If the pedestrian only has a fixed route though the environment and no other navigation context, they will be required to go back on themselves and cause further congestion in the narrow corridor (Fig. 3.26(c)). While it is possible to dynamically calculate a new route to the destination, this means running the A* algorithm to the destination each time a new route is needed, which can be costly especially with a larger Environment Graph and large numbers of pedestrians. Pre-calculation of all possible routes for each pedestrian separately means redundancy in storing the parts of the routes which are the same and would still require graph traversal to find a new route for the pedestrian without a searchable graph structure. If binary search is applied, the algorithmic complexity of the operation would be $O(\log \sum_{i=1}^{r} N_r)$ where N_r is the number of nodes for each existing route r.

Having implemented a fully-resolved branching and searchable Navigation Graph, the advantage is that there is no redundancy in storing a route which may branch and converge but ultimately end up at the same final destination. When the pedestrian is outside its Navigation Graph node's bounds, it performs a lookup through the Environment Graph R-tree (Section 3.2.1) using its position, Itinerary index and Navigation Graph index in order to locate the Navigation Graph node relevant to the current position and thus obtain a new set of navigation coordinates (Fig. 3.26 (d) and (e)). The access operation to the R-tree has complexity $O(\log n)$, where n is the number of Environment graph nodes.

3.3 On the Two Navigation Approaches

This chapter presented two novel approaches to agent-based pedestrian navigation. The first is based on a grid-based navigation system using a grid of Navigation agents that acts as a set of vector field maps (Section 3.1). The second is a fully-resolved and searchable graph-based navigation system (Section 3.2). The following subsections will compare these in terms of memory use and navigation performance.

3.3.1 Memory Use

In order to compare the memory use of the two approaches, two environments were used with sizes varying from 10x10m to 500x500m (Tables 3.2 and 3.3). The first is a plain Wedge-shaped environment where the left side is fixed at 4m and the right side is the same length as the environment, e.g. for a 10x10m environment, the right side is 10m in length. Fig. 3.27 shows the environment at 10x10m and 25x25m. The Wedge

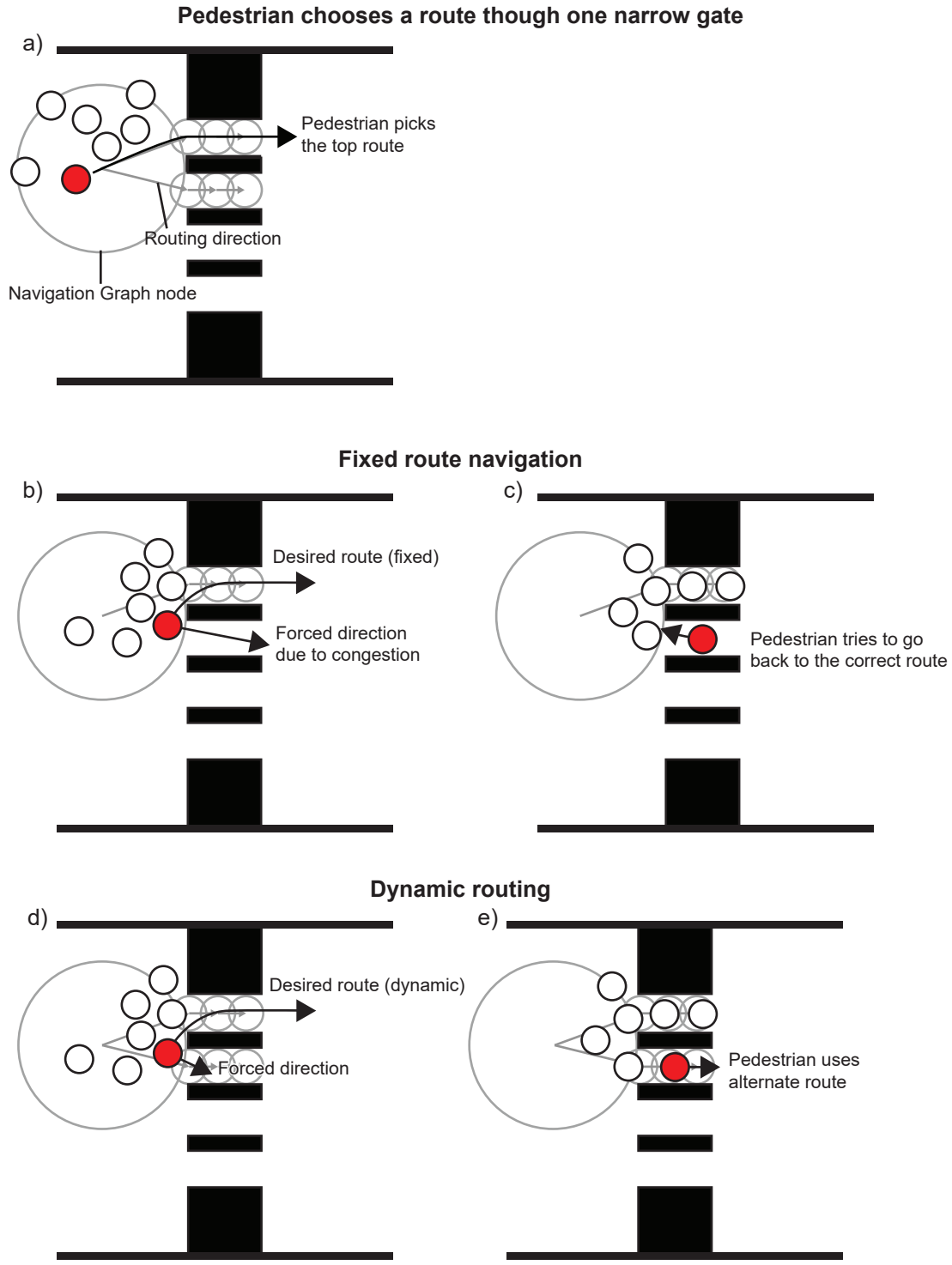

Figure 3.26 Non-walkable areas in black represent the walls and a set of turnstiles. The pedestrian (red circle) has a desired route (a) but is forced to go through another slot due to congestion (b). The pedestrian tries to go back to the pre-specified route causing congestion (c). With dynamic routing (d), the pedestrian is able to switch to using the alternate turnstile without problem (e).

environment is used to test the case where environment complexity remains constant while the environment size changes.

The second is a Corridor which is a bounded rectangle and contains a 2m square obstacle for every 5m of the environment length. These square obstacles are evenly distributed to form a large number of corridors with many possible branching routes. Fig. 3.28 (a & b) shows the environment at 10x10m and 25x25m, respectively. As the number of corridors in the environment is directly proportional to the environment size the complexity of the environment grows exponentially with the environment size. The green capsules in the environment represent portals. In all cases a simple itinerary is created where the pedestrian must get from the portal on the left to the portal on the right.

For the grid-based navigation approach, the resolution used is 4 cells per square metre. A single grid is required for static obstacle avoidance where each cell contains 12 bytes of obstacle information. Additional grids containing navigation information require 8 bytes for each cell.

For the graph-based navigation approach, each Environment Graph node uses 12 bytes of memory to store its own position and radius, and an edge uses 8 bytes of memory to store an index to the node on each side. Each Navigation Graph node costs 16 bytes and each edge costs 12 bytes. A Navigation Graph's memory consumption is variable and depends on the complexity of the itinerary but has a baseline requirement that there is one connection between every traversable node.

As can be seen in Fig. 3.29 and Table 3.2, the memory usage of the grid-based navigation approach increases exponentially with the environment size irrespective of the environment complexity. The memory use for the navigation graph approach meanwhile depends on the complexity of the environment. There is a linear relationship between the number of nodes in the Environment Graph and the total static memory required which includes the nodes and edges in the Environment Graph, the R-tree for searching the Environment Graph and the static obstacle search tree.

The wedge environment example shows how the graph-based approach excels in environments with a mixture of large and small spaces. It is shown to use significantly less memory than the grid-based approach (Fig. 3.29). Even in a scenario such as the corridor scene, where memory requirements for the static data structure are higher than the grid-based approach, the memory required to create an itinerary is consistently less than the grid-based approach (Table 3.3), which makes it more suitable for applications where pedestrians have a wide range of objectives.

Chapter 4 describes how the searchable graph-based navigation approach has been integrated into a prototype pedestrian simulation called Concoursia. Two real-world environments, a shopping mall and a train station, are used to demonstrate this system.

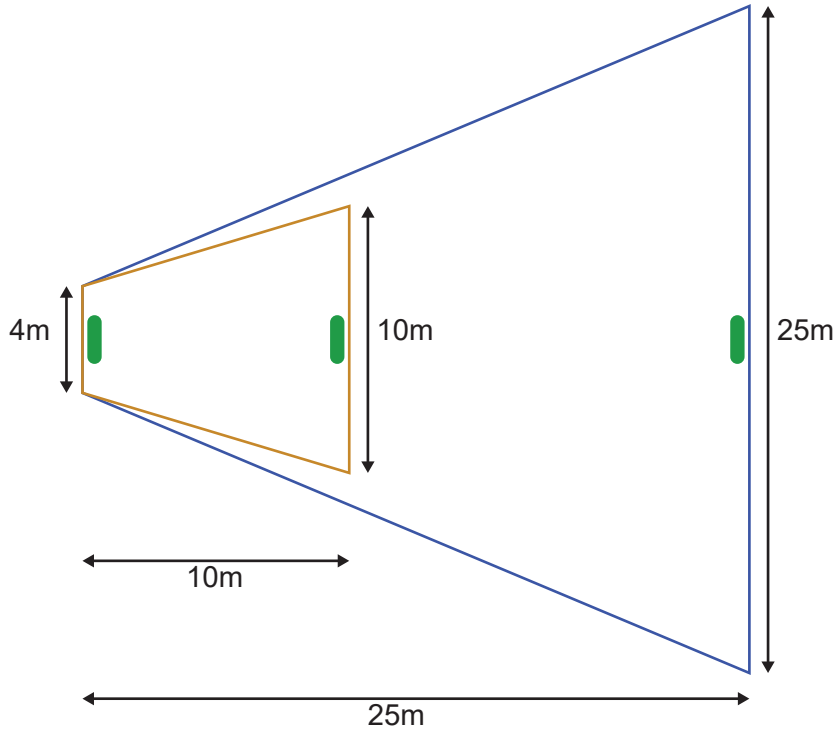

Figure 3.27 A subset of the Wedge environments used to compare memory usage between the two navigation approaches (Tables 3.2 and 3.3). Green capsules represents portals. The environment bounded by brown lines is 10x10m and the environment bounded by blue lines is 25x25m. The left side is fixed at 4m and the right side is the same length as the environment.

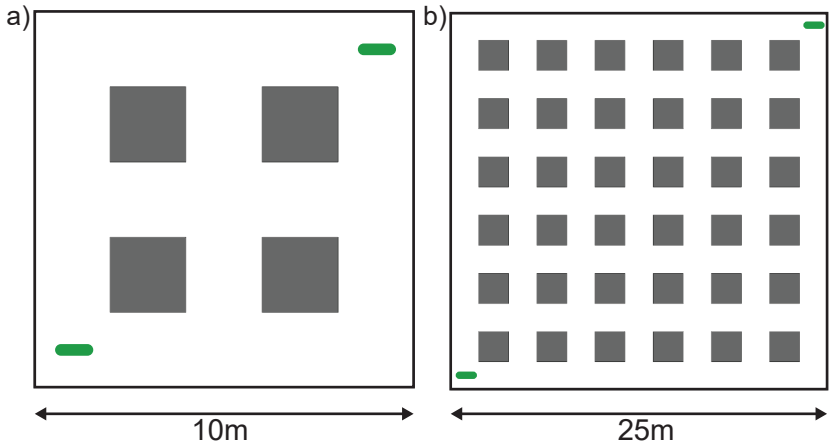

Figure 3.28 A subset of the Corridor environments used to compare memory usage between the two navigation approaches (Tables 3.2 and 3.3). The environment is bounded by black lines and the green capsules represents portals. The environments shown in (a)-(b) are of size 10x10m and 25x25m, respectively. For every 5m of the environment a 2m square obstacles is added and evenly distributed to create a corridor layout so that the complexity of the environment grows exponentially with its size.

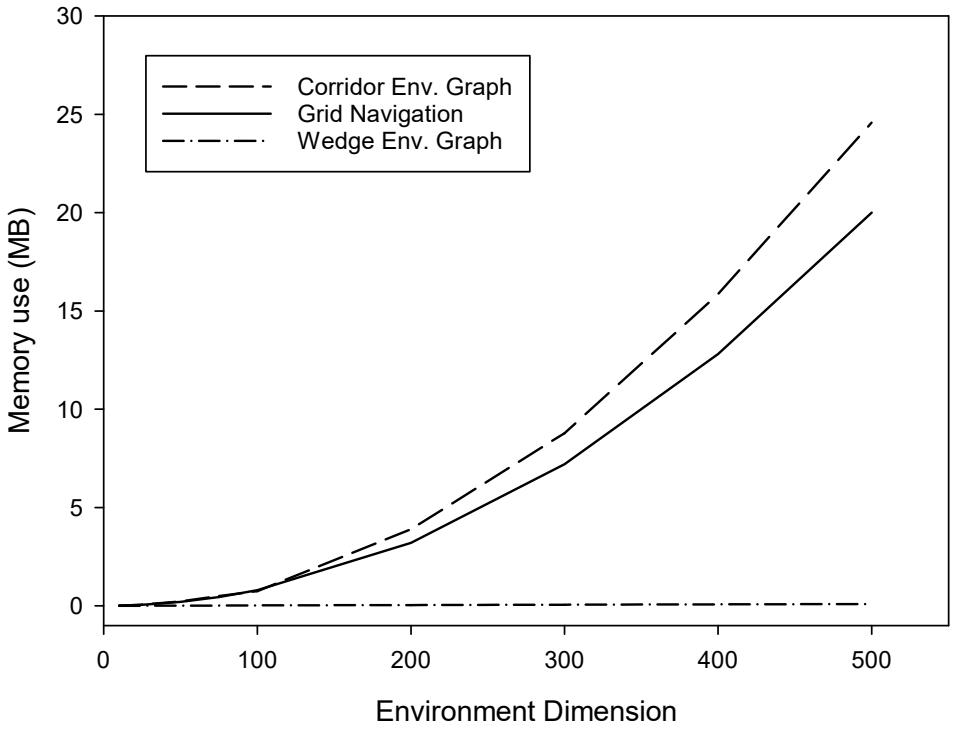

Figure 3.29 Memory usage comparison for the navigation types on a wedge shape environment and corridor environment. For grid-based navigation (solid line), the memory use is the same for both environments and increases exponentially with environment size. For graph-based navigation (dashed lines), memory use is highly dependent on the complexity of the environments rather than the size as can be seen by the disparity for the Corridor environment (dashed line) and Wedge environment (dash-dotted line).

Env. Size (m)	Grid navigation - Wedge & Corridor Env.		
	Num. Cells	Mem. use per CVF (kB)	Mem.use per NVF (kB)
10x10	400	4.8	3.2
25x25	2,500	30	20
50x50	10,000	120	80
75x75	22,500	270	180
100x100	40,000	480	320
200x200	160,000	1,920	1,280
300x300	360,000	4,320	2,880
400x400	640,000	7,680	5,120
500x500	1,000,000	12,000	8,000

Table 3.2 The table shows memory use for various environment sizes for the grid-based navigation system. The same memory consumption applies to both Wedge and Corridor environment.

Env. Size (m)	Wedge Env.				Corridor Env.			
	Num. nodes	Total Base Mem. (kB)	Itin. nodes	Itin. Mem. use (kB)	Num. nodes	Total Static Mem. (kB)	Itin. nodes	Itin. Mem. use (kB)
10x10	22	1.804	2	0.284	37	8.344	13	0.64
25x25	56	3.916	5	0.74	222	54.708	37	3.244
50x50	126	8.212	5	1.58	797	211.672	73	10.72
75x75	184	11.868	5	2.276	1702	474.108	109	22.156
100x100	245	15.656	6	3.024	3280	705.796	151	41.764
200x200	496	31.188	6	6.036	12805	3738.87	301	158.464
300x300	741	46.184	7	8.992	28580	8430.864	451	350.164
400x400	983	60.924	8	11.912	50605	15243.08	601	616.864
500x500	1256	77.368	9	15.204	78880	23617.344	751	958.564

Table 3.3 The table show, for both Wedge and Corridor environment, memory used by the navigation graph system. Base memory includes the Environment Graph, search R-tree and the static obstacle tree and the memory used for a simple Itinerary for getting from one end of the environment to the other.

3.3.2 Navigation Performance

With the agent-based navigation grid approach (Section 3.1), there are two main processes that use computation time: the outputting of navigation messages and access to the navigation message. At every iteration, each navigation agent always outputs a single navigation message and, as they're discrete agents, the message does not need to be sorted or hashed. This means the message output time has a linear relationship with the number of navigation agents. Accessing the navigation information involves translating the pedestrian's current location to a discrete coordinate and accessing a specific message in an array, which is a constant time operation. The message read time is then a linear relationship with the number of pedestrians accessing the navigation grid.

For the searchable graph based approach (Section 3.2), the underlying data structure is static and, as it does not use navigation agents, there are no message output costs, as the next node information is provided by the current node. Thus, in the best case, where a pedestrian can directly follow a route without congestion, this has a constant access time. However, when a pedestrian falls out of the Navigation Node bounds, an R-tree search must be performed to find the nearest node. This has a worst case algorithmic complexity of $O(\log n)$.

We can extrapolate this to a HPC infrastructure. As the searchable navigation graph is passive, there will be no difference in access performance on different nodes. With the agent-based navigation grid, the navigation agents can be split-up with domain decomposition techniques and each node will only need to perform message processing for the navigation agents that falls within in its domain. While domain partitioning is viable for the navigation grid approach, the cost of storing NVF for the subdomain can quickly add up in scenarios where there are a large number of destinations.

3.3.3 Discussion

With the grid navigation being agent-based, there is a disadvantage in that additional overhead is added for every simulation iteration, which scales up with the size of the environment. However, as the environment itself are agents, the advantage is that additional dynamic features such as spread of smoke could potentially be implemented into the existing framework.

With both approaches grid and graph approaches, it is currently costly to rebuild the environment should there be a major change such as multiple routes becoming blocked in the middle, e.g. as a result of part of the building collapsing, as the entirety of the creation of grid or graph structure and path-finding involved needs to be performed again.

The grid-based navigation has an advantage in that it generally offers smoother paths due to the dense data structure and artefacts introduced in the graph-based approach as a result of the spatial sub-division strategy as can also be seen with other graph-based approaches [51, 55, 58]. The smoothness of the path is unlikely to make a major contribution to the flow at the systematic level especially during congestion when pedestrian's movement becomes much more dependent on their neighbours.

In models where the environment size and number of pedestrian objectives are limited, the use of agent-based navigation grids has an advantage in that it provides a robust system for navigation that is simpler to implement and dynamically modify. However, the disadvantage of the grid-based approach is that as we start to consider larger and more complex environments it is not only the size that increases but the number of objectives that must be represented within the environment. This memory requirements limitation makes the case for the use of the graph-based navigation approach much more attractive.

Chapter 4

Concoursia, a prototype pedestrian simulation system

Although the navigation system is an essential part of a pedestrian simulation system, many other pieces are required in order to create a complete platform useful for pedestrian flow analysis. Concoursia is a prototype crowd simulation platform that takes advantage of a Graphics Processing Unit (GPU)'s parallel computational architecture with the use of the FLAME GPU framework (Section 2.6.3). This makes it possible for simulations to run at real-time or faster [71]. The pedestrian navigation system in Concoursia uses the graph-based navigation approach as described in Section 3.2.4. This is implemented for the GPU with CUDA to allow integration with the FLAME GPU framework and allows a pedestrian agent in FLAME GPU to access the navigation system directly from within its agent function. Concoursia also provides a GUI which allows quick creation of scenarios for testing, visualisation and analysis for various complex environments. The system was designed to support a wide variety of use cases. However, through the work with industrial partner for the research, Network Rail, features were added to support common scenarios that occurs in transport hubs such as queuing, waiting areas and ability to specify public transport schedules.

Section 4.1 explains the various components within Concoursia. The building and running of the simulation model are then discussed in Sections 4.3 and 4.4, respectively, including a description of the GUI. Section 4.5 discusses visualisation of real-time simulations. Essential functions such as reporting of metrics from the executed simulation can be found in Section 4.6.

Section 4.7 presents a quantitative evaluation of pedestrian navigation in Concoursia. The evaluation is done by testing the route utilisation of a complex branching environment. Section 4.8 presents two case studies based on real-world environments are presented which highlight typical use-cases for the platform, as well as providing

benchmarks under realistic conditions. The first is a simple environment based on a single floor of a shopping mall. The large floor space offers a fairer comparison of memory use for the grid and graph-based navigation approaches previously covered in Chapter 3. The second is a more complex example of a busy train station in London spanning over three floors that utilises every feature implemented in Concoursia. Two scenarios are used for both examples, one for normal operations and one for evacuation. Finally, Section 4.9 discusses the Concoursia system.

4.1 System Overview

Concoursia is divided into five major components as can be seen in Fig. 4.1. The software was designed with a Client and Server view in mind. The Client handles the editing, creation and analysis of the environment and scenario. The Server application performs the building of the scenario, as well as running the simulation without direct user interaction or visualisation.

With the intention for later adapting the software to a Client and Server application in mind, the software was designed by following the Model View Controller (MVC) pattern to create a loose coupling between the logic of the program and its visual representation. The Scenario Model, Simulation Builder and Simulator can be thought of as models. The functionality of these model components are provided as API functions usable by the views and controllers. These model components completely encapsulate the functionalities of creating and running a simulation and can be used without visual representation, a concept further explored in Chapter 5. The GUI component is the sole controller and also provides a view for standard GUI items, e.g. buttons, menu, dialogue boxes, etc. The Visualisation component is purely a view for visually representing the physical items in the scenario (e.g. walls, floor, etc.) and the resulting simulation, i.e. the rendering of pedestrians. The model essentially serve as the Server part and the view and controller as the Client part. This pattern also makes it easier for multiple developers to work on separate parts of the application at the same time after having designed the interface between the components.

The Scenario Model component provides an API and data structure for manipulating, storing and accessing data necessary to build and simulate a scenario (Environment Data and Navigation Data). The component's function is akin to a database which stores and provide data to all other components. Only the serialisation of the Scenario Model component is necessary to store the working state of the application. After the scenario creation process is finished, the Environment Data is passed over to the Simulation Builder (Section 4.3) which generates GPU-optimised Navigation Data.

After the build process, the Scenario Model contains all data that is required to run a simulation. As the building of the Navigation Data is an independent process from management of Environment Data, it is separated into the Simulation Builder component. Having generated the Navigation data, it is now possible to run the simulation using the Simulator component. The Simulator component (Section 4.4) is a wrapper over FLAME GPU that handles necessary tasks on the CPU such as scheduling of events and recording of metrics.

The GUI component uses Qt [275] to present user interface components for user interaction as well as a viewport for 3D OpenGL rendering. The rendering within the viewport is done by the Visualisation component that renders all physical items related to the scenario using OpenGL. The rendering is done by using data from the Scenario Model and the Simulator. The GUI and Visulisation are conceptualised as two separate components in order to draw a distinction between interaction between standard Qt user interface components and 3D rendered scenario items.

The GUI's Environment Designer (Section 4.2.1), Network Editor (Section 4.2.2) and Itinerary & Schedules editor (Section 4.2.3) are used together in conjunction with the Scenario Model to produce a set of Environment Data to be used by the Simulation Builder. The Simulator on the other hand is controlled by the GUI's Simulation mode (Section 4.4). The following sections will explain the various components of Concoursia in order by using the creation of a simple two-platform bus station environment as an example (Fig. 4.2).

4.2 Creation of A Scenario

As previously mentioned, the Scenario Data component (Fig. 4.1) is a set of APIs and data structures used for manipulating, storing and serializing scenario data. Within it, the Environment Data contains information about layout of the environment, objective locations and how they are connected, as well as the Itinerary and Scheduled events that are due to happen.

4.2.1 Environment Objects & Environment Designer Mode

Physical items within the environment are called Environment Objects and can either be represented as rectangles, circles, capsules or polylines. The Environment Objects consists of Walkable areas, Obstacles, Portals, Waypoints, Queues and Waiting areas. All Environment Objects are stored in the Environment Objects List.

For the simple two-platform bus station example, the placement of all Environment Objects are shown in Fig. 4.2. Walkable areas bounded with black lines mark an area

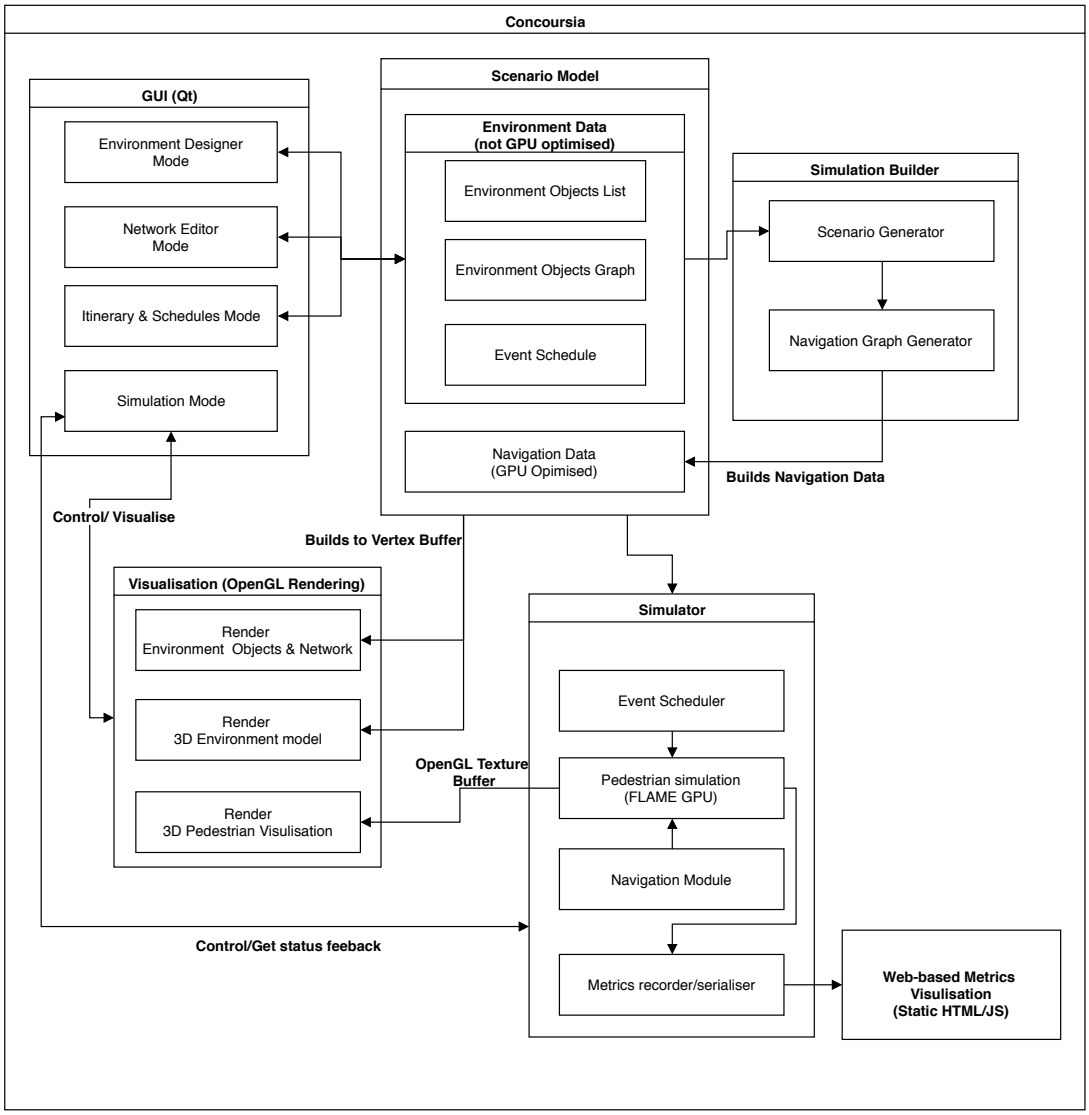

Figure 4.1 A simplified data flow and interaction diagram of Concoursia's different components.

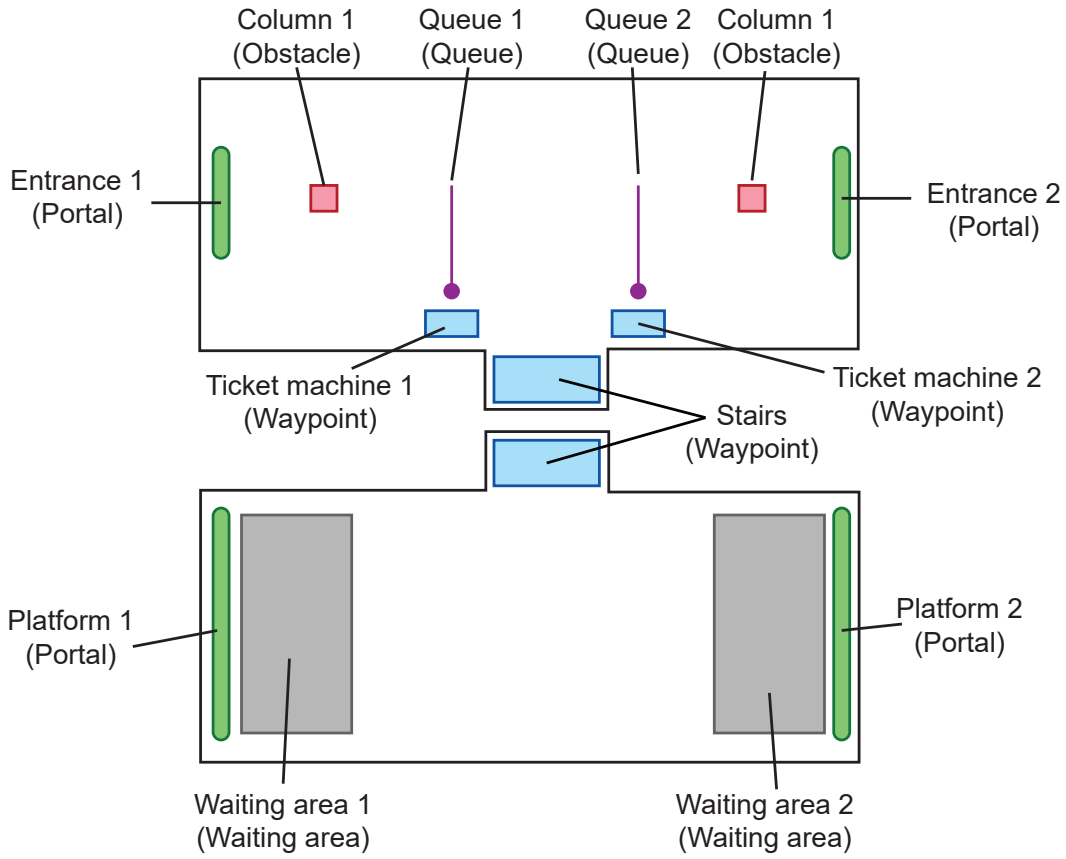

Figure 4.2 A simple bus station environment with two floors and a set of stairs connecting them. The environment is bounded by the walkable area within the black lines. It shows the Static Obstacles (red areas), Portals (green areas), Waypoints (blue areas), Waiting area (grey areas) and Queues (purple dots and lines).

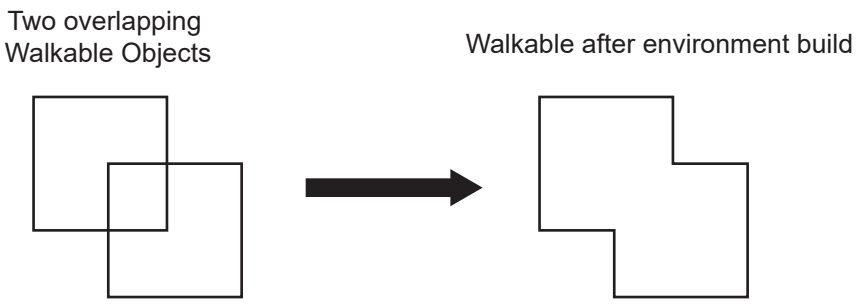

Figure 4.3 Overlapping Walkable Objects are merged into a single walkable area during the building of the simulation.

of the environment walkable for the pedestrians. Its boundaries are then considered as impassable walls. When walkable areas overlap, they merge to form a single shape making it possible to form more complex walkable areas from simpler shapes (Fig. 4.3). Obstacle objects are static impassable obstacles which can be placed inside Walkable areas. Portals represents entrances and exits to the simulation such as doorways of buildings, train doors, etc. Waypoints provide the ability to give pedestrians specific routes within the environment and for connecting multiple levels together in the case of stairways, lifts or escalators. Queues are linked to Portal or Waypoints and provide an indication of how a line should be formed when waiting to reach an objective. Waiting areas can be linked to Portals or Waypoints in order to clearly mark the areas that pedestrians can wait while a Portal or Waypoint is closed, e.g. the platform area when waiting to board a train. Pedestrians in the Waiting area will attempt to evenly distribute themselves within the area.

The creation and modification of these objects are done through the GUI in the Environment Designer Mode. Each time the software starts, an empty Scene is created to contain all objects and events related to the simulation. The Environment Designer Mode allows users to deal with the placement of objects in the physical environment. Pre-existing floor plans can be imported from Adobe's dxf file format and converted to either Walkable or Obstacle objects. Users can then create new Walkable and Obstacle objects or modify existing ones. As shapes of Portals, Waypoints, Queues and Waiting areas are restricted to rectangle, circles or capsules, users have to manually place these objects within the scene within the program. Images can be imported to serve as an underlay to guide the creation of the environment. Objects can be grouped together to help avoid repetitiveness of the object connection process, described in Section 4.2.2, when using the user interface.

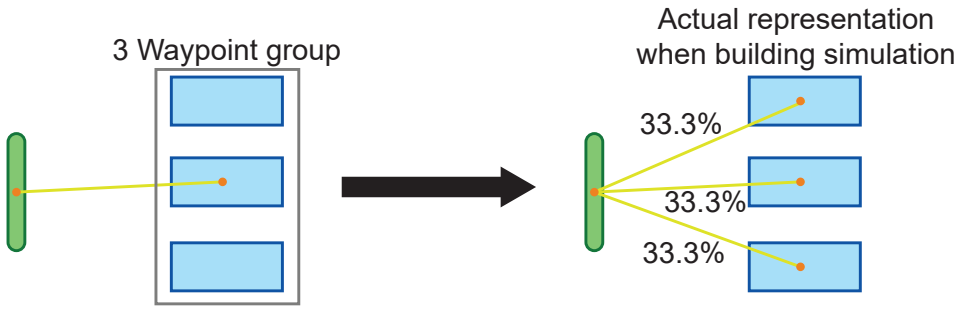

Figure 4.4 Environment objects of the same type can be grouped in the Network Editor Mode. To the simulation they are seen as branches with equal probability that a pedestrian can choose from.

4.2.2 Environment Object Graph & Network Editor Mode

The Environment Objects Graph can be created within the GUI using the Network Editor Mode (Fig. 4.5). The graph contains connections between the networkable Environment objects which are the Portals and Waypoints. A connection between objects can be specified as bi-directional or uni-directional to represent different types of flow control strategies. Groups of either Portals or Waypoints can also be made. Grouped objects behave similarly to singular objects in the connection process but the underlying network also makes a connection to each object within the group. During the building of the simulation, this results in branching paths of equal probability to each item in the group (Fig. 4.4). As Concoursia operates in a 2D environment, multiple floors are laid out on the same plane next to each other and Waypoints must be placed at the transfer locations between the floors and then connected together to allow pedestrians to navigate between the floors.

Queues and Waiting areas are not included in the graph and are contained as properties within Portal and Waypoint objects. The graph allows the creation of sequences of objectives, e.g. pedestrians must go to a ticket machine first before heading to a platform. Fig. 4.5 shows the connections between the Environment Objects in the bus station example, where yellow lines indicate a connection between objects on the same floor and the dotted grey line is a connection between objects on different floors.

4.2.3 Event Schedule & Itinerary and Schedules Mode

The Event Schedule stores a timetable of events such as when a Portal is emitting and when a Portal or Waypoint is closed and pedestrians must initiate queueing or waiting behaviour. This can be used for example to create schedules for arrival and departures

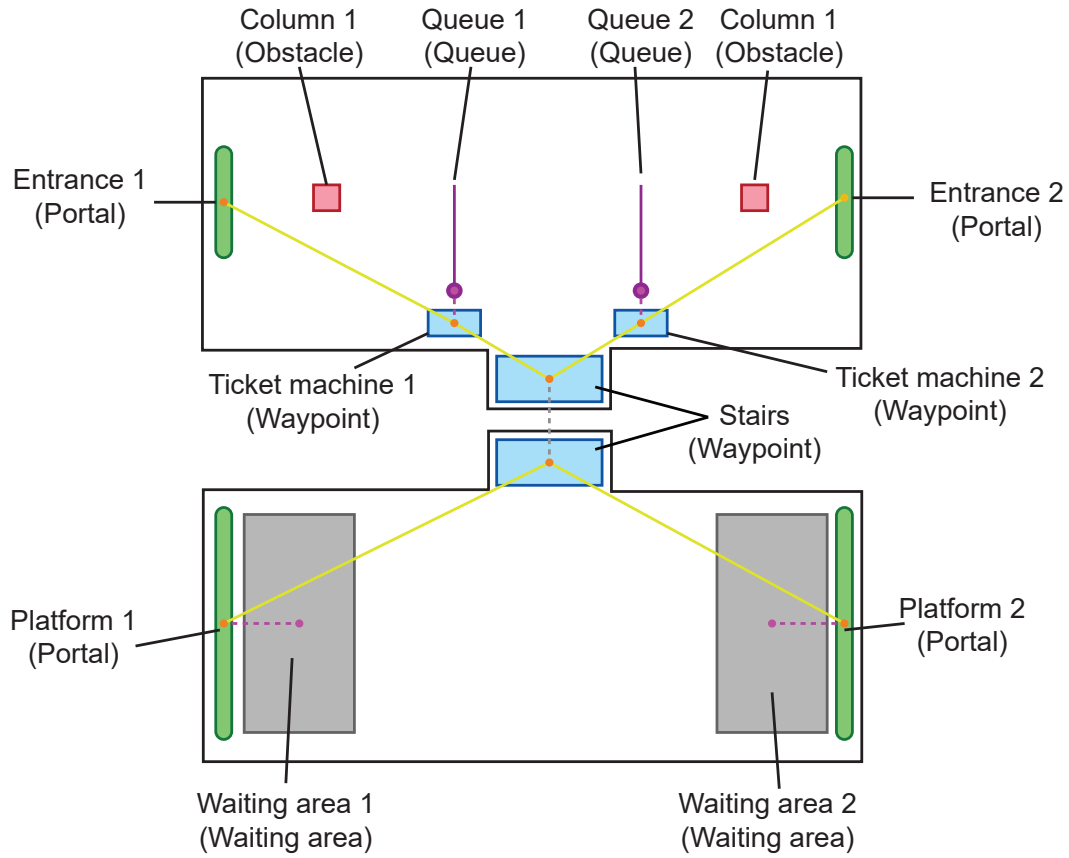

Figure 4.5 The connections created for the simple bus station environment in Fig. 4.2. Each orange dot represents a networkable Environment object. They are connected with the yellow lines on the same floor and a dotted grey line on different floors representing stairs. Waiting areas and queues are special cases where association with a Portal or Waypoint is set through object properties rather than in the network. They are shown connected to the Portal or Waypoint object with dotted purple lines.

of trains and buses. The Event Schedule can be created from the GUI's Itinerary and Schedules Mode (Fig. 4.6).

The Event Schedule consists of two main parts, the Itinerary Graph and the Schedule. The Itinerary Graph (Section 3.2.2) encapsulates a user's navigational goals at a higher level. It is composed of Portals, Services and Activities. Timetabled public transport such as buses or trains normally have associated platforms for arrival and departure. These timetabled services are often cyclical in that there are multiple buses going to a particular destination on the same day. Services allows the representation of a timetabled transport service going to a set destination and affect the route taken through the environment. For example, pedestrians will go and wait at the platform that has the bus they're waiting to catch. An Activity represents an objective that must be completed when following the Itinerary, e.g. getting a ticket. An activity is bound to a Waypoint Object. The software makes the assumption that whatever the pedestrian's behaviour within the environment the pedestrian always tries to exit from the environment at the end so, defining a single Itinerary is a matter of choosing a starting and ending portal/service and compulsory activities. For the example shown in Fig. 4.6, an Itinerary Graph as shown in Fig. 4.7 is generated. The Schedule is then a list of events and determines the timing of when pedestrians are emitted, e.g. when pedestrians come into the environment by foot or when a service arrives at a platform.

4.3 Building the Simulation

The Simulation Builder prepares the Environment Data for simulation. It creates a GPU optimised navigation data structure and a Scheduler for handling events during a simulation run.

Navigation of the pedestrians in Concoursia uses a GPU-optimised navigation graph approach, as described in Section 3.2. The Navigation Builder in Concoursia reduces the environment into an Environment Graph which contains a series of connected walkable areas. From the Itineraries specified in the Event Schedule, specific routes between connected objects in the Environment Object Graph are found by finding the Environment Graph node nearest to the Environment Objects. Routes are found between two connected objects using A* and the resulting information stored in the Environment Object Graph's edges. For every Itinerary, at least a single Navigation Graph is created, but more can be used to represent a complex Itinerary involving multiple objectives or requiring a pedestrian to walk back on itself, e.g. Fig. 3.19 when a pedestrian has to walk to the shop then double back to the exit. The Navigation Data creates a Structure of Arrays (SoA) data structure (Fig. 2.2) to store all generated

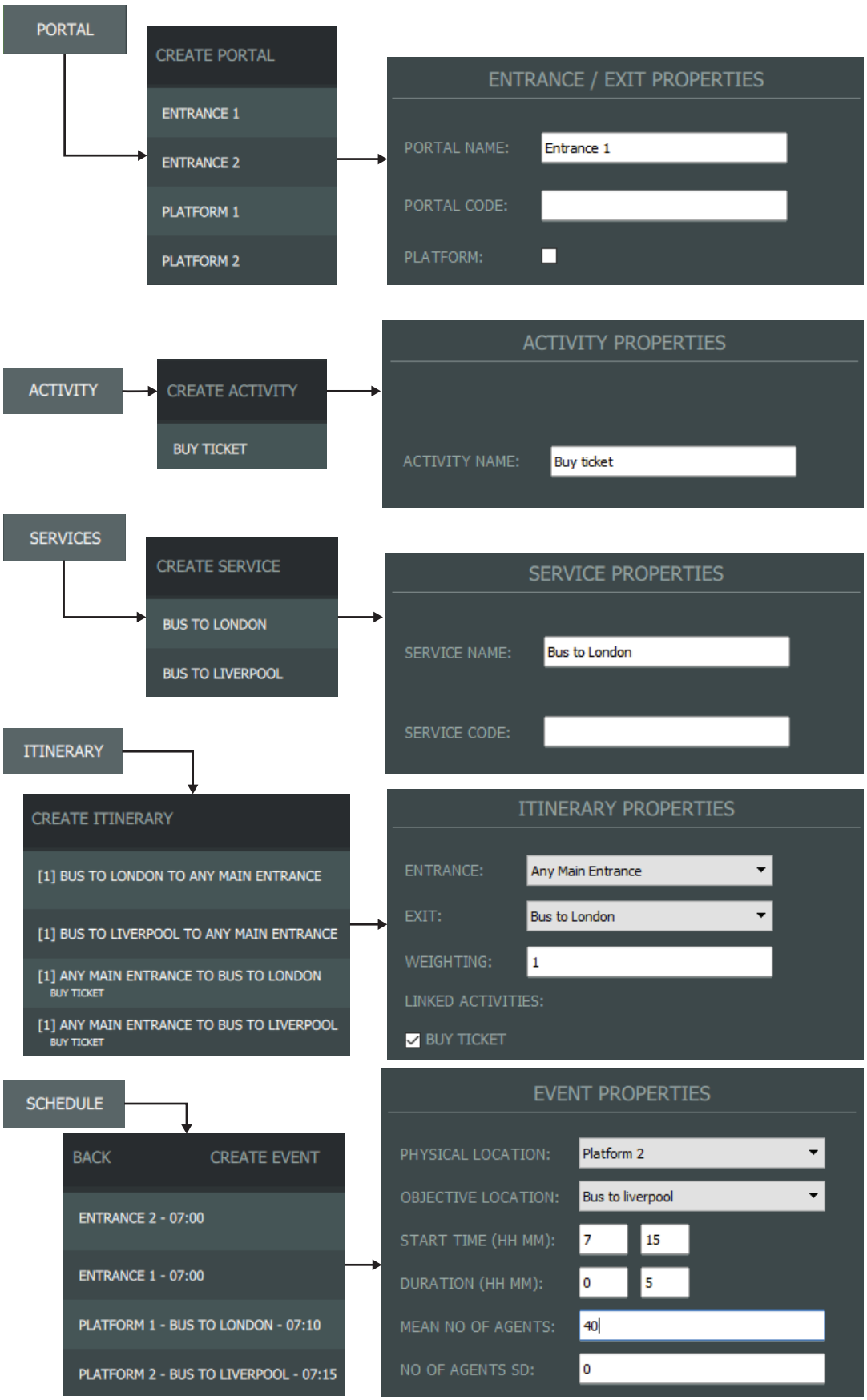

Figure 4.6 The Portal, Activity, Services, Itinerary and Schedule pages in the Itinerary and Schedules Mode.

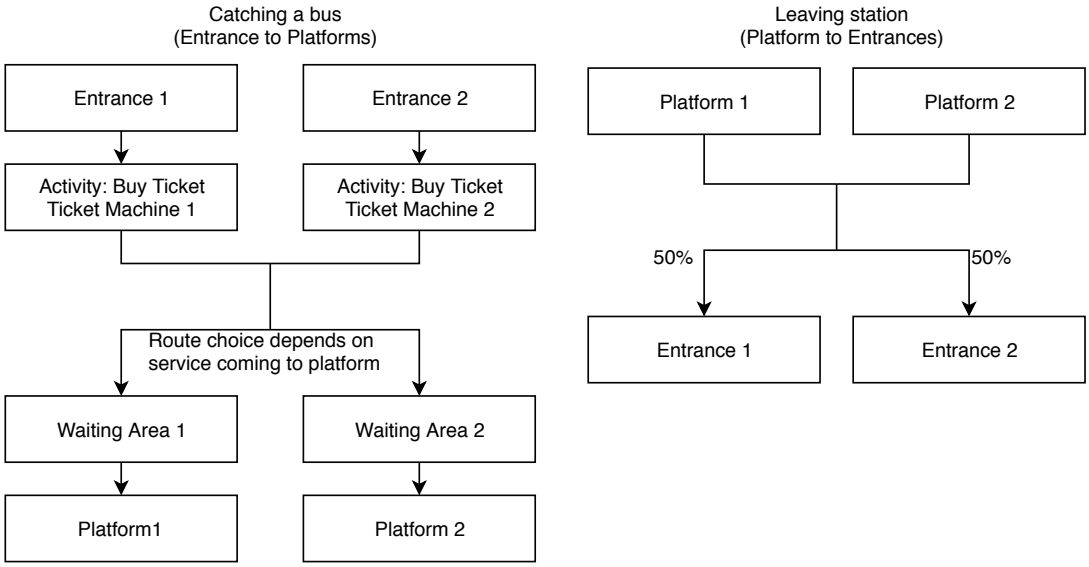

Figure 4.7 The Itinerary Graph generated with the environment in Fig. 4.5 for the example in Fig. 4.6.

navigation information required by the agent during simulation and is uploaded to the GPU once the simulation starts. The Navigation Data structure is shown in Fig. 3.20 laid out as Array of Structures (AoS) to make it easier to visualise but is laid out in memory as SoA (Fig. 3.21) in actual use on the GPU. The use of the Navigation Data structure is explained in Section 4.4.2. Portals and Waypoints store position indices to their counterparts in the Navigation Data allowing properties such as opening and closing or branching probability to be changed during simulation run. An example is the arrival and departure of Services causing a portal to open and close (Listing 4.1).

```
//During a simulation iteration
services_event = scheduler.check_services_event()
foreach service_event in services_event:
    portal = service_event.get_associated_portal()

    if service_event.arrived():
        portal.open()
    else if service_event.departed():
        portal.close()

    navigationData.target_array.is_open[portal.
    target_array_index] = portal.is_open()
    navigationData.routes_array.is_open[portal.
    route_array_index] = portal.is_open()
    navigationData.synchronise_with_gpu()
```

Listing 4.1 Pseudocode for a part of the Schdeduler's process every simulation iteration. A portal object updates its status and synchronises data with the GPU.

A scheduler is also created which tracks and controls the simulation time and dispatches events on the CPU. The events include emission of pedestrians, opening and closing of Portals and Waypoints, arrival and departure of services and recording of metrics.

4.4 Simulator

Concoursia's Simulator is the part that performs the running of the simulation. It is composed of the Pedestrian Simulation engine, Event Scheduler and the Metrics recorder.

The Pedestrian Simulation engine in Concoursia is a custom version of FLAME GPU that has been modified to run embedded in an application. It allows Concoursia to take advantage of the GPU's processing power without re-implementation of essential functions such as GPU optimised spatially sorted message passing. The agent model used in Concoursia is explained Section 4.4.1 and the Navigation Module which the pedestrian agents use for navigation is explained in Section 4.4.2.

The Event scheduler created by the simulation builder runs on the CPU and any event dispatched, e.g. emission of pedestrians, has a wrapper that allows the upload of this information to the GPU where the pedestrian model is computed. At set intervals

of simulation time, it tells the Metrics recorder to process the model creating metrics from the simulation, download the information from the GPU and save the data to disk. Further details on the metrics recorded can be found in Section 4.6.

4.4.1 The Concoursia Agent Model

Concoursia's agent model consists of two different types of agents, pedestrian agents and queue agents. Unlike navigation behaviour which is essentially static data access with no inter-agent communication, the queueing behaviour was introduced as queue agents due to the need for active management of queue status and the need for a single queue to communicate with multiple pedestrian agents and resolve queuing order conflict. Each agent's function and the messages used for communication are shown sequentially in Fig. 4.8.

The *navigate* function deals with accessing the navigation graph structure (Section 3.2.4) at runtime, and assessing the pedestrian's current objectives and headings. The Navigation Module provides a single function `nm_navigate()` where pedestrians can pass in the details of their current navigation objectives and get a vector directing them to their destination. Further explanation of the Navigation Module and the `nm_navigate()` function can be found in Section 4.4.2.

The Queueing behaviour was deemed essential for simulation of busy public transport hubs as the behaviour can be observed in many cases such as getting a ticket, waiting to go through turnstiles or waiting to board a bus/train. The problem with queueing is that it is essentially a sequential operation and so a queueing agent has to be made to arbitrate the order in which pedestrians get to queue. Each queue in the environment contains a single queue agent. The pedestrians looking to join the same queue must first request a queue time from a queue agent which in turn gives a unique number to each requester. The pedestrians use this queue id as a ticket and order themselves by queueing behind agents with a lower queue number. Pedestrians that are queueing then continually make requests to the queue agent to check whether they are at the head of the queue and, if they are, receive a queue "ticket" allowing them to go and use the thing they were queueing for. After the pedestrian is done, it sends the ticket back to the queue agent notifying that a space is now available for the next agent.

For local collision avoidance against static obstacles and other pedestrians the Optimal Reciprocal Collision Avoidance (ORCA) model is used, based on the paper by Snape et al. [130]. By using the ORCA approach, simulation time steps can be kept constant. This makes the simulation faster than solving a force-based equation that restricts the simulation time step by a maximum force that can be processed within a

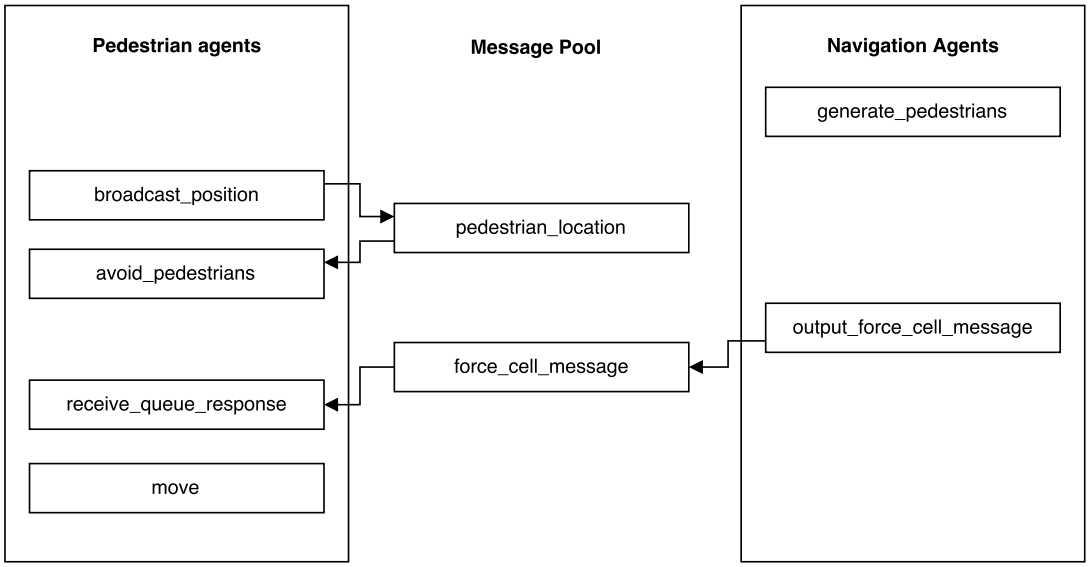

Figure 4.8 Concoursia's agent model showing the Pedestrian and Queue agent functions and the messages used.

time slice. The static environment is stored in a searchable KD tree structure to be accessed during the `local_collision_avoidance` function. For avoiding other pedestrians, each pedestrian first broadcasts its own location into a spatially partitioned message pool (`avoid_pedestrians`). In the `local_collision_avoidance` function the pedestrian accesses location messages within their own spatial bin. Due to the inefficiency in dynamic memory allocation on the GPU device, instead of using a dynamic list to store all obstacles within the perception range, the list is restricted to a total of 30 obstacles (15 dynamic and 15 static obstacles). When adding an obstacle to a full list, the furthest obstacle from the pedestrian is rejected.

4.4.2 Navigation Module

In order to allow pedestrians to navigate complex environments, a system has to be implemented to provide global navigation. The Navigation Module is implemented as an extension in the FLAME GPU framework and uses the searchable navigation graph approach detailed in Section 3.2. It provides an API that pedestrian agents can use in order to access navigation information from the kernel. In addition, it provides support for static obstacle avoidance, waiting and queueing. Specific details for the implementation of the module can be found in Section 3.2.6.

The navigation data used in the module is created by the Navigation Graph Generator and data management is entirely separate from FLAME GPU. When starting a simulation the Navigation Data is uploaded by this module to the GPU.

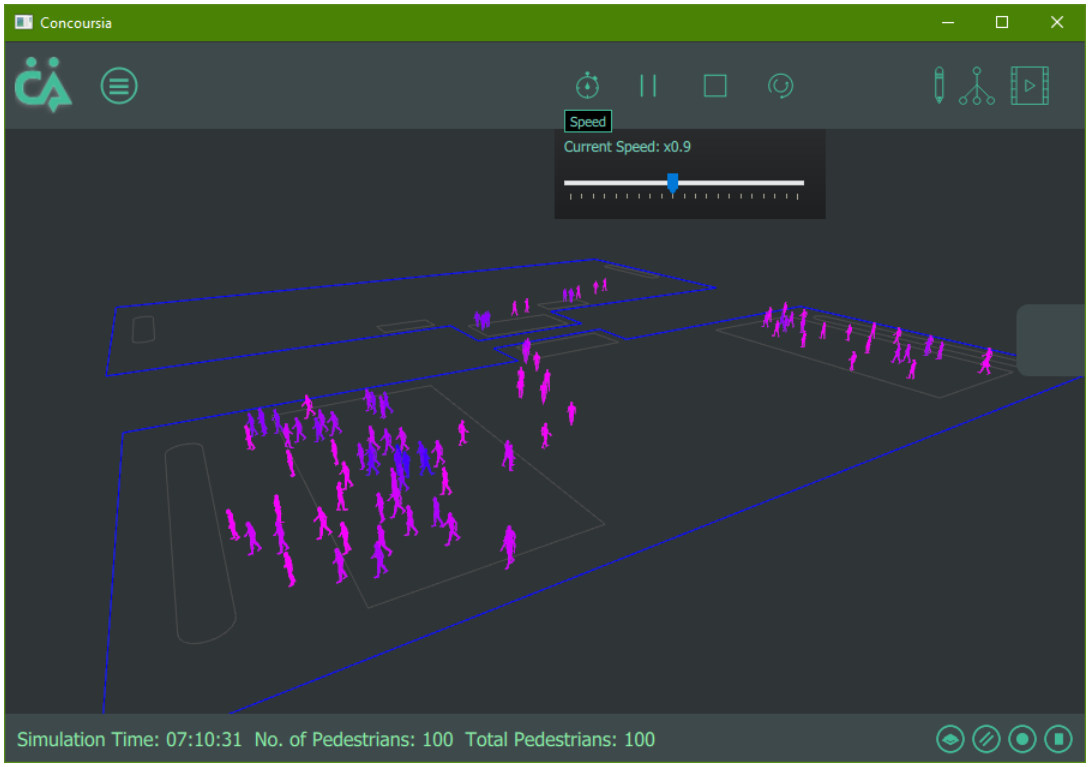

Figure 4.9 Simulation and 3D visualisation of the bus station model. The 'Current speed' bar at the top allows for adjustment of simulation speed either to gain increased simulation performance (speeding up) or easier visual inspection (slowing down).

4.4.3 Simulation Mode

The GUI's Simulation mode (Fig. 4.9) is used to build, control and visualise the running simulation. After a scenario has been created and the simulation is started by the user, the Scenario Data is passed to the Simulation builder. The resulting data is in turn sent to the Simulator to start the simulation process.

Once the simulation starts, by default the simulation time is locked to real-time. A slider within the interface allows a change in the time parameter allowing the simulation to be sped up, paused or slowed down. When the user chooses to speed up the simulation, the frame-rate is reduced to maximise simulation capacity as simulation and visualisation both use the GPU for computation. Locking the simulation speed to real-time, and pausing or slowing down the simulation allows for easier inspection of model behaviour.

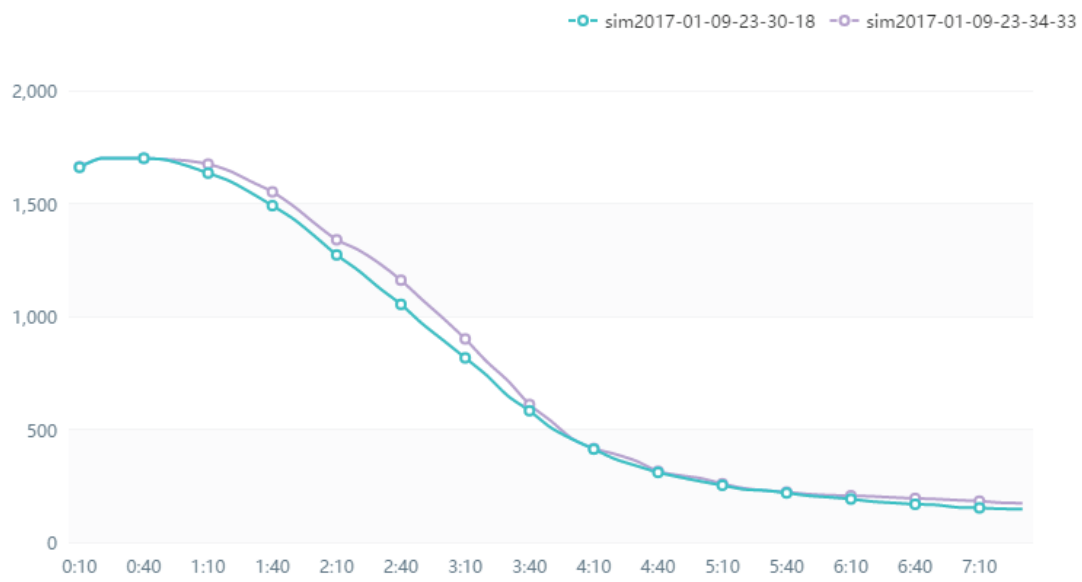

Figure 4.10 Screenshot of the pedestrian count metric showing number of pedestrians in the simulation over time for all simulation instances. X axis is time (hours and minutes) and Y axis is the pedestrian count.

4.5 Visualisation

Concoursia uses OpenGL rendering across all of the main GUI interfaces for visualising the environment and pedestrians during a simulation. The environment renderer create a buffer of environment geometry on the GPU from the Environment Data. For pedestrian rendering, a CUDA kernel selectively copyies pedestrian information from the FLAME GPU simulation, e.g. positions and headings, writing directly to an OpenGL Texture Buffer Object. Two human geometry poses are used to animate the pedestrian. The first geometry has the pedestrian with right foot forward and left foot back while the other is the opposite. The walk animation is simply a cycle of linear interpolation between the two shapes. A vertex shader uses the texture data to interpolate pedestrian geometry to create a walk animation. The pedestrian geometry is then rotated to the correct heading and translated into the correct position in the environment.

4.6 Metrics

Concoursia collects three metrics: pedestrian count, journey time and Level of Service (LoS) [276]. The metrics are automatically recorded every 5 seconds of simulation time interval during the simulation run.

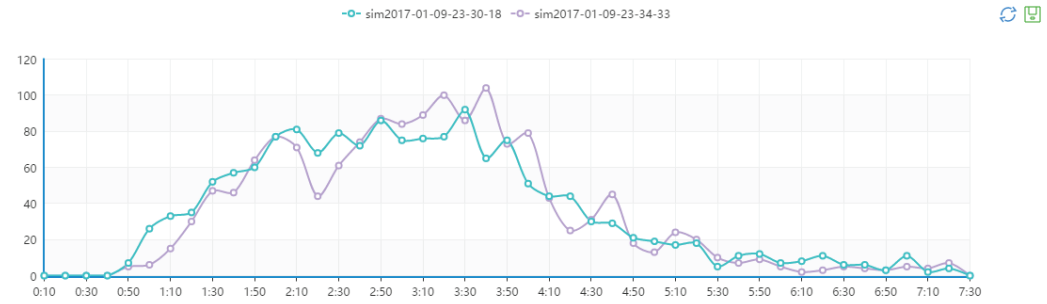

Figure 4.11 The evacuation graph shows the number of people who have exited the environment during a certain time period, which is used for analysing the rate of exit. X axis is time (hours and minutes) and Y axis is the number of pedestrians that exited during the period.

LoS	Walking (m^2/ped)	Standing/Queueing (m^2/ped)
A	≥ 3.24	≥ 1.2
B	2.32 to 3.24	0.93 to 1.2
C	1.39 to 2.32	0.65 to 0.93
D	0.93 to 1.39	0.28 to 0.65
E	0.46 to 0.93	0.19 to 0.28
F	≤ 0.46	≤ 0.19

Table 4.1 The LoS values used for measuring crowdedness of the environment.

The pedestrian count consists of the number of pedestrians in the environment over time (Fig. 4.10). It is used as a general indicator of how busy the environment is and for how rapidly pedestrians can exit the environment during an evacuation scenario.

The pedestrian journey starts when it enters the environment and ends when it exits (Fig. 4.11). This metric is useful in a wide range of cases such as checking the time that a pedestrian took to evacuate a station or for getting from an entrance to boarding a train. Each pedestrian has a variable that records the simulation time that it enters the simulation. At every iteration, a check is made to see if a pedestrian has exited the environment and the current simulation time minus the time that they entered the simulation is recorded.

The LoS (Figs. 4.19, 4.21, 4.25 and 4.27) is a standard metric used to measure the quality level of traffic for both vehicles and pedestrians [276]. It is measured in levels as shown in Table 4.1. The metric starts at level A indicating the lowest measured density, as area of clearance around each pedestrian, and ends at level F with the highest density. The metric takes into account that activities like walking require more room around the pedestrian than queueing. So, the lowest LoS when walking, level F, which requires $0.46m^2$ per pedestrian, is roughly the same as LoS level D when queueing or standing.

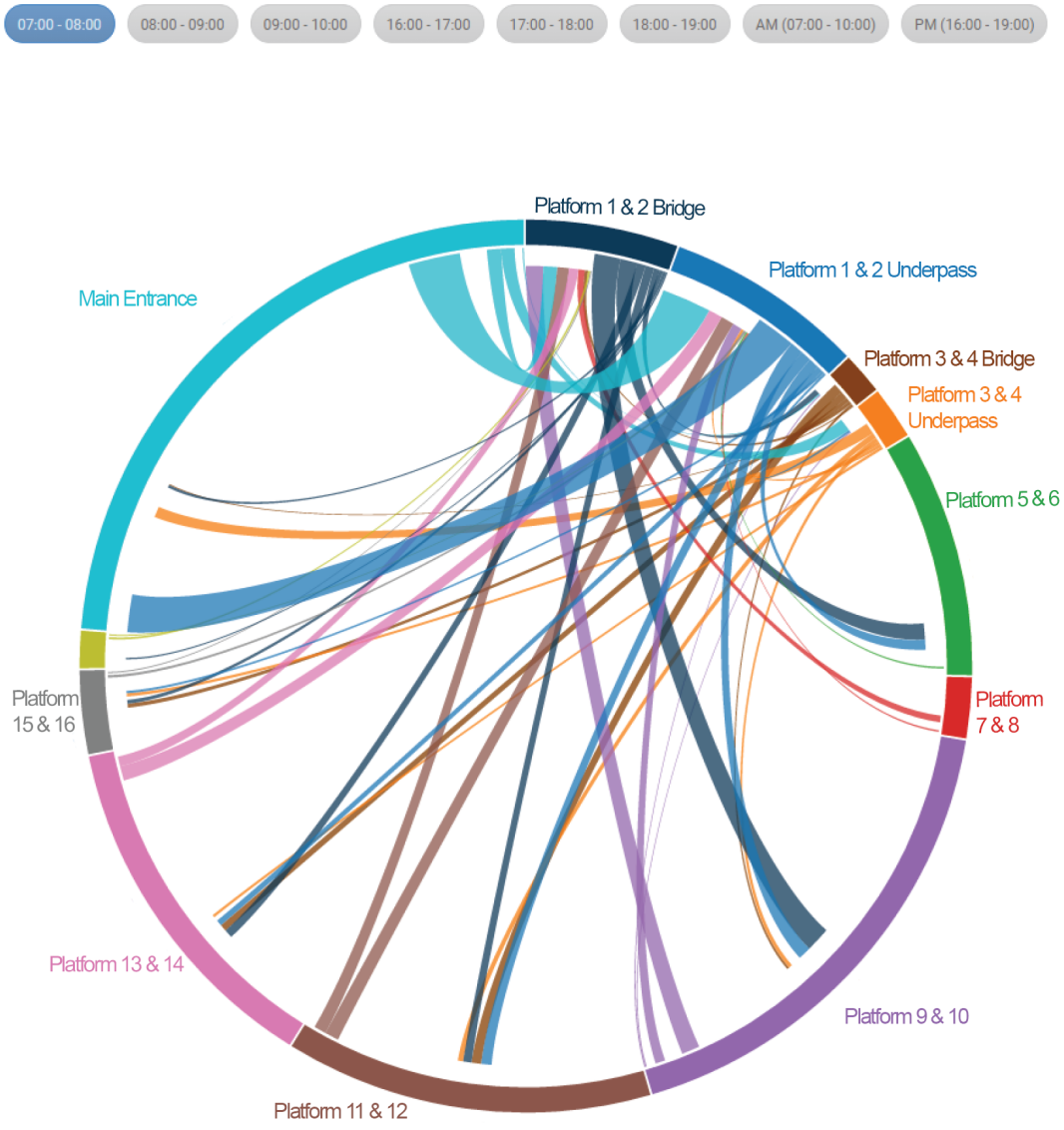

Figure 4.12 Origin and Destination matrix shown as a circular migration flow plot.

Visualisation of the metrics after the simulation has finished running is done through a web-based interface. This was done as the first phase of transitioning Concoursia to a web-based tool. The main dashboard shows a graph that gives the number of pedestrians in the simulation over time (Fig. 4.10). The Migration page shows pedestrians' origin and destination using a circular migration flow plot (Fig. 4.12). The Heat maps page display a cumulative LoS map that shows the maximum LoS experienced in the environment over the course of the simulation (Figs. 4.19, 4.21, 4.25 and 4.27). The Evacuation page shows a graph of the number of pedestrians that has exited over the time period, results from multiple simulation runs can be plotted on the same graph to show a comparison (Fig. 4.11).

4.7 Quantitative Evaluation of Agent Navigation

A tree-like environment (Fig. 4.13) is used to verify that pedestrian agents are able to navigate through a complex branching environment, correctly following the predefined branching probabilities. This environment starts from a single corridor on the left with a height of 150m, which, after 50m, branches into two corridors, each of them half its width. Multiple environments are tested where the branching happens from once to four times following the same branching rule. The pedestrians are emitted from a single Portal on the 'trunk' (i.e. the left side of the environment) and have to navigate to the exit Portal at the 'leaves' (i.e. the right side of the environment). Each exit Portal is numbered sequentially starting from the top. At each junction, there is a 50% probability of a pedestrian either choosing the top or the bottom branch and so every exit Portal should statistically obtain a similar number of pedestrians.

For each simulation run, 1000 pedestrians are emitted continuously and the simulation ends when all 1000 pedestrians have exited the environment upon reaching an exit Portal. The simulation is run 100 times for each environment. The cumulative numbers of pedestrians exiting for each environment over 100 simulation runs are shown in Table 4.2. The percentages of chosen exit Portals are also included in Table 4.2. This shows that pedestrians spread out to use all exits evenly with a maximum of 0.52% deviation due to the stochastic nature of pedestrian emission and the random probability when choosing branching routes.

Figs. 4.14 to 4.17 show the distributions of exit time and distance travelled for the environment with one to four branches, respectively. The distributions are shown for each individual exit portal. The mean speed of the pedestrian is in the range of 1.495 to $1.505 ms^{-1}$ for all environments with standard deviation in the range of 0.187 to $0.189 ms^{-1}$ which is in line with the desired speed of $1.5 ms^{-1}$ with standard

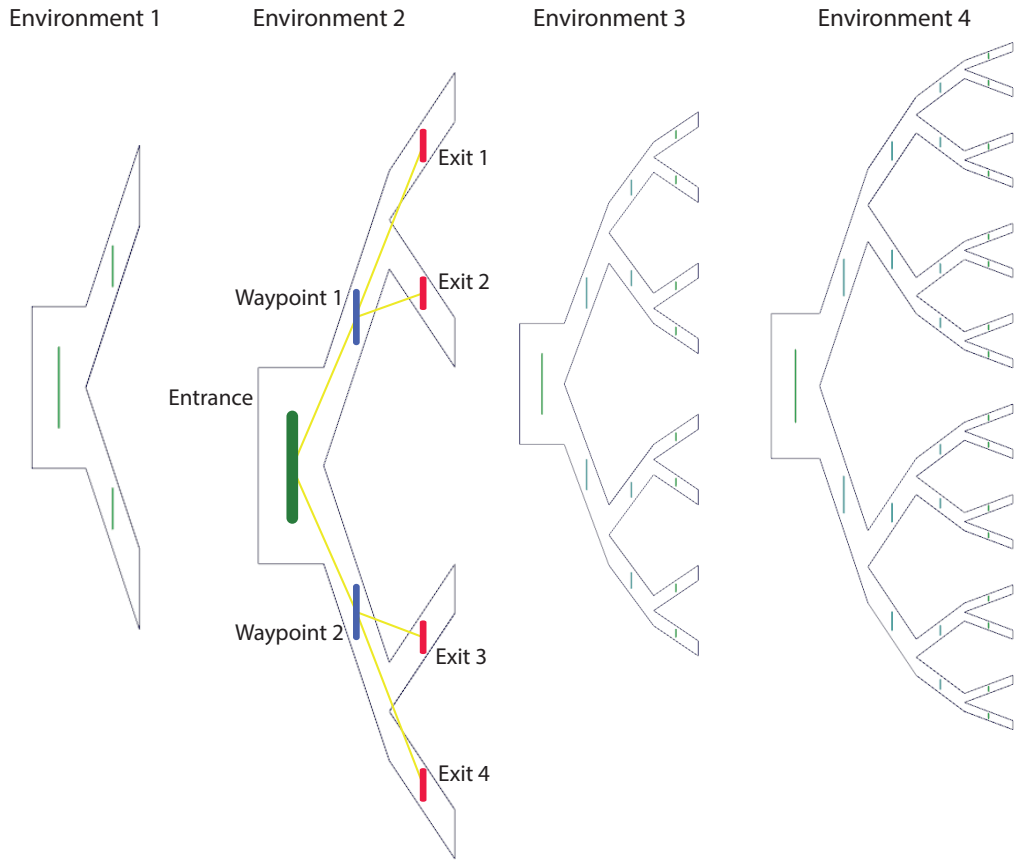

Figure 4.13 The four Tree environments used for testing branching probability during pedestrian navigation. Environment 2 has been enlarged and annotated to show the location of the entrance (green), waypoints (blue) and exits (red). Yellow lines represent branching connections in the itinerary.

deviation of $0.2ms^{-1}$. The relatively uniform distribution of both exit time, and distance travelled, as well as the low variance in mean speed of pedestrians, shows that minimal congestion occurred. This is in line with the uniform distribution for low congestion evacuation from an environment as observed by Viswanathan et al. [277] for the RVO model. Although their paper uses zonal division when assessing the evacuation time and distance travelled distribution, in this case, where pedestrians start at the same point, they are instead grouped by the exit portal that they've taken.

4.8 Real World Environments in Concoursia

Plans of real-world environments were used in order to test the Concoursia platform. The first example is a shopping mall layout spanning a single floor with the internal layouts obtained from Google Maps. The environment was arbitrarily chosen as a

	Pedestrian exit count				Pedestrian exit percenage			
	Env 1	Env 2	Env 3	Env 4	Env 1	Env 2	Env 3	Env 4
Exit 1	54109	24667	12901	6178	49.56%	24.67%	12.90%	6.18%
Exit 2	55077	25264	12552	6208	50.44%	25.26%	12.55%	6.21%
Exit 3	-	25524	12645	6272	-	25.52%	12.65%	6.27%
Exit 4	-	24545	12266	6419	-	24.55%	12.27%	6.42%
Exit 5	-	-	12754	6305	-	-	12.75%	6.31%
Exit 6	-	-	12479	6359	-	-	12.48%	6.36%
Exit 7	-	-	12119	6187	-	-	12.12%	6.19%
Exit 8	-	-	12284	6333	-	-	12.28%	6.33%
Exit 9	-	-	-	6213	-	-	-	6.21%
Exit 10	-	-	-	6088	-	-	-	6.09%
Exit 11	-	-	-	6205	-	-	-	6.21%
Exit 12	-	-	-	6415	-	-	-	6.42%
Exit 13	-	-	-	6274	-	-	-	6.27%
Exit 14	-	-	-	6202	-	-	-	6.20%
Exit 15	-	-	-	6196	-	-	-	6.20%
Exit 16	-	-	-	6146	-	-	-	6.15%

Table 4.2 Exit count and percentage for the Tree environments from branching number 1 to 4 over 100 simulations

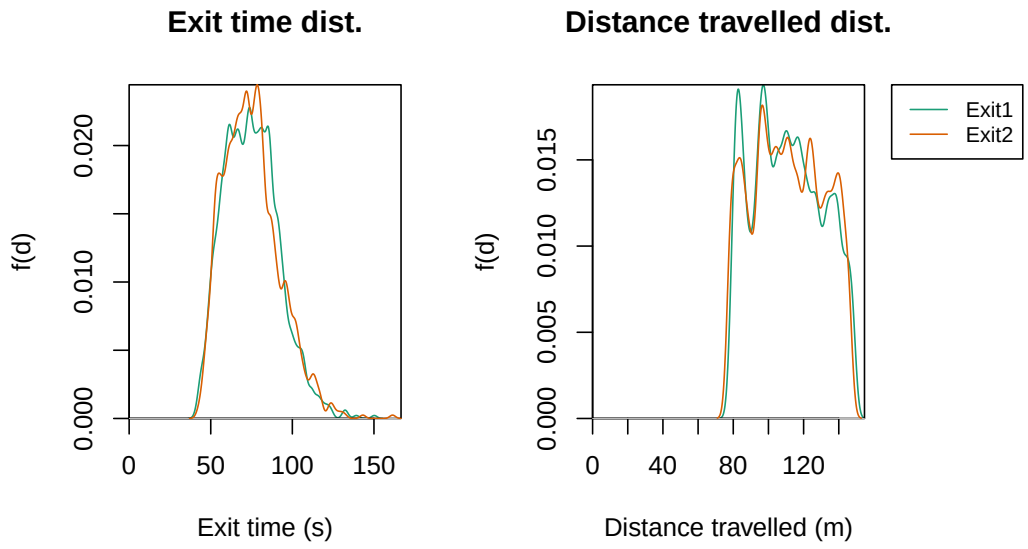

Figure 4.14 Exit time and Distance travelled distribution for the Tree environment with one level of branching. Distributions are separated by the pedestrians' chosen exit. The distribution is obtained over 100 simulation runs.

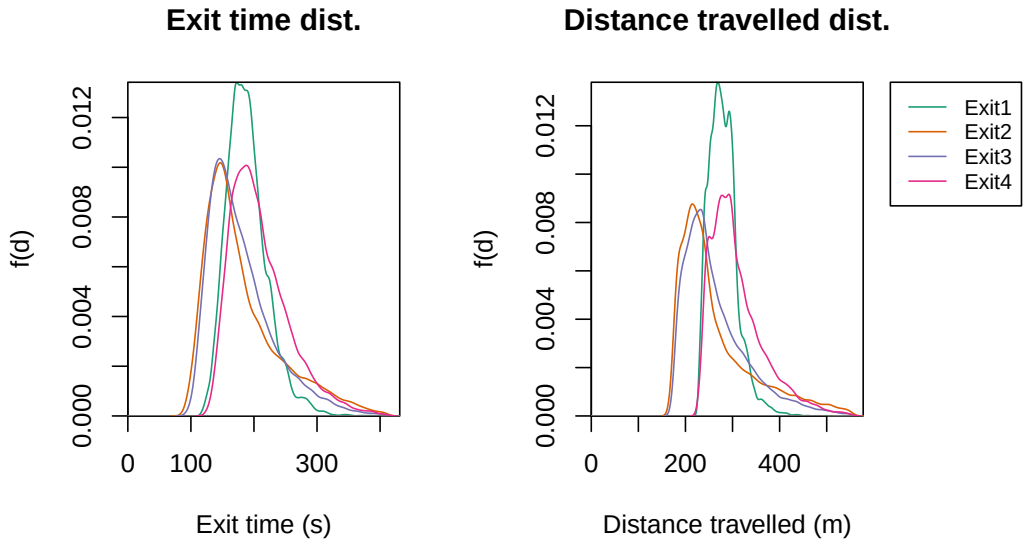

Figure 4.15 Exit time and Distance travelled distribution for the Tree environment with two levels of branching. Distributions are separated by the pedestrians' chosen exit. The distribution is obtained over 100 simulation runs.

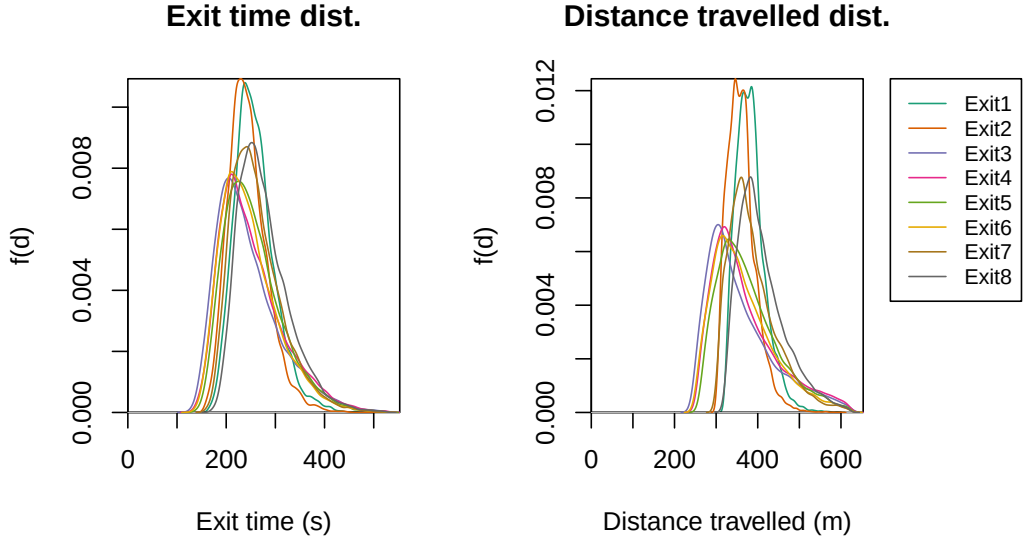

Figure 4.16 Exit time and Distance travelled distribution for the Tree environment with three levels of branching. Distributions are separated by the pedestrians' chosen exit. The distribution is obtained over 100 simulation runs.

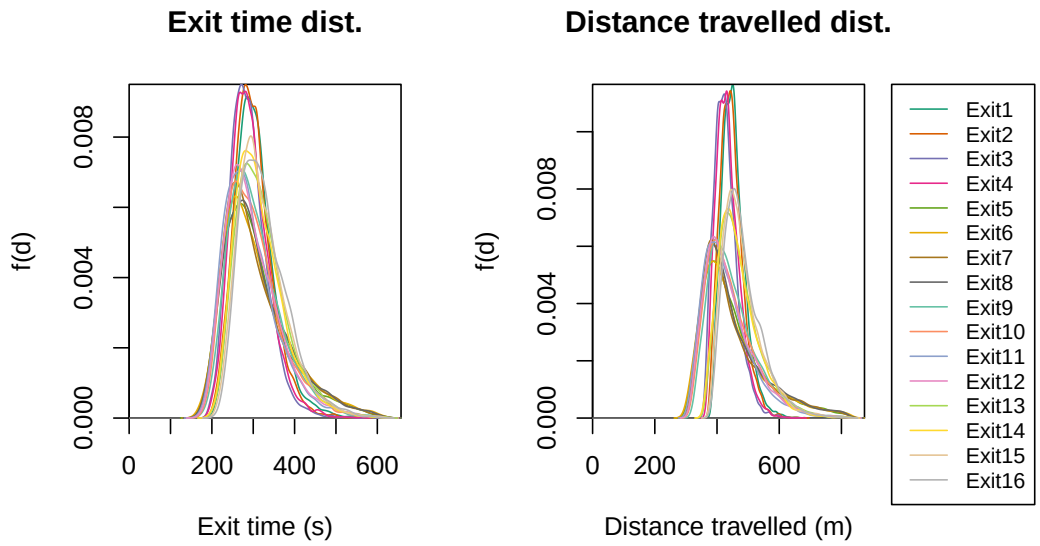

Figure 4.17 Exit time and Distance travelled distribution for the Tree environment with four levels of branching. Distributions are separated by the pedestrians' chosen exit. The distribution is obtained over 100 simulation runs.

sample for testing the initial software. The second example is a layout of Clapham junction train station in London obtained from our industrial partner Network Rail. This spans over three floors and utilises all features within the Concoursia software. Although congestion data is not available for use as a comparison, simulation speed, memory use, and environment build time are used for benchmarking. For both environments, two scenarios, are used with one representing normal operations and the other an evacuation scenario. All benchmarks in this section are performed on an Intel Intel i7-3770K machine with 16GB of RAM and an Nvidia GTX Titan graphics card.

4.8.1 Shopping Mall

The shopping mall has a size of approximately $0.099km^2$ (Fig. 4.18). It has four entrances (Portals) and 12 shops (Waypoints) which are used as pedestrian objectives. As the goal is to simulate only a single floor, elevators, escalators and stairways were not included. During the normal behaviour scenario, each pedestrian enters from one of the entrances, randomly select shops to visit and exit the environment through another entrance. In the evacuation scenario, pedestrians are generated from within the shops and along the main corridor and must exit the environment using the nearest exit.

Table 4.3 shows the memory when using the graph-based navigation system. The coverage refers to the amount of walkable space encapsulated by the Environment

Coverage (%)		97
Environment Graph	Node	4,792
	Edges	10,914
	Mem (kB)	145
R-tree	Nodes	7,191
	Mem. (kB)	144
Obstacle (kB)		139
Avg. Mem per Nav Graph (kB)		134
Build Time (s)		19.4

Table 4.3 Graph-based memory use for the Shopping mall environment.

Cell width (m)		0.25
Grid Size (WxH)		1,257x1,266
Walkable cells (%)		41.13
Collision	Per cell (B)	12
	Per field (MB)	19.09
Navigation	Per cell (B)	8
	Per field (MB)	12.73

Table 4.4 Grid-based memory use for the Shopping mall environment.

Graph, which is 97% in this case. Navigation for the environment uses base memory of 428KB in total. As a navigation graph's memory consumption is variable per itinerary and depends on the objectives within it, an example case is used entering from the south entrance and exiting at the north entrance which results in 21 additional edges and the total memory use is 134KB per navigation graph. The environment build, with all itineraries for both normal operations and evacuation, took an average of 19.4 seconds.

Table 4.4 gives the memory use for the grid-based approach. The environment contains relatively large proportion of walkable area due to the inclusion of shop floor space compared to the Train station example. For this environment, an obstacle avoidance field uses 19.09MB and each navigation field uses 12.73MB. Walkable areas only make up 41.13% of the memory as the diagonal alignment of the environment means a lot of wasted space on the top right corner. The memory requirements for the graph-based approach is only 2.24% for the base data such as collision avoidance and search trees and 1.05% for each Navigation Graph/Itinerary compared to grid-based approach. For this example where there might be 16 itineraries for ease of comparison, one for each Portal and one for each shop, the memory use will be 2.57MB for graph-based as compared to 222.77MB for grid-based navigation.

For the normal scenario 1000 pedestrians are emitted from each Portal over the duration of an hour. The number is an estimation for a busy time based on yearly visitor figures to the site. Each pedestrian's objective is to visit one shop with a random delay time, then exit the environment through another Portal. The cumulative LoS map shown in Fig. 4.19 gives an indication of how the environment is utilised. Iteration

time is found to be proportional to the number of pedestrians in the environment and a filled area graph (Fig. 4.20) is used to show the time for each agent function with respect to the number of pedestrians averaged over 100 simulation runs. The function names stated in the graph corresponds with the agent functions from Fig. 4.8. The timing is recorded and averaged every 50 iterations. The graph (Fig. 4.20) shows that the navigation graph access in the `navigate` function has a roughly constant time cost even with increasing number of pedestrians. Inter-pedestrian communication cost however, used for local collision avoidance in the `avoid_pedestrian` function, increases with the number of pedestrians as expected when pedestrians can perceive more pedestrians within their communication range at higher densities.

For the evacuation scenario, 1100 pedestrians, an approximate peak number during the normal scenario, are emitted in the first two minutes from the 12 shops and must evacuate to the nearest exit (Fig. 4.21). Iteration times averaged over 100 simulation runs are shown in Fig. 4.22 and indicate similar behaviour to the normal case. The overall average walking speed of the pedestrian is $1.502ms^{-1}$ with standard deviation of $0.18ms^{-1}$ which is in line with the desired speed of $1.5ms^{-1}$ with standard deviation of $0.2ms^{-1}$, which suggests that no significant congestion occurred. A more detailed plot of distance travelled over the time taken to evacuate is shown in Fig. 4.23 where the locally weighted scatterplot smoothing (LOWESS) line is shown in red.

Figure 4.18 A shopping mall environment with four entrances and 12 shops. The grey colour represent walkable areas, green capsules are portals, and blue rectangles are shops.

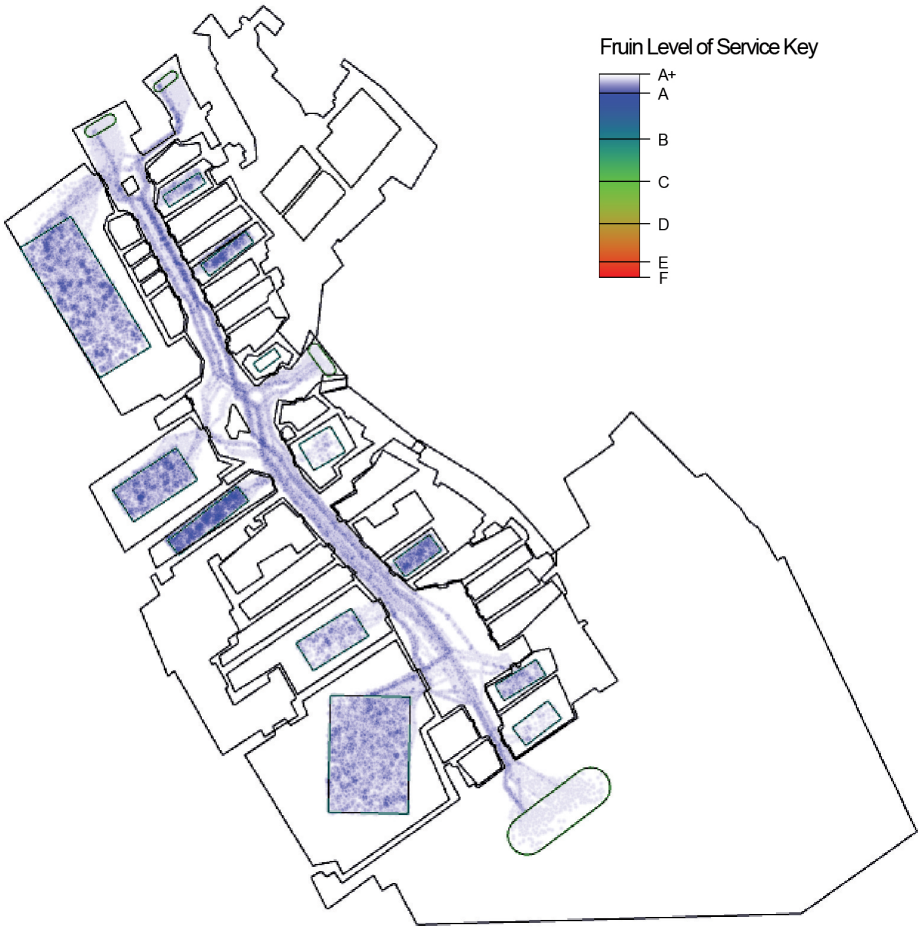

Figure 4.19 Shopping mall normal scenario - a cumulative LoS map of the simulation run.

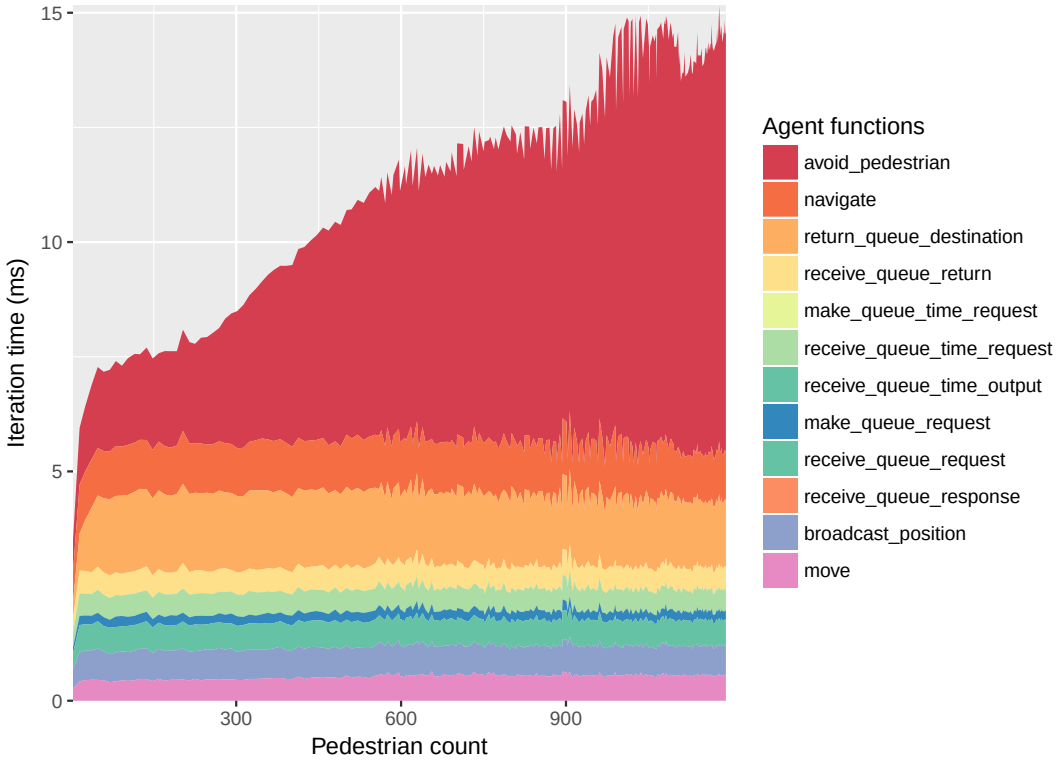

Figure 4.20 Shopping mall normal scenario - an iteration time over number of pedestrian graph broken down by the agent functions specified in Fig. 4.8 averaged over 100 simulation run.

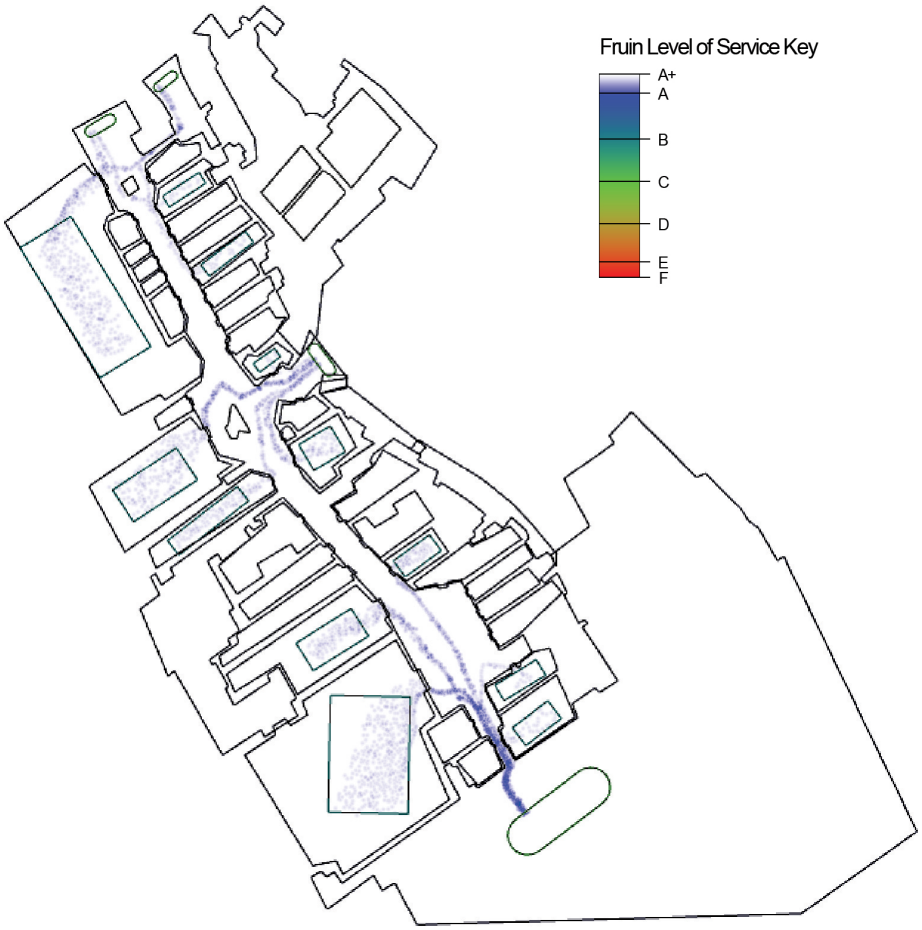

Figure 4.21 Shopping mall evacuation scenario - a cumulative LoS map of the simulation run.

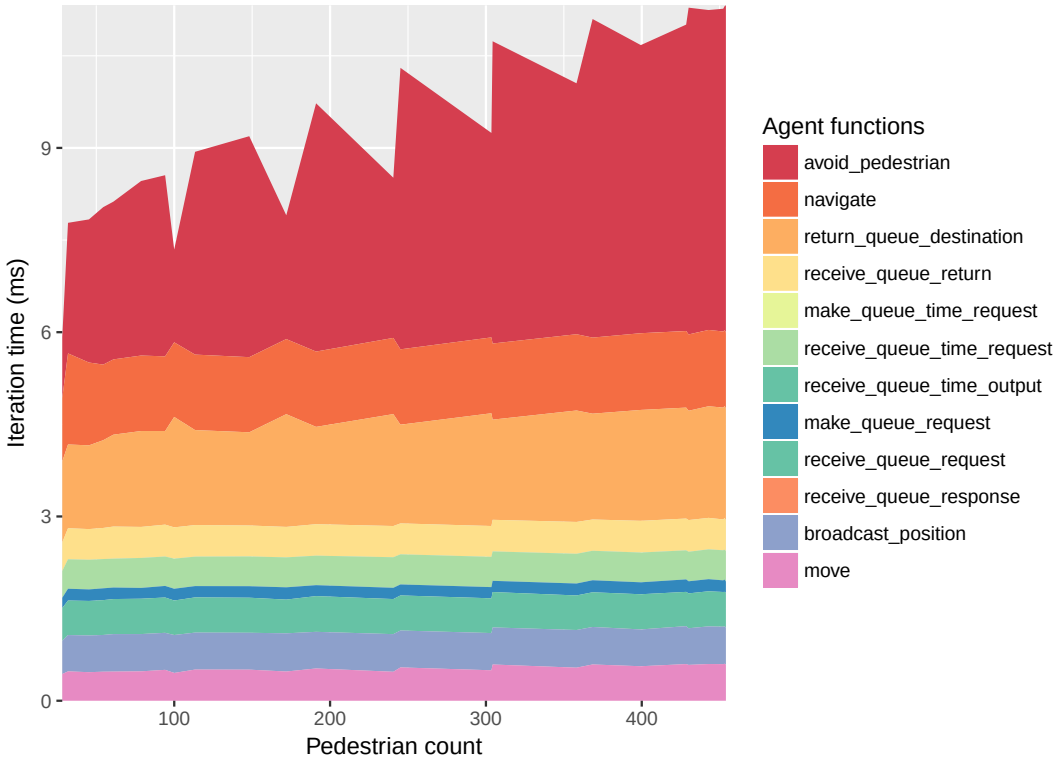

Figure 4.22 Shopping mall evacuation scenario - an iteration time over number of pedestrian graph broken down by the agent functions specified in Fig. 4.8 averaged over 100 simulation runs.

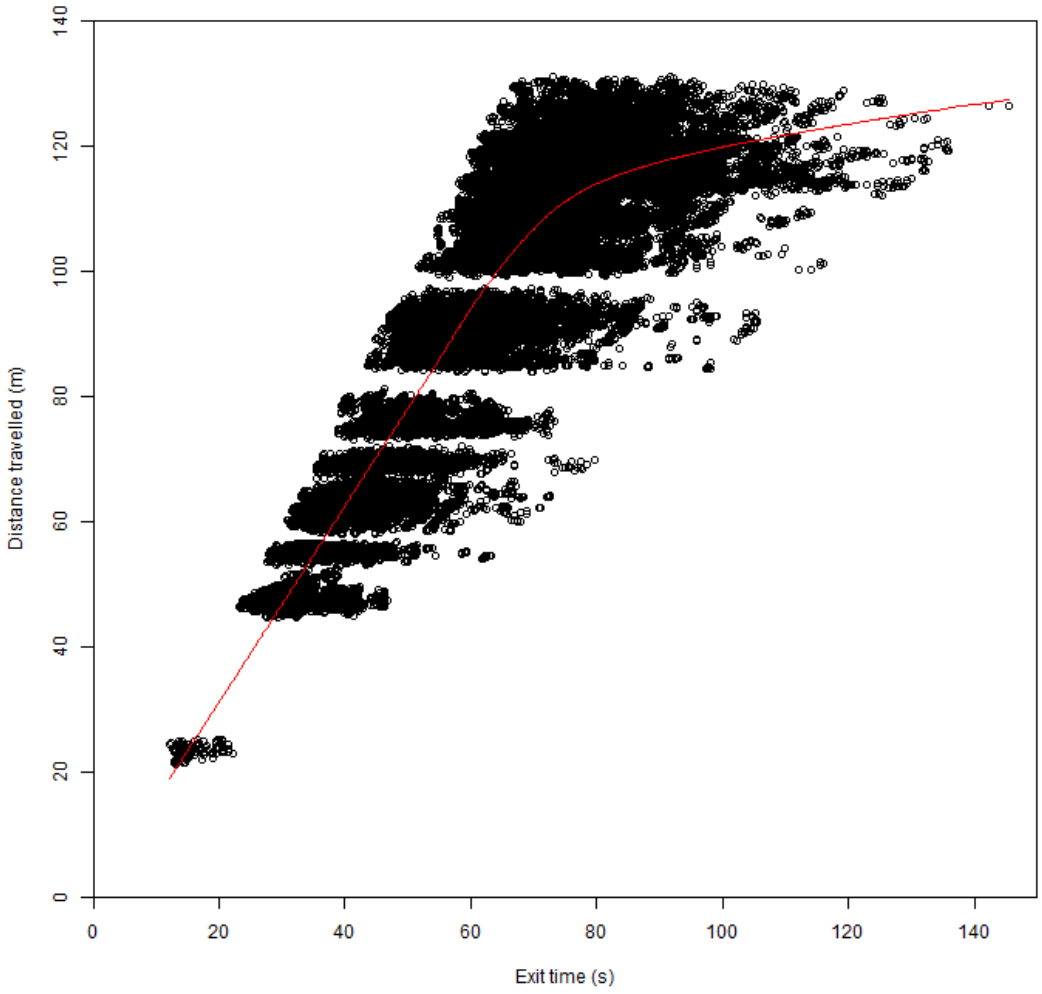

Figure 4.23 Shopping mall evacuation scenario. A scatter plot of Exit time (time to exit the environment) against Distance travelled over 100 simulation runs. The red line represents the LOWESS line.

4.8.2 Train Station

The train station has an area of $0.2km^2$, 4 entrances, 1 bus stop outside a main entrance and 17 platforms spanning over 3 levels connected by stairs (Fig. 4.24). A small train and bus schedule was crated in order to test the entire functionality of the software. There are ticket stations near all of the entrances (purple squares with black outlines). Screenshots of the simulation running in Concoursia can be seen in Figs. 4.30 and 4.31.

As can be seen from Table 4.5, graph-based navigation with coverage of 92% result in 8,698 nodes with 18,691 edges of the Environment Graph. Memory consumption is 254KB for the Environment Graph, 261KB for the environment search tree and 974KB for static obstacle data structure's memory consumption making a total of 1.51MB. An example itinerary of going from all entrances to Platform 1 requires one navigation graph of 240KB. It took an average of 90.2 seconds to build the environment with all itineraries for both normal operations and evacuation.

For the grid-based navigation example (Table 4.6) the obstacle avoidance grid is 43.64MB and each grid with navigation information is 29.09MB with only 12% of the cells walkable. The higher wastage ratio is due to the environment consisting of mostly long narrow and irregularly shaped corridors.

To compare the memory use of the two approaches, 21 itineraries are used, one for each objective going to the entrances, bus stop, and all the platforms. This gives a total memory use of 6.52MB for the graph-based navigation compared to 653.89MB for the grid-based navigation. The memory required for graph-based navigation is only 1% of what is needed for the grid-based navigation and would reduce further should there be more itineraries.

The normal scenario takes place over an hour with 12 train services that arrive at regular intervals. The pedestrian emission rates are based on the combination of entrance and exit counting and pedestrian movement tracking data provided by Network Rail. A provided origin destination matrix for the time 7am to 8am indicates 5,937 pedestrians entering and 1,196 exiting over the hour. The pedestrian tracking data is used to estimate the proportion of pedestrians to have exited and entered each train. A pedestrian coming into the station via the 3 main entrances to catch a train service must get a ticket before going through the turnstile and waiting at the platform before boarding the train. Pedestrians getting off the train travel to one of the main entrances to exit the environment. Pedestrians will queue at a ticket machine if the machine is busy. The cumulative LoS map shown in Fig. 4.25 gives an indication of how the environment is utilised and shows that only minor congestion occur near Entrance 1, 2 and the bus service stop. The area plot in Fig. 4.26 shows iteration times broken down by agent functions. Similar to the Shopping mall example, the

Coverage (%)		92
Environment Graph	Node	8,691
	Edges	18,691
	Mem (kB)	245
R-tree	Nodes	13,050
	Mem. (kB)	261
Obstacle (kB)		974
Avg. Mem per Nav Graph (kB)		240
Build Time (s)		90.2

Table 4.5 Graph-based memory use for the Train station environment.

Cell width (m)		0.25
Grid Size (WxH)		2,386x1,524
Walkable cells (%)		12.04
Collision	Per cell (B)	12
	Per field (MB)	43.64
Navigation	Per cell (B)	8
	Per field (MB)	29.09

Table 4.6 Grid-based memory use for the Train station environment.

graph (Fig. 4.26) shows that the `navigate` function has a roughly constant time cost and that inter-agent communication in the `avoid_pedestrian` function increases with the number of pedestrians in the environment.

For the evacuation scenario, each of the 17 platforms emits 100 pedestrians in the first minute and each pedestrian uses the nearest exit to evacuate. For a single simulation, a cumulative LoS map is shown in Fig. 4.27 and iteration times averaged over 100 simulation runs are shown in Fig. 4.28. It can clearly be seen from the LoS map that the scenario generates enough pedestrians to cause a congestion and that most congestions occur at the ticket barriers where the exit is narrowest near Entrance 1 (red areas in the LoS map). This is similar to the example from Section 3.2.8 where a large corridor becomes narrow. The overall average walking speed of the pedestrians is $1.397ms^{-1}$ with standard deviation of $0.176ms^{-1}$. Compared to the desired speed of $1.5ms^{-1}$, a difference of $0.103ms^{-1}$ suggests that minor congestion occurred. A more detail plot of distance travelled over the time taken to evacuate is shown in Fig. 4.29 where the LOWESS line is shown in red.

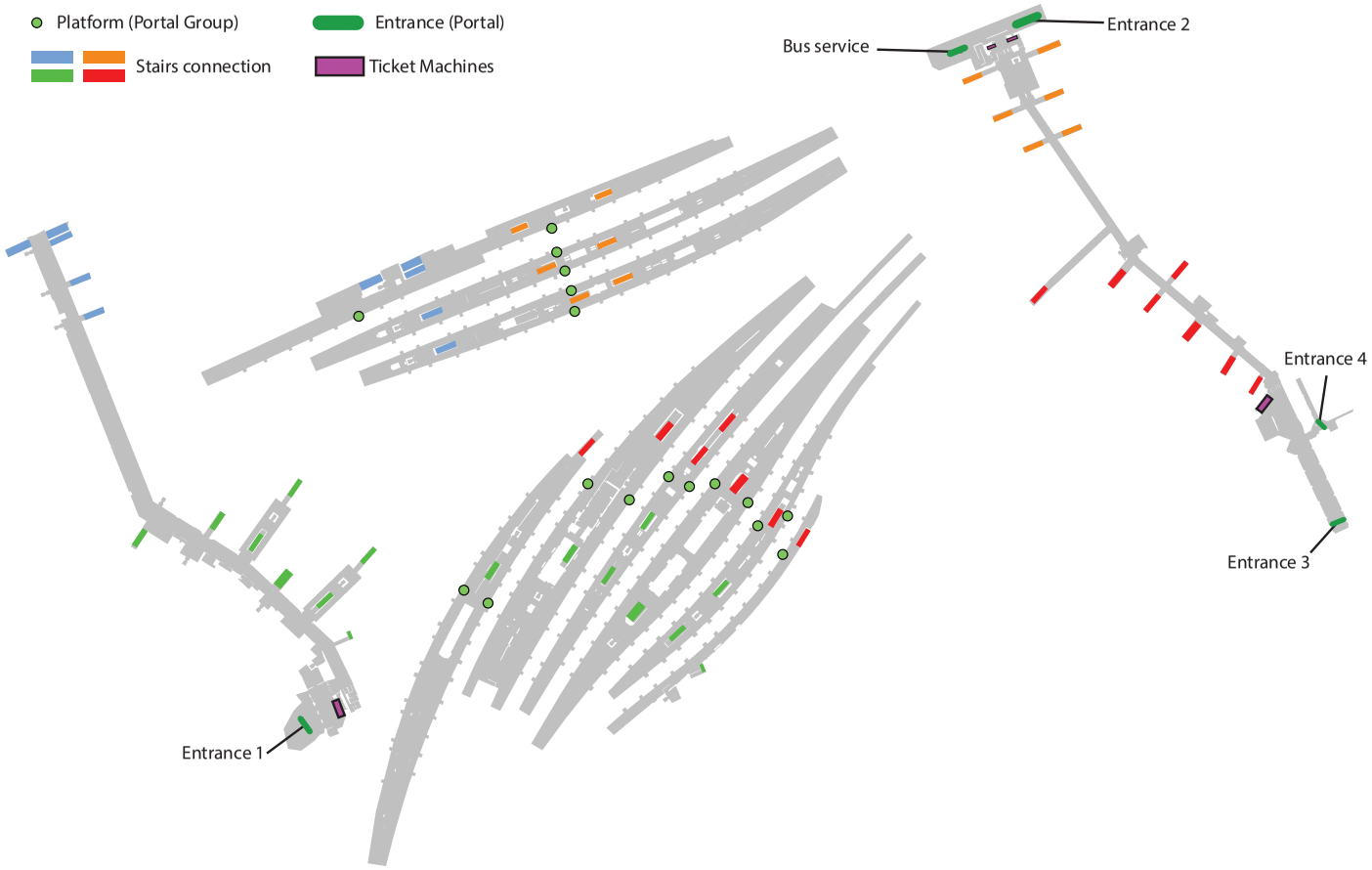

Figure 4.24 A train station environment with three separate sections, 4 entrances, 1 bus service and 17 platforms. Gray colour represent walkable areas. The sections are connected by stairs grouped by blue, orange, red and green rectangles. Each platform is a group of portals representing each set of train doors. The purple rectangle with black outline represent ticket machines.

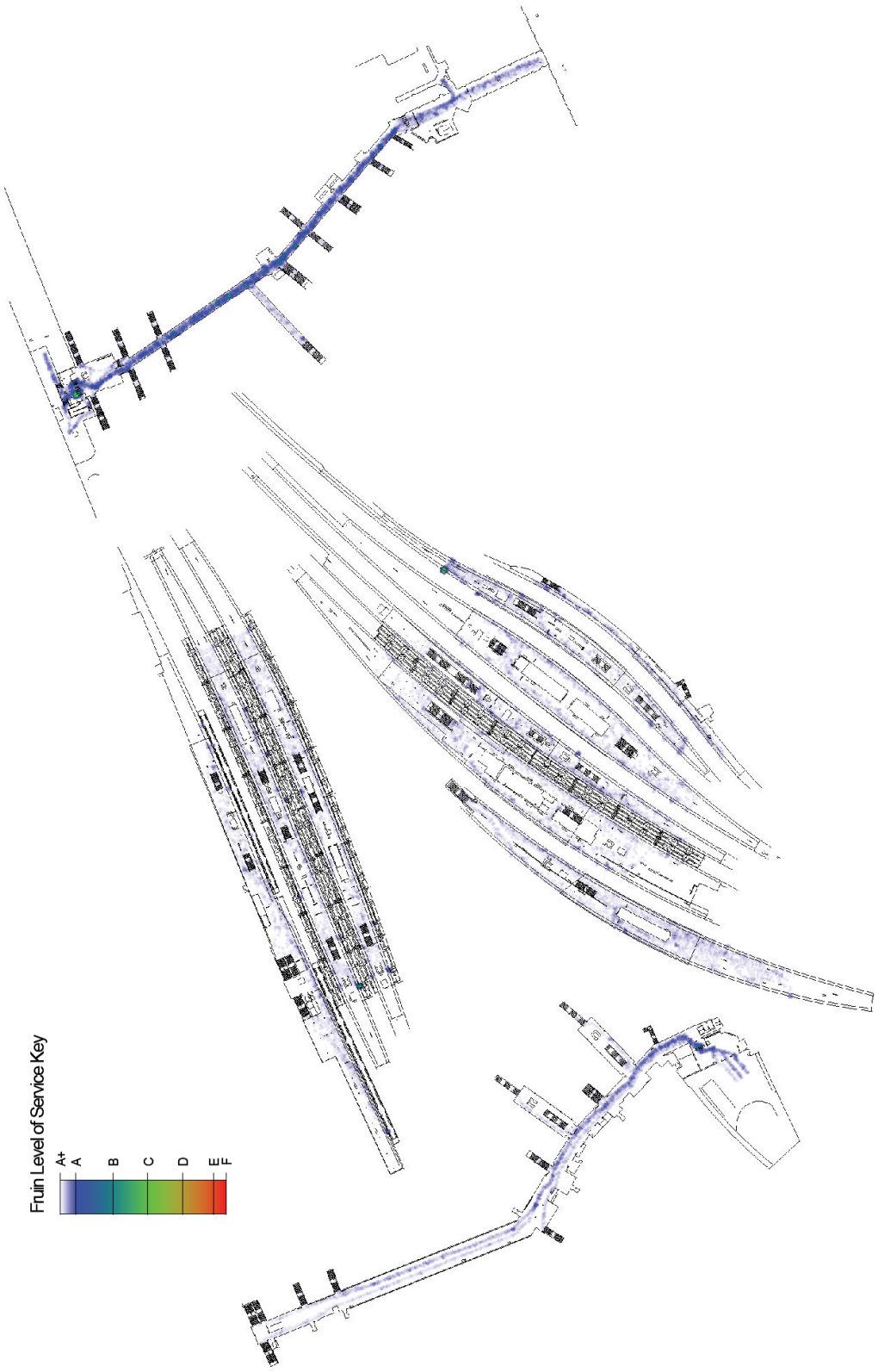

Figure 4.25 Train station normal scenario - a cumulative LoS map of the simulation run.

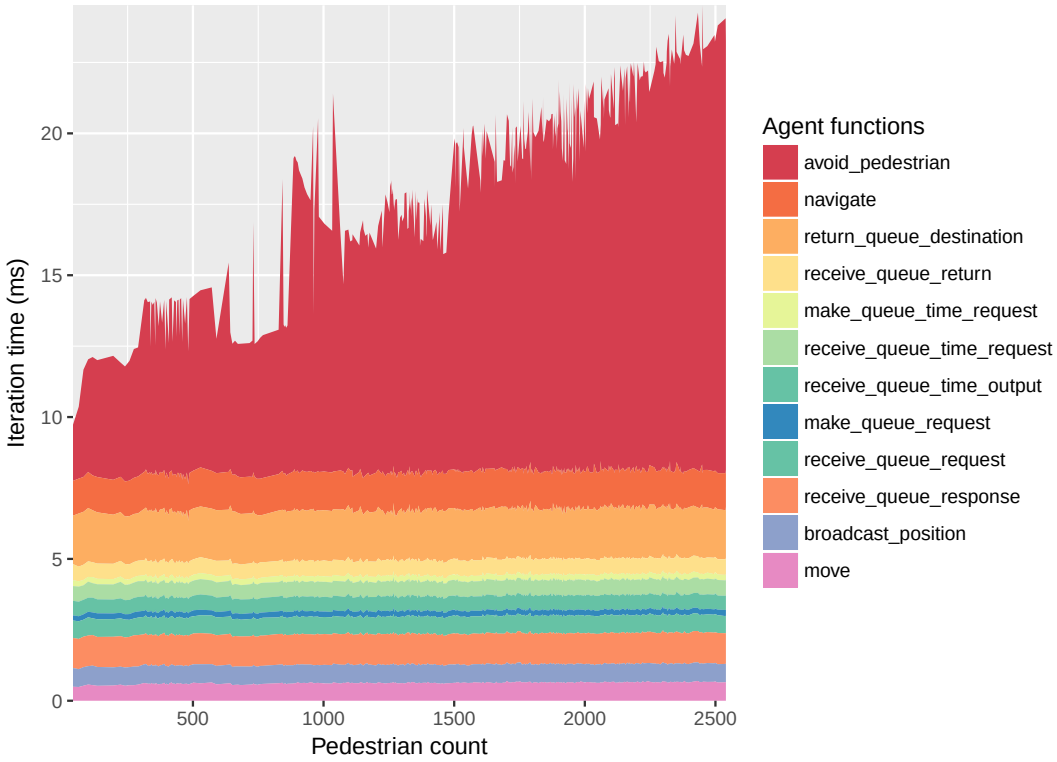

Figure 4.26 Train station normal scenario - an iteration time over number of pedestrian graph broken down by the agent functions specified in Fig. 4.8 averaged over 100 simulation runs.

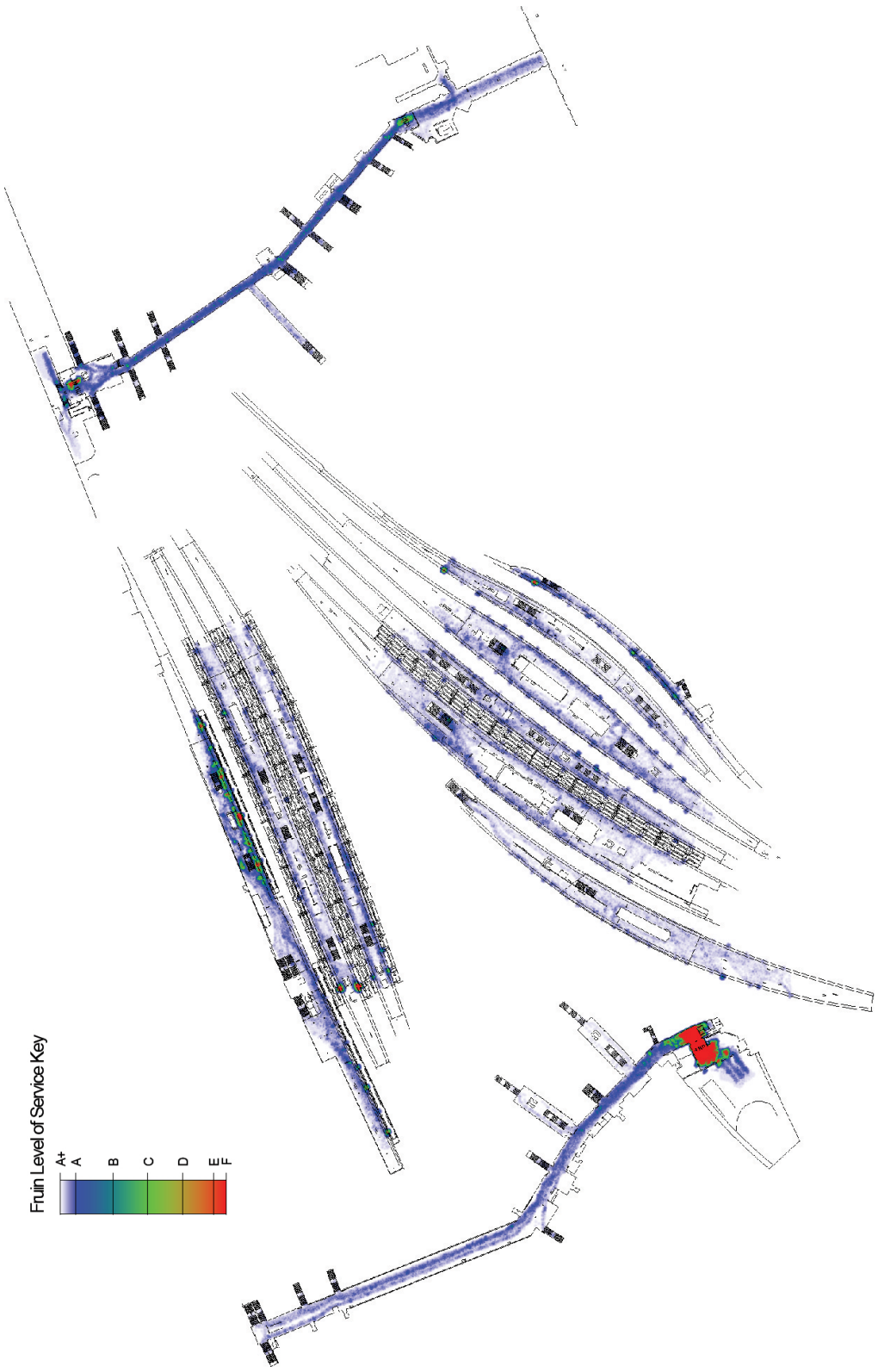

Figure 4.27 Train station evacuation scenario - a cumulative LoS map of the simulation run.

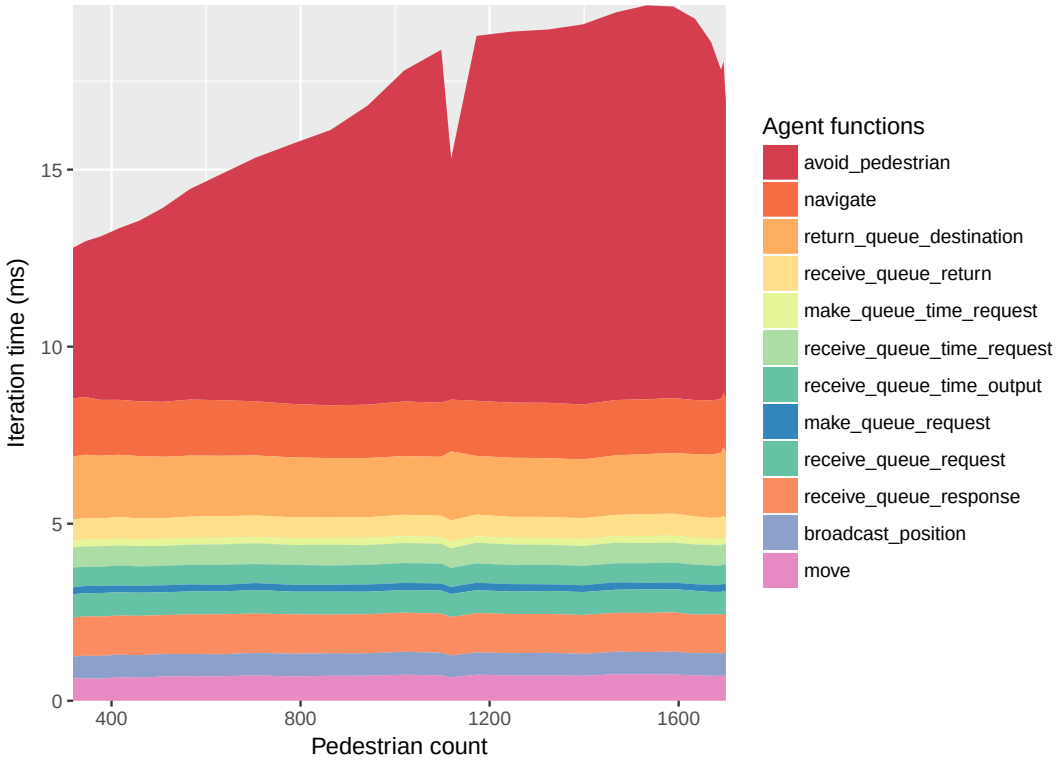

Figure 4.28 Train station evacuation scenario - an iteration time over number of pedestrian graph broken down by the agent functions specified in Fig. 4.8 averaged over 100 simulation runs.

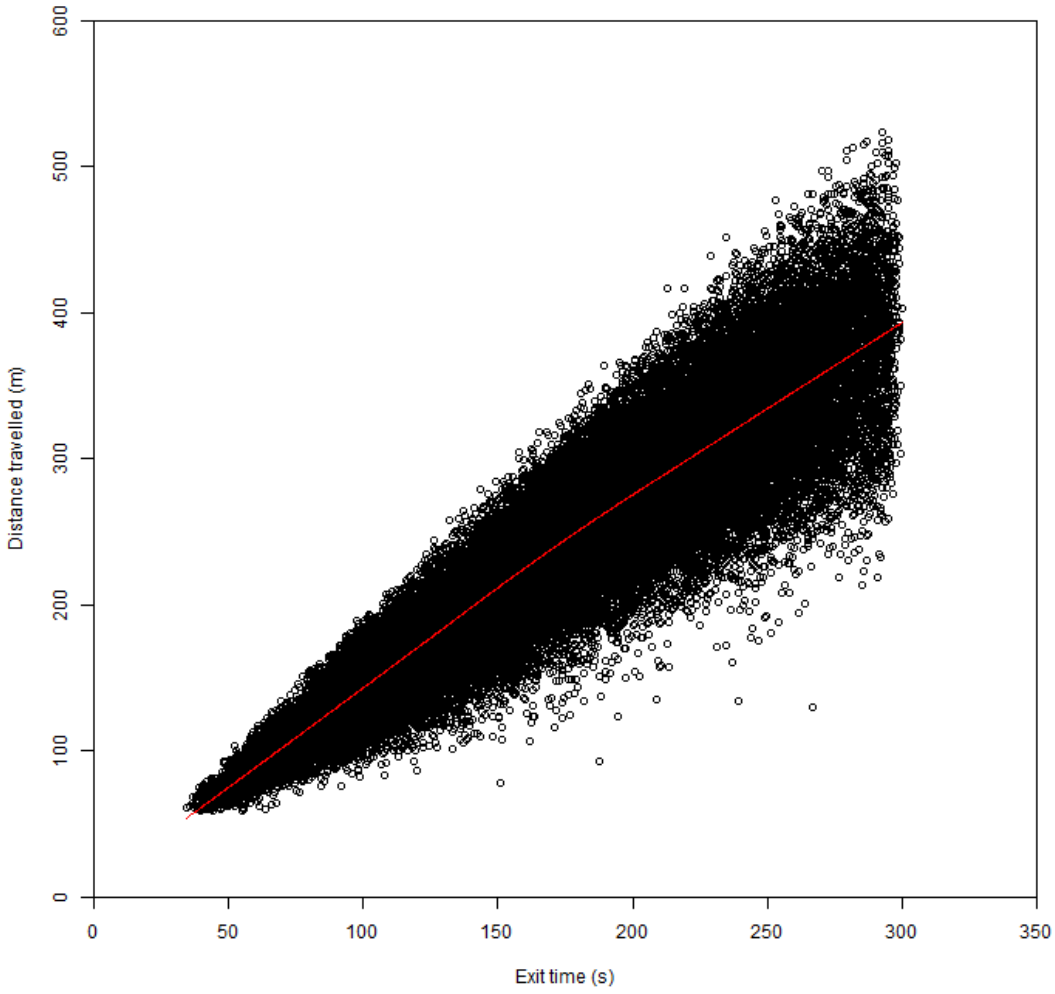

Figure 4.29 Train station evacuation scenario - A scatter plot of Exit time (time to exit the environment) against Distance travelled over 100 simulation runs. The red line represents the LOWESS line.

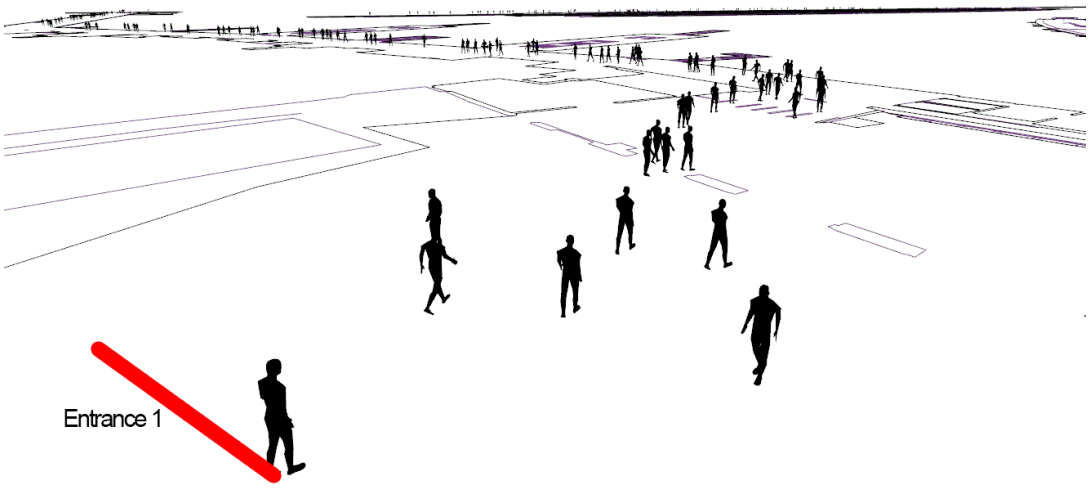

Figure 4.30 Screenshot of the train station scenario (Fig. 4.24) running in Concoursia. The view shows the station's Entrance 1.

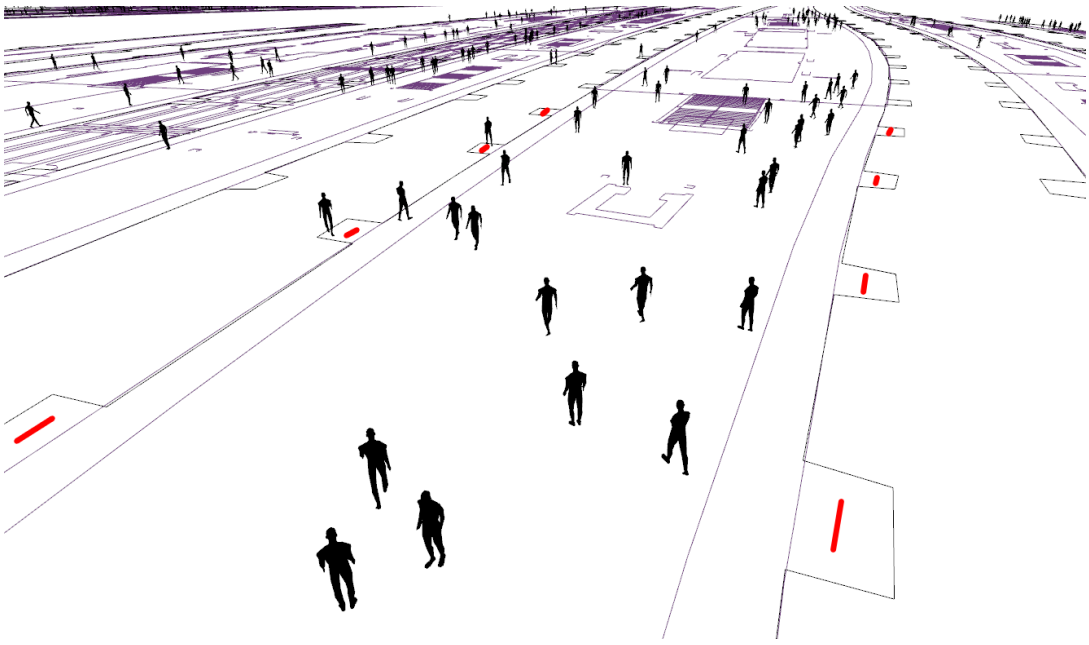

Figure 4.31 Screenshot of the train station scenario (Fig. 4.24) running in Concoursia. The view focuses on one of the the station's platform. The red areas indicate train carriage entrance.

4.9 Discussion

The previous examples show that Concoursia is capable of creating and simulating pedestrian flow in a large and complex multi-floor environment in a range of scenarios. The ability to edit the environment and simulate in the same system allows for rapid and interactive testing of scenarios. Such scenarios can be used for designing a new environment or planning an intervention to potential problems to support the final decision on the implementation of crowd control measures. The Shopping mall scenario (Section 4.8.1) took around 1 hour to implement from scratch while the more complex Train station scenario (Section 4.8.2) took roughly 4 hours, with all data about schedule events prepared beforehand. Concoursia is however, still a prototype system and still lacking in some features that can be found in more mature platforms.

Chapter 3 presented findings that the graph-based navigation system excels in large environments with varying complexity. The examples in Section 4.8 support this finding and show that, when applied to real environments (Tables 4.3 to 4.6), the memory use of the graph-based approach approximately 100 times less than the grid-based approach. This difference is especially relevant in larger environments where pedestrians may have many more objectives. This case can be illustrated by envisioning the simulation of multiple inter-connected facilities (e.g. an airport connected to a train station). Itineraries for pedestrians using a single facility and a combination of facilities will need to be created causing a large rise in the number of itineraries.

The access times for navigation information (Fig. 4.26) has shown to be stable at up to 2,800 pedestrians and, with R-tree access times being $O(log^n)$, it is expected to scale well even when more pedestrians are introduced or if the complexity of the environment increases. While it can be seen in Fig. 4.26 that other agent functions remain constant with increasing numbers of pedestrian, the bottleneck is the inter-pedestrian communication. The `avoid_pedestrian` function retrieves location messages from other pedestrians that are within range. These location messages are spatially partitioned so that each pedestrian agent can retrieve only messages that are within a certain distance. When the environment is more crowded, however, each pedestrian will receive more messages and hence take more time to process all the messages.

The results in Section 4.8 show that Concoursia is capable of running simulations much faster than real-time. When simulating 2500 pedestrians it is able to simulate at a speed of approximately 25ms per iteration, which, at 0.2s time step per iteration means the simulation is running 8 times faster than real-time. Further optimisation should also be possible by limiting pedestrian communication range and reducing pedestrian location messages that need to be processed by each pedestrian during congestion. The

amount that the simulation has to run faster than real-time is very much dependent how far in the the future a problem can be predicted and how long it takes to implement the longest intervention. Consider an example where a problem can only be predicted with good accuracy 45 minutes from the current real-time data (e.g. a train delay at the next station will cause congestion at this station) and that an intervention must be executed at least 30 minutes before the problem (e.g. deploy staff to warn people to stop coming into the station/platform). If the time it takes for an operator to use the system is 5 minutes, there is a time window of around 10 minutes to predict 45 minutes ahead to forecast the problem and then another 45 minutes to predict the result of an intervention (total of 90 minutes). We must then have the simulation run at least 9 times faster than real-time to get a prediction within the 10 minute window.

The ORCA model by van den Berg et al. [129] was used in the pedestrian model due to the attractiveness of the velocity-based approach, which allows for the use of a fixed time-step and the use of K-trees for accurately representing the static environment. Viswanathan et al. [277] showed it is the social force model that is closest to observed pedestrian behaviour when evacuating from a room. However, Wolinski et al. [278] observed that the ORCA(RVO2) model has a more realistic avoidance movement then the social force model when compared to captured video due to pedestrians anticipating each other's movement. Given more time, it may be possible to introduce a more social force like behaviour to the ORCA model. Alternatively, the individualised continuum based approach by Wolinski et al. [36] is attractive, provided that the simulation can be accelerated as the performance is currently stated to be around 15% of what's achievable with ORCA. However, a pedestrian's local movement behaviour consists of far more than collision avoidance. With further research, there are also other local movement behaviours that could be represented such as grouping, someone pushing a wheel chair or someone pulling luggage behind them.

It was not possible to obtain a real dataset for congestion evaluation as the industrial project concluded before the industrial partners could provide access to the location. To be able to provide a more accurate simulation of the pedestrian flow, the model should be calibrated similar to the approach by Berrou et al. [279], by measuring pedestrian's desired speed, personal space, and flow is measured at a more systematic level. While tracking technology was explored for data collection, approaches such as Bluetooth and Wifi tracking are very inaccurate as they require the pedestrian to have a mobile phone and also do not give precise location information (Section 2.4.1). Installation of cameras or radar-like devices would be optimal for data collection but the process is costly and can be disruptive in order to obtain the needed coverage.

On the interface and editing side, Concoursia is still lacking tools that can be used to precisely edit the environment which can be found in commercial systems

such as Adobe's Autocad. It is also lacking the ability to import or directly enter spreadsheet-style data that could reduce the time needed when specifying event schedules. Additionally it currently does not interface with other systems for automatically downloading data such as train schedules.

Elevators have yet to be implemented and while it was not essential for the scenarios presented (and in emergency situations they are normally turned off) they do have an effect on the pedestrian flow. This is true especially in some locations such as major airports and tall building complexes that normally use lifts or shuttles as the main way of transporting pedestrians from one location to another.

Due to the stochasticity of the model, each simulation of the same scenario should be run multiple times with different random seeds and results averaged to get a good distribution spread. In addition, a Real-time Decision Support System (RDSS) would continuously run new simulations based on live sensor data, and, should it forecast potential problems, it will need to run multiple simulation with various interventions for shaping the pedestrian flow to safe levels. This ability to run multiple similar models at the same time and average the results, even ranking them by efficacy, becomes an essential requirement and is a feature which does not yet exist in Concoursia. Chapter 5 presents a prototype multi-simulation tool and touches on the need to run multiple concurrent simulation. It presents a system that can potentially be integrated into the future development of Concoursia.

Chapter 5

A Prototype System for Multi-Simulation of Pedestrian Models

As discussed at the end of Chapters 2 and 4, many types of intervention in an environment can be used for reducing congestion and maximising pedestrian flow. For each type of intervention there are many possible configurations. As an example, consider the redesigning of a space. It may be that a corridor is made into a 'V' shape or that a column could be placed near an entrance as both solutions have been found to help increase flow during congestion [118]. When testing the effectiveness of each configuration by using a simulation, it is also necessary to run it multiple times in order to get a good distribution spread due to the stochastic nature of the pedestrian model. In order to explore these multiple interventions and configurations within acceptable time, a system is needed for performing the batching of these simulations with varying configurations that utilises the capabilities of available hardware within a network.

This chapter presents a prototype system for the simulation of pedestrian models with various configurations across a number of machines, equipped with one or more GPUs, over a network. The FLAME GPU framework is used to provide the simulation capabilities for the pedestrian model. An overview is given of the components of the system in Section 5.1. The batching of simulations, which requires modification to the FLAME GPU framework, is discussed in Section 5.2. Metrics collection when simulating multiple simulation intances on a single GPU due to batching is outlined in Section 5.3. A scenario based on a dirty bomb explosion within a shopping mall is used as a case study for the application of the system and is outlined in Section 5.4. The agent model used to represent the scenario is then discussed in Section 5.5. Finally, preliminary results of the system are shown in Section 5.7, using

real-world environments based on Nottingham and Sheffield town centre. Benchmarks are provided for running multiple simulations on multiple GPUs on the same node. These demonstrate the ability of the software to run concurrent simulation on multiple GPUs faster than real-time.

5.1 System Overview

In order to create a system that is modular and scalable to a network of GPU-enabled computers, a distributive approach was taken whereby the system is split into different applications each with their own functionality.

The system consists of four separate applications which are the Simulator Manager, Local Simulator Manager, Simulator and GUI Client. The modules form a client-server relationship where the Simulator Manager can be considered as the server and the other as clients. A network diagram of the system is shown in Fig. 5.1. Inter-application communication is performed over network sockets (TCP/IP).

For each GPU utilised, efficiency is increased by merging of multiple smaller simulations (Section 5.2). During simulation runs, metrics are collected in order to be used for evaluating the effectiveness of a configuration (Section 5.3).

5.1.1 Simulation Manager

The Simulator Manager (middle of Fig. 5.1) can be considered as the host and communicates with all the applications. Simulation instances, which contain all agent and parameter data required to start a single simulation that are received from the Client, are queued or dispatched to Simulators with available capacity through the Local Simulation Manager.

The Simulation Manager is informed once an instance of a simulation has finished running. It can then request for the metrics to be sent to it through the Local Simulator Manager. The Simulation Manager stores the metrics data locally, ready to be sent in response to a request by the GUI Client.

5.1.2 Local Simulation Manager

Each node in the network used for running simulations requires an instance of the Local Simulation manager (bottom right of Fig. 5.1). It detects the number of Graphics Processing Unit (GPU)s available on the machine and spawns Simulators to handle simulation jobs sent by the Simulation Manager. It also keeps track of the number of simulations running within the entire node and the slots available to handle further

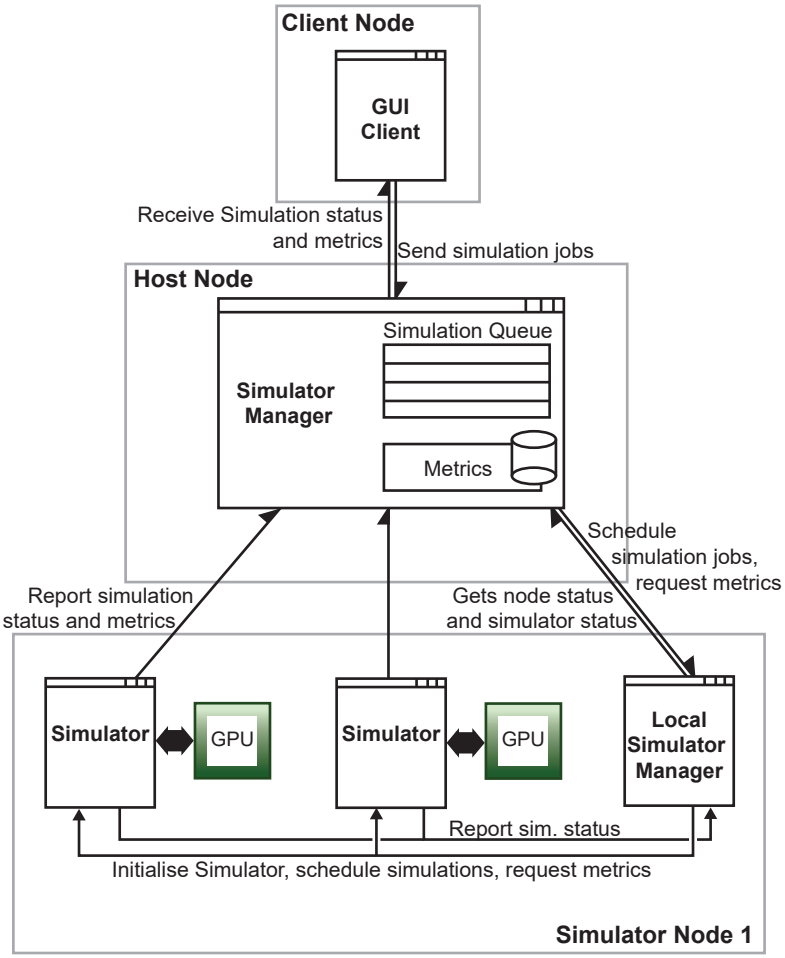

Figure 5.1 The network diagram shows how each of the application in the system interact.

simulation jobs. The Local Simulation Manager will then try to batch as many simulations on to a single GPU as possible to increase efficiency.

5.1.3 Simulator

The Simulator (bottom of Fig. 5.1) contains the simulation framework and is responsible for the actual simulation of the model. An instance of a Simulator is created for each GPU on the machine. This is done to avoid the complexity of extending FLAME GPU so that it supports running multiple instances on multiple GPUs in the same application. It also makes the application design simpler and more modular. The Simulator uses the FLAME GPU framework [237, 262] with modifications that enables simulation batching (which is covered in Section 5.2) and collection of metrics (which is covered in Section 5.3). The model used in the simulation was created based on the scenario outlined in Section 5.4. This consists of Pedestrian and Environment agents which are discussed in Section 5.5. In order to avoid overloading the network, all metrics data generated during a simulation run are saved locally on the machine and sent across the network when requested by the Simulation Manager.

5.1.4 GUI Client

The GUI Client (top of Fig. 5.1) is local to the user's machine and is the only part of the system that allows direct user interaction. The Client is able to create and load a simulation instance and forward it to the Simulation Manager for batch simulation. Metrics received from completed simulations are visualised either in comparison with other instances or individually. Pedestrian counts and zone counts are displayed as a graph over time, while Level of Service (LoS) is shown as an image (Section 5.3).

5.2 Simulation Batching

It has been demonstrated by Richmond et al. [40] that larger models, involving tens of thousands of pedestrians, are required in order to make effective use of the parallel processing capabilities of the GPU. For smaller models, a way to increase efficiency is by merging multiple simulations together and running them effectively as a single large simulation. When running multiple simulations together in the same memory space, considerations must be taken in order to avoid cross-contamination of the agents between different simulations. This section will discuss the mapping of agents of multiple instances on to the global memory, stopping cross-contamination between the simulations and how agents are unloaded when an instance finishes running.

Multiple environments can be supported by creating a global environment large enough to fit the number of local environments required. The local environment refers to the environment for each individual simulation. Fig. 5.2 shows visually how the mapping takes place for four instances of a simulation. The screenshot in Fig. 5.3 shows two simulation instances loaded and running at the same time. An alternative way to think of the global environment is as a grid containing each individual simulation. Thus, in case of the example in Fig. 5.2, the grid size would be 2 by 2 as it can fit two simulations row-wise and two column-wise. This will be referred to as the simulation grid. When mapping the local environment to the global one, its x and y coordinates can be translated using Equations 5.1 and 5.2.

$$x = x_0 + c_s width_l \tag{5.1}$$

$$y = y_0 + r_s height_l \tag{5.2}$$

The variables x_0 and y_0 are the local coordinates of the Environment agents. The variables x and y represents the global coordinates of the Environment agent. The variables c_s and r_s are the columns and rows, respectively, of the simulation in the simulation grid e.g. for the example in Fig. 5.2, $c_s = 2$ and $r_s = 2$. The variable $width_l$ is the width of the local environment and the variable $height_l$ is the height. Since the 2D environment grids are actually stored as a row-wise one dimensional array, with the new global x and y coordinates, the agents must also be written to the correct place in the array by using Equation 5.3.

$$env\_agent\_position\_in\_global\_array = y \times width_s + x \tag{5.3}$$

The variable $width_s$ represents the cell width of the simulation grid e.g., for the example in Fig. 5.2, $width_s = 6$. Similar x and y coordinate translation must also be applied to pedestrian agents.

Pedestrian agents exist in continuous space unlike Environment agents. This means the Pedestrian agents are stored in an unordered one dimensional list called the Global Pedestrian List (Fig. 5.2). Within the Global Pedestrian List, the Pedestrian agents from all running simulation can appear in any order (Fig. 5.4). When a Pedestrian agent is created in any simulation they are simply appended to the end of the Global Pedestrian List.

Pedestrian and Environment agents are assigned simulation ids in order to differentiate each other and prevent cross-contamination. This helps in boundary cases when pedestrians are performing local collision detection and when pedestrian agent

related metrics are collected as their location in the global pedestrian list is not sorted simulation by simulation. Pedestrians should run in environment that is completely bounded so they cannot walk outside their own simulation area. If one does go outside its own simulation area, i.e. a pedestrian gets a message from an Enviroment agent with a different simulation id, that simulation instance is halted and an error is reported.

When a simulation instance finishes running, the Environment agents are left as-is but the Pedestrian agents must be removed from the Global Pedestrian List. The process is not straightforward as the agent positions in the array are not ordered by id and must be shifted to replace the removed agents. The process for removing Pedestrian agents in parallel is illustrated in Fig. 5.4 and is done at the end of a simulation iteration. First, a one-dimensional array `do_copy` is created with length equal to the Global Pedestrian List. A CUDA kernel goes through the Global Pedestrian List in parallel and marks the position with 1 if it is to be copied and 0 if it is to be removed, the red pedestrians in this case. A parallel sum is then performed on the `do_copy` array so that we can obtain an array `new_index` that has the new index position without the red pedestrians. A new Global Pedestrian List is created and another CUDA kernel goes through the Global Pedestrian List and uses the `new_index[current_index] - 1` as the new index for copying the pedestrian data. Finally the reference to the Global Pedestrian List and the new Global Pedestrian List are swapped so that the simulation now uses the data in the new Global Pedestrian List.

5.3 Collection of Metrics

Metrics provide a way to summarise the result of a simulation run according to specific criteria and are collected as standard in commercial pedestrian simulation software such as the ones mentioned in Section 2.3. They are especially useful for comparing results of multiple simulation runs and various configurations where it is infeasible for the user to observe every iteration of every simulation run. Various metrics can be collected such as density, LoS, flow rates, evacuation time, or distance travelled. The usefulness of each metric varies according to the scenario, the zone count metric (Section 5.6.1) for example, would not be relevant in a scenario where the environment is not divided into zones.

Running the simulation on the GPU adds an extra complexity to the collection of metrics. As the agent's data is located on the GPU device, it is simply not efficient to have to synchronise this agent memory with the host machine's RAM before the data can be collected. Hence, the metrics should be collected on the GPU itself using parallel approaches. Additionally, due to simulation batching, metric results must be

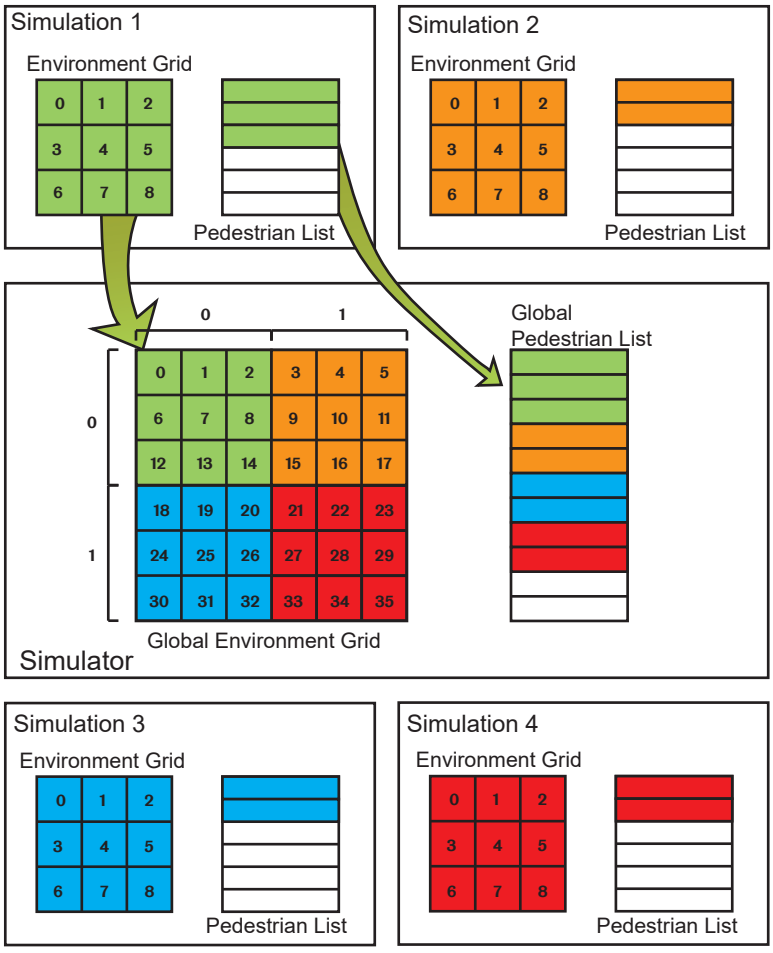

Figure 5.2 Multiple simulations can run concurrently in the global environment and make use of the Global Pedestrian List. Coordinates transformation must be performed in order to move from local to global and vice-versa.

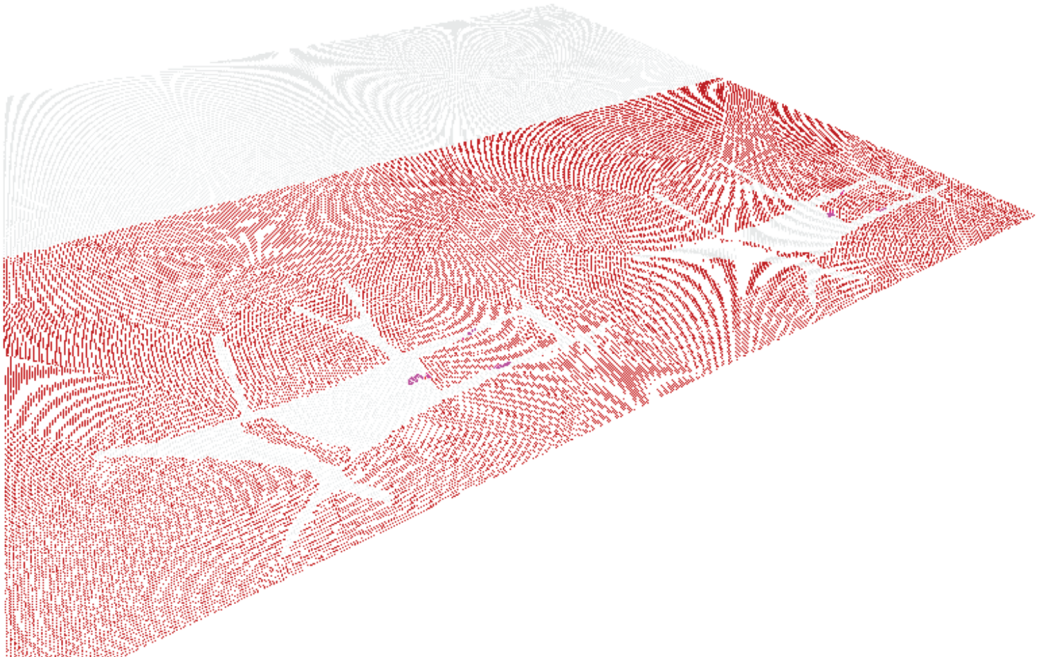

Figure 5.3 Simulator visualising two simulation instances running side-by-side.

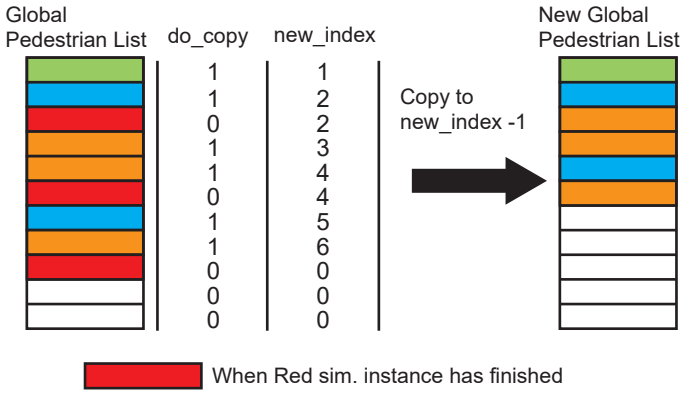

Figure 5.4 Parallel removal of pedestrian from the Global Pedestrian List. When a simulation instance stops, pedestrians of the stopped instance are removed from the simulation.

re-mapped with respect to each individual simulation. Although there are examples of pedestrian simulation on the GPU in the literature (Section 2.5.1), they do not explain the process by which metrics are gathered.

This section details the process in which two categories of metrics are collected using an approach suitable for a parallel architecture like the GPU. The first metric, the pedestrian generation metric (Section 5.3.1), is an accurate count of the number of pedestrians generated over a pre-defined time period. The second metric, the LoS metric (Section 5.3.2), presents a visual view of the aggregate density data.

These metrics can be collected at pre-defined time intervals or after every simulation step. By assigning longer intervals, the simulation performance can be improved due to a reduction in processing during the metrics collection and in the time required for transferring data out of the GPU. Parallel sum and parallel sort operations provided by the CUDPP library [280] are used to optimise the collection of metrics.

5.3.1 Obtaining an Accurate Pedestrian Generated Count

An accurate pedestrian emission count is crucial in enforcing an emission limit on the simulation. The environment agents are responsible for emitting pedestrian agents. In order to collect this emission information, it must transferred from the GPU to the Simulator. Each environment agent updates a count when a pedestrian is emitted. Fig. 5.5(a) shows an example simulation grid with four simulation instances (green, orange, blue and red) - the numbers represent pedestrians generated since the last time the metric was collected. These counts can then be collected by using a CUDA kernel that processes all of the environment agents placing the count in a multi dimensional array with each instance separated by row (Fig. 5.5(b)). By identifying the simulation id of the environment agent, the counts can be placed into the correct row. A parallel sum operation can be performed concurrently on each of these rows in order to obtain the final count specific to each simulation instance, as shown in Fig. 5.5(c). The last value of the column shows the total pedestrian generated count for each simulation instance.

5.3.2 Crowd Density and Flow

Areas of dangerous congestion can be visually identified with a density map. Fruin's LoS [276] is a widely used standard for measuring density as a comfort level. LoS metrics vary depending on whether a pedestrian is waiting or walking. The LoS metric has been described previously in Section 4.6 and Table 4.1. In the model, only the LoS level for walking pedestrians is used as there is no organised queuing.

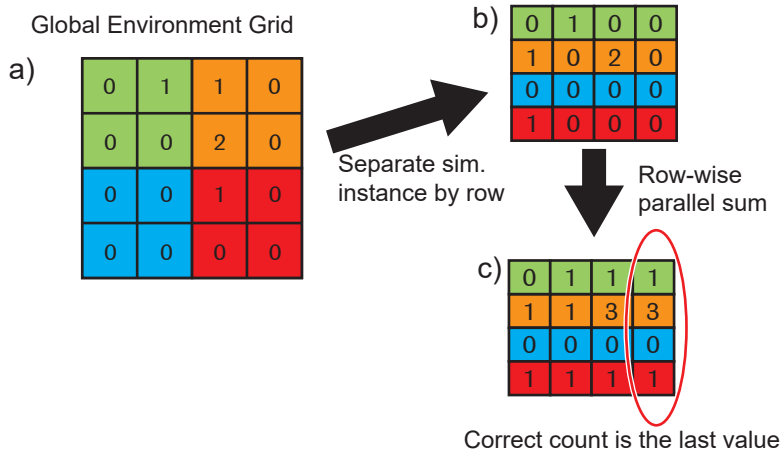

Figure 5.5 The colours green, orange, blue and red represent an individual simulation. (a) The grid of generated pedestrian counts. (b) These are separated by row for each simulation instance. (c) A row-wise parallel sum can then be performed and the counts can be obtained in the last column.

While LoS is collected from the Global Pedestrian List, it is required to be spatially relative to its local environment. The resolution of the LoS grid can be specified. Fig. 5.6(a) shows an LoS grid of 2x2 for each simulation instance and the pedestrians that are inside the grid. Each pedestrian agent has an LoS value which is obtained by dividing their local area by the number of pedestrians present within it as described in Section 4.6 (Fig. 5.6(b)). A CUDA kernel processes each pedestrian agent and collects the LoS and position, with respect to the global environment (as an array position), and a boolean counter value (COUNT) of 1 and writes it into a multi-dimensional array (Fig. 5.6(c)). The results are then sorted in parallel by their POS (Fig. 5.6(d)).

Another kernel is then used to find the boundary between each of the POS counts (SEG row). A segment starts and continues with a sequence of 0 until 1 is found and indicates that the next value is the start of the next segment. A single value segment starts with 1. The segmentation is needed to perform a segmented sum operation, only summing the values within a segment, on the LoS row and the COUNT row. The results are the sum of the LoS values and the number of pedestrians counted within for each grid point (Fig. 5.6(e)). The resulting LoS row and COUNT row can then be divided together in order to obtain the average LoS at each grid point. Further mapping to a local navigation grid can be performed by simply taking the POS row and calculating the reverse of Equation 5.3. The final result is shown in Fig. 5.6(f). Note that a cell that is left blank in Fig. 5.6(f) is set at a constant value much higher than LoS level A to signify that there are no pedestrians in that particular cell.

5.4 The Scenario: Evacuation After a Dirty Bomb Incident

There are many possible disastrous events that could happen within an urban area that warrant a mass evacuation. It can be from human errors (e.g. toxic spills), natural disasters such as flooding, earthquakes or terrorist activities such as bombing or releasing of toxic agents. The evacuation, depending on the location, could involve tens to hundreds of thousands of people. Further complications arise when toxic agents are involved and evacuated pedestrians have to be decontaminated. It is often impractical or impossible to shut down large sections of busy urban areas in order to perform real-life emergency drills so the use of simulations can provide a practical and cost-effective alternative.

The scenario presented in this section was implemented on suggestion of an industrial partner, BAE Systems, and their client, the fire services of Nottingham and Avon in the U.K., who also provided valuable information on the state of current approaches and feedback on the system and model. After discussions with the fire services, the evacuation after a dirty bomb incident inside a busy public building was deemed suitable as a scenario, where a simulation system to test a variety of response plans and compare their performance could be effective in helping to minimise casualties. This is due to the complex coordinated responses that must be organised in situations that can change rapidly and many decisions must be made that will have important consequences.

In a dirty bomb incident, the standard response is to establish cordon zones (Fig. 5.7) around the areas that have been effected. This is in order to manage, contain and prepare pedestrians for decontamination. The hot zone (red) represents the area where there is still danger, either from fire hazard or high risk of contamination. The warm zone (orange) is an area that is determined to not represent an immediate danger to the pedestrians but could still be contaminated. It is the zone where pedestrians will be told to wait in order to be processed. Physical barriers will often be erected in the warm zone in order dissuade breaches, encouraging pedestrians to stay within the zone to be processed, and reducing the spread of contamination. The cold zone (blue) is then defined as a safe area where there is no danger of contamination. However, since it is still an operational area of emergency services, it is not accessible to the general public.

Having established the zones, disrobing kits are then handed out. These are modesty kits that allows the pedestrians to remove their contaminated clothes. These kits can be distributed though fire services personnel or the establishment of disrobing points, where pallets containing disrobing kits are placed in strategic locations. Having

disrobed, the pedestrians can proceed to the decontamination tunnel that contains a shower to help remove the rest of the contaminated material from their body.

The focus is in the crucial first hour when the emergency services arrive at the scene and start establishing cordon zones, as it is the time that dangerous congestion will most likely occur. It is also important to make sure that pedestrians are as comfortable as possible in order to dissuade them from leaving the cordon areas to later present themselves at a hospital, spreading contamination along the way.

The simulation is used for planning the cordon areas so that it will comfortably fit all evacuees and prevent dangerous congestion. Decontamination tunnels are placed and verified that they fit within the desired areas. Placements of disrobing points, which is crucial in deciding where the crowd of pedestrians congregate, can be tested. The system is used to run the scenario with differing parameters and a variety of physical configurations in order to find the most favourable solutions.

5.5 The Agent Model

The model for the scenario consists of two types of agents. Pedestrian agents represent individuals evacuating from the disaster (Section 5.5.1). Environment agents are discrete grid-based navigation agents as used in Section 3.2.4 but each are also aware of the cordon zone it belongs in (Section 5.5.2). The process of cordoning pedestrians normally happens only when the entire cordon has or is nearly established. For this initial test scenario, in order to simplify the model, it was determined that the event of establishing the cordon is done instantaneously rather than simulating microscopic behaviours of the service members.

The model starts from the moment the first evacuee exits the affected building and ends when a mass decontamination structure is established, since afterwards the pedestrian density steadily decreases in a controlled way and so is less important from a safety perspective than immediate control. The sequence of behaviours in the model is explained further in Section 5.5.3.

5.5.1 Pedestrian Agent

Pedestrians agents in this model (Fig. 5.8) use the social force model with contact forces [22, 99] for local inter-pedestrian collision avoidance. A more detailed explanation of the social force model used and the agent functions can be found in Section 3.1.1. For this model, two additional parameters are used. The first indicates whether a pedestrian is injured which reduces the walk speed. The reduction is arbitrary set to 25% reduce in speed as the exact value could not be obtained at the time. The second parameter

is the self-presenting probability, the probability that a pedestrian will try to leave the environment even after the cordon has been established. Although the Avon fire services did not have the exact figure, from the meeting, it was mentioned that 5-10% is a good estimate. Hence, the self-presenting probability was set as 5% for the test case. Both of these additional parameters are set by the Environment agent at the time when a pedestrian is emitted. A velocity threshold is applied so at higher pedestrian density smaller time steps are used in order to avoid numerical instability. The model uses a maximum time step of 0.2 seconds.

5.5.2 Environment Agent

Environment agents encode global navigation data and can be considered as overlapping grids of vector fields, an approach previously explained in Section 3.1. The parameters of a single Environment agent can be found in Listing 5.1. The model makes use of three vector fields in total. A Collision Vector Field (CVF) contains information necessary for avoiding static obstacles (Fig. 5.9(a)) in variables `collision_x`, `collision_y`, and `collision_d`. Only two Navigation Vector Field (NVF)s are needed as pedestrians only have two goals, either to evacuate from the mall to the nearest exit or to stay in the cordon area and wait to be decontaminated. The first NVF is the evacuation map which is used to guide the pedestrians to the exit nearest to their location (Fig. 5.9(b)). This uses the variables `evac_x` and `evac_y`. The second NVF is the disrobing route map which is used in directing pedestrians to their nearest disrobing point (Fig. 5.9(c)). This uses the variables `dcon_x` and `decon_y`. Each Environment agent stores extra information, which are the entrance markers, disrobing point markers, exit markers, zone markers, the probability that a generated pedestrian will be injured(`prob_injured`) and the probability that the pedestrian will always seek to exit the environment (`prob_force_exit`). Entrance markers (`entrance`), shown as green squares in Fig. 5.9(d), signify that the agent belongs to an entrance with a particular id and pedestrians are emitted at configurable rates per entrance. The disrobing point markers (`disrobe`), which are shown as orange squares in Fig. 5.9(d), allows the pedestrian to check whether they have reached and collected a disrobing kit. The exit markers (`exit`) are shown as red squares in Fig. 5.9(b). When a pedestrian reaches an Environment agent with `exit > 0`, they exit the simulation. Pedestrians that exit the environment cannot return. The cordon zone marker (`zone`) shown in Fig. 5.9(e) stores the zone of that particular environment cell. The red, orange and blue squares represent hot, warm and cold zones, respectively.

```
environment_agent:agent{
    x:float32
    y:float32
    prob_injured:float32
    prob_force_exit:float32
    exit:int
    entrance:int
    zone:int
    disrobe:int
    collision_x:float32
    collision_y:float32
    collision_d:float32
    evac_x:float32
    evac_y:float32
    decon_x:float32
    decon_y:float32
}
```

Listing 5.1 Pseudocode of a single Enviroment agent's parameters.

5.5.3 Model Behaviour

The model has a set sequence of behaviours:

- When the model begins, pedestrians rush from inside the building (Fig. 5.10(a)). Occupancy of the building is determined before the simulation starts.
- Pedestrians will try to take the shortest route available to escape from the incident area (Fig. 5.10(b)). A percentage of pedestrians are injured and therefore have a slower average walking speed.
- After a certain time period, the emergency services reach the scene and start establishing cordons (Fig. 5.10(c)). In the model, the entire cordon is established instantaneously. In a real situation the emergency services are likely to wait until sufficient numbers of the team are in place.
- Pedestrians that are outside of the cordon at the time it is established will carry on heading to the exit (Fig. 5.10(d)).
- Pedestrians that are still within the cold zone will then be given instructions to go back and wait in the warm zone. A configurable percentage of these people will not follow instructions and will leave the scene right away. The ones that

stay behind are required to walk to a disrobing point where they will be given disrobing kits (Fig. 5.10(e)). In practice, a pedestrian simply switches from using the evacuation map (Fig. 5.9(b)) to the disrobing map (Fig. 5.9(c)).

- Once the disrobing kit is obtained, they will then wait within the area until the decontamination tunnels are established and the simulation finishes.

The switching of evacuation map to disrobing map is done by using a global variable `CORDON_ESTABLISHED` which at the start of the simulation is set to `false`. After a certain simulation time, this variable is set to `true` and the pedestrians use the information obtained from the Environment agents and their internal parameters to decide whether to stay and get decontaminated or continue to exit the environment.

5.6 Collection of Zone and Disrobing Metric

A metric specific to the dirty bomb evacuation scenario (Section 5.4) is collected in addition to the general metrics discussed in Section 5.3. The zone and disrobing metric gives the pedestrian counts within each cordon zone and whether or not they have collected the disrobing kits.

5.6.1 Zone and Disrobing Metric

There are two elements to the zone and disrobing metric: the zone count and disrobing count. For the zone count, the pedestrian counts in each of the zones are collected in order to allow the observation of the general flow though each of the cordons. The disrobing count represents a count of how many pedestrians have collected the disrobing kits.

Pedestrian agents store the zone that they're currently in and also whether they have reached the disrobing point. Fig. 5.11(a) shows an example list of pedestrians with different statuses, where the letters H, W, C and D refer to hot zone, warm zone, cold zone and disrobed, respectively. The four collected metrics, number in hot zone, warm zone, cold zone and how many have reached the disrobing point can be collected in parallel into a multi-dimensional array where each status of each simulation is separated by row. The value 1 represents a count of pedestrian and 0 means there's no pedestrian. In the top row of Fig. 5.11(b), it can be seen that there is a single pedestrian in the hot zone in the green simulation instance. A kernel processes through all pedestrian agents and places a count of one in the correct row of the metric. Row-wise parallel sum can then be performed on the resulting array to find the counts for each metric. The result of the above example is shown in Fig. 5.11(c) where the

last column shows the final counts for each metric separated by simulation. In the example it can be seen that for the green simulation instance there is 1 in the hot zone, 2 in the warm zone, 1 in the cold zone and 1 that has collected a disrobing kit. Having collected these metrics, other information can be found such as when the last person left the hot zone or how long it takes for everyone to collect the disrobing kit.

5.7 Results

This section outlines the benchmarks that are used for testing the system. The Nottingham shopping mall environment (Fig. 5.12) will be used as an example to illustrate the behaviour. Pedestrians enter the simulation from the entrances, shown as green lines, and can exit the simulation at the exits, shown by the red lines. The simulation always follows the scenario where a dirty bomb has exploded inside a building causing pedestrians to evacuate (Section 5.4). The simulation starts at the point where the first pedestrian exits the building. Pedestrians then start evacuating to the nearest exits. A screenshot of the beginning of the evacuation is shown in Fig. 5.22. After 10 minutes, a cordon is established around the perimeter of the warm zone and a disrobing point becomes available (Figs. 5.13 and 5.14). At this point, any pedestrian already in the cold zone will continue to head to the exit. Pedestrians in the warm or hot zone have a 5% chance to continue to exit the environment (as discussed in Section 5.5.1), otherwise they will stay in the warm zone and pick up a disrobing kit.

Section 5.7.1 presents the 3 environments. For each environment, there are 2 configurations for the cordon zone and disrobing area that is used to test the flexibility of the system. Two benchmarks were conducted to measure the performance of the system. The first benchmark measures the simulation speed relative to real time when the simulation is running the scenario with Evacuation plan 1 (Fig. 5.14). The second is a multi-GPU performance benchmark that was carried out to show the effect of running multiple Simulator instances on the same machine.

5.7.1 Environments

In order to ensure that the simulator is able to work in a range of environments, three different city centre locations were used for testing. The first is a shopping mall in the centre Nottingham (Figs. 5.13 and 5.14), the second is Sheffield town hall (Figs. 5.15 and 5.16) and the third is Sheffield city hall (Figs. 5.17 and 5.18). Each of the environment has two different configurations in the placement of decontamination tunnel, warm zone and disrobing point.

Simulations are run using these various environments. Screenshots of pedestrians gathering at dirobing points for the Nottingham shopping mall evacuation plan 1 and 2 are shown in Figs. 5.24 and 5.26, respectively. All simulation ends at 45 minutes when the decontamination tunnel is established. Fig. 5.19 shows the Nottingham shopping mall environment plan 1 (left) and plan 2 (right) being run concurrently. Fig. 5.20 shows all four Sheffield environments running concurrently.

5.7.2 Model performance

For this test, Nottingham shopping mall plan 1 is used (Fig. 5.13). A total of 1200 pedestrians enter the environment from the four entrances of the mall, and the number is distributed evenly between the four entrances (Fig. 5.22).

The use of a social force model for local collision detection means a dynamic time step must be used in order to avoid numerical instability. Because of this, simulation performance can vary greatly according to the densities and forces experienced by the pedestrians. The benchmark measures the performance of the simulation taking this factor into account. A maximum time step of 0.2 seconds was used. The velocity threshold, a limit on how much pedestrians can change velocity in a single time step, was changed between each run. They take the values of 0.02, 0.05 and 0.1, as shown in Fig. 5.21(a-c), respectively. The increase in velocity threshold is akin to increasing the time step size allowing pedestrians to move more within a certain amount of time. In turn, this is are likely to produce more collisions. Teknomo [281] found that acceleration of pedestrians walking across a zebra crossing has a mean of $0.68ms^{-2}$ with a range of $1.39ms^{-2}$. As for how these figures compared to the velocity threshold, the threshold of 0.1 for $0.2s$ time step equates to a maximum acceleration of $0.5ms^{-2}$ before the time step is reduced. The threshold of 0.02 and 0.05 then equates to $0.1ms^{-2}$ and $0.25ms^{-2}$ limits, respectively. This means the threshold value of 0.02 and 0.05 is a conservative limit being only at 15% and 37% of the observed mean acceleration value. For the graphs in Fig. 5.21, iteration count forms the x axis. The left y axis represents the real simulation speed dt/rdt calculated as the dynamic simulation time step (dt) divided by the real time needed to process that iteration (rdt). Values higher than 1 (represented by the red line) means that the simulation is running faster than real-time. The dt and rdt values are an average taken every 100 steps. Finally the right y axis (blue) shows the number of pedestrians currently in the environment.

Until the cordon is established, it can be seen from the graphs that value of dt/rdt fluctuates wildly depending on the maximum force that pedestrians experience. The dt/rdt then slightly declines as the behaviours change and pedestrians stop trying to exit the environment and start to congregate in the warm zone near the disrobing

point. After all pedestrians have been emitted and the simulation stabilisess, the speed can then be seen to increase rapidly due to the fact that pedestrians are able to spread out into available spaces. The velocity threshold has a drastic effect on the real simulation speed. For a 45 minute simulation, the simulation with threshold value of 0.02 (Fig. 5.21(a)) took approximately 29 minutes to complete. Simulations with threshold values of 0.05 (Fig. 5.21(b)) and 0.1 (Fig. 5.21(c)) completed in 13.8 and 7.2 minutes, respectively. This means a conservative threshold value of 0.05 would result in the the ability to perform simulations at 3 times the speed of real-time.

The benchmarking was performed on a 3.4GHz Intel Core i7 machine with 8GB of RAM using the Nvidia GTX590 GPU. Visualisation on the Simulator was turned off for benchmarking as would be done in a real situation where the user will only see the recorded metrics.

5.7.3 Performance on multiple GPUs

In order to evaluate the overheads in running multiple instance of the Simulator on the same machine, performance when running the Simulators on multiple GPUs was measured. A blank environment was created and gradually filled with pedestrians. Each Simulator was launched as a separate executable and each occupies a single GPU. A batch of four simulations were run on each Simulator simultaneously. An iteration time value, the time to run a single simulation step, was recorded for every 1,000 pedestrians on the GPU starting from 1,000 and ending at 100,000 pedestrians. When enough pedestrians were generated to start measurement, the emission was stopped. The simulation was left to stabilise for at least 2 seconds of simulation time. The iteration time was averaged over the next 100 iterations and recorded. The benchmark was run first for one instance of the Simulator (using one GPU). It is then performed again using two instances of the Simulator (two GPUs), effectively doubling the number of pedestrians simulated. The benchmarking was performed on a 3.5GHz Intel Core i7 machines with 16GB of RAM equipped with two Nvidia GTX Titan GPU.

The results are plotted on the graph in Fig. 5.27, which shows iteration time (ms) over pedestrian count. The result when running with two GPUs over a single one is a mean increase in iteration time of 1.61% with standard deviation of 2.33%. It shows that there is only a minor difference in the iteration time for running on a one and two GPUs although the number of concurrent simulations being run is effectively doubled. This is an expected result as the Simulators are essentially independent from each other and there is no inter-communication needed between the two GPUs.

The Simulator essentially runs one large simulation with each instance of a scheduled simulation taking up part of the environment space. This means the maximum number of concurrent simulations that can be run on each Simulator will depend on the total environment size and number of agents being simulated at the same time. In this multi-simulation system, the biggest bottleneck would be the CPU cycles needed for scheduling the simulator. CPU to GPU communication are minimal, large data transfers such as uploading the Environment agent data or collecting the LoS metric is done only once per simulation instance. As the Simulators are independent of each other, on a multi-core machine where there are at least the same number of cores as the number of GPU, the system is expected to be able to scale accordingly.

5.8 Discussion

This chapter has presented a prototype system that can be used for running multiple simulation instances in parallel by spreading the simulation workload across a network of computers equipped with GPUs. For each GPU, batching is performed by merging multiple simulation instances into a single large simulation. Metrics are collected using parallel approaches in order to maximise efficiency. An initial trial was conducted using four machines, with the same specification as the ones used to perform benchmarking in Section 5.7.2, connected over a local network. The Simulation Manager was sent a random batch of evacuation plans (Figs. 5.13 to 5.18) to fill up all available simulation slots. This showed that the system can schedule simulation across multiple machines, run simulations concurrently and have no trouble with collection of metrics from the nodes used for Simulation. The prototype system was shown to successfully run multiple simulations efficiently. However, the project was concluded before an empirical study can be made to verify the results of the model.

In an interview with the Avon FRS Planning Specialist after the project ended, it was stated that the model appears to behave correctly according to their experience but there are still many features needed to make it useful to planners. First, they would need a user-interface where, with little training, the planners could use it by themselves to plan a wider range of scenarios. Also, other contaminants or dangers needed to be taken into account, e.g. if it's a chemical leak then containing it would be an additional priority but may not need decontamination. Additional environment conditions are also needed, such as terrain elevation, wind patterns and other factors that can affect the safety of the holding location, e.g. ensure there's no nearby tall glazed building.

As discussed at the end of Chapter 4, Concoursia is currently lacking in concurrent multi-simulation capabilities. This chapter has shown how this functionality could be

implemented. Concoursia was designed to be split into two parts, analogous to the GUI Client and Simulator as described in this chapter. The Client side would need additional features for sending batch jobs and viewing metrics of multiple simulation instances. The Simulator side would need the simulation batching feature as described in Section 5.2 and metrics collection as described in Section 5.3. When managing multiple simulation instances in the Simulator, it would be simplest to store navigation graphs and obstacle graphs separately for each running simulation instance and provide agents with a simulation id for differentiating the simulations they are currently in. The Simulator manager and the Local Simulation manager could then be used with relatively little modification for scheduling simulation runs.

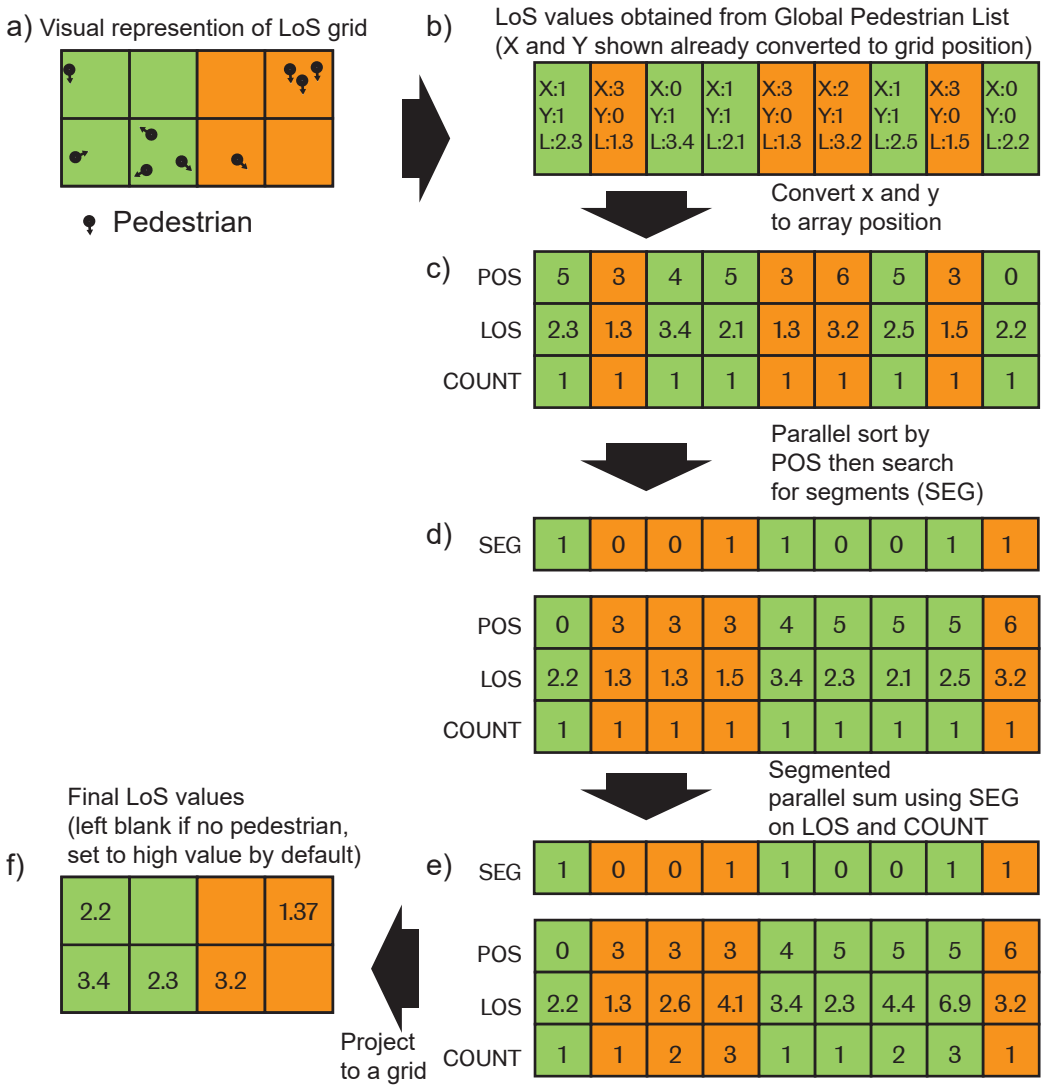

Figure 5.6 Parallel generation of LoS grid for multiple simulation instances. Each simulation instance is represented by a different colour. Local simulation space is spatially divided into a custom LoS grid of 2x2 in this example.

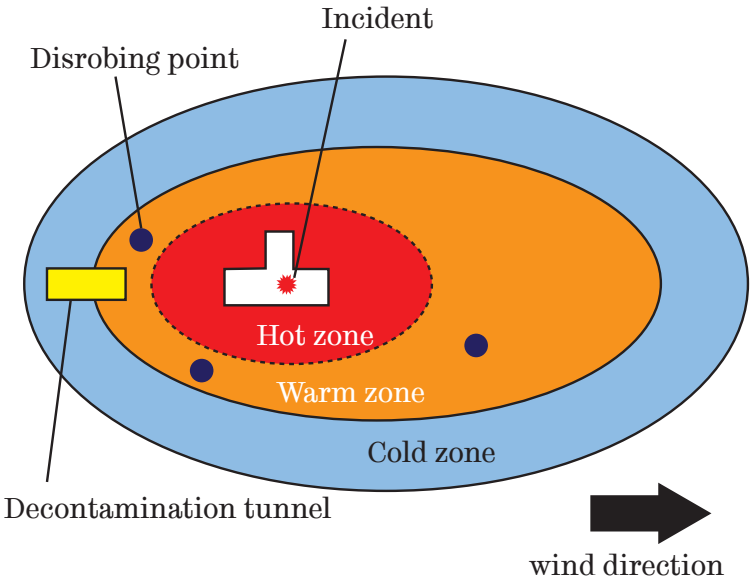

Figure 5.7 The different cordon zones that are set up in order to deal with mass decontamination incidents. The size and layout of these zones can change due to the nature of the threat and environmental factors such as wind direction. The hot zone (red) is an area at high risk of contamination or with active dangers such as fires. The warm zone (orange) is an area where there are no immediate danger but it could also be contaminated. Disrobing points (purple) contain kits to enable pedestrians to remove contaminated clothes modestly. The cold zone (blue) is a safe, non-contaminated area.

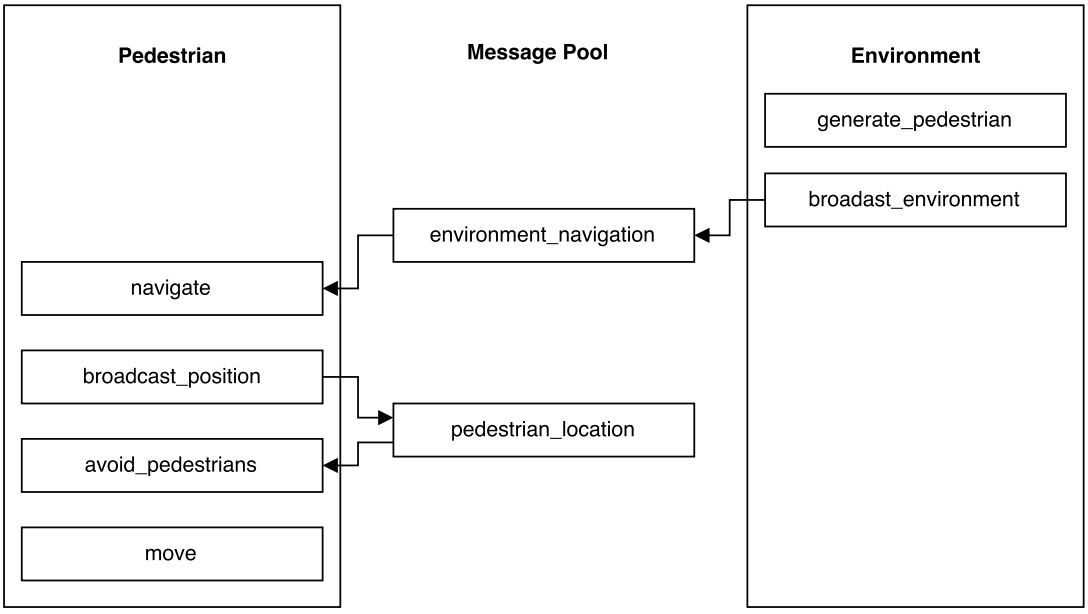

Figure 5.8 The pedestrian agent and environment agent functions used in the model.

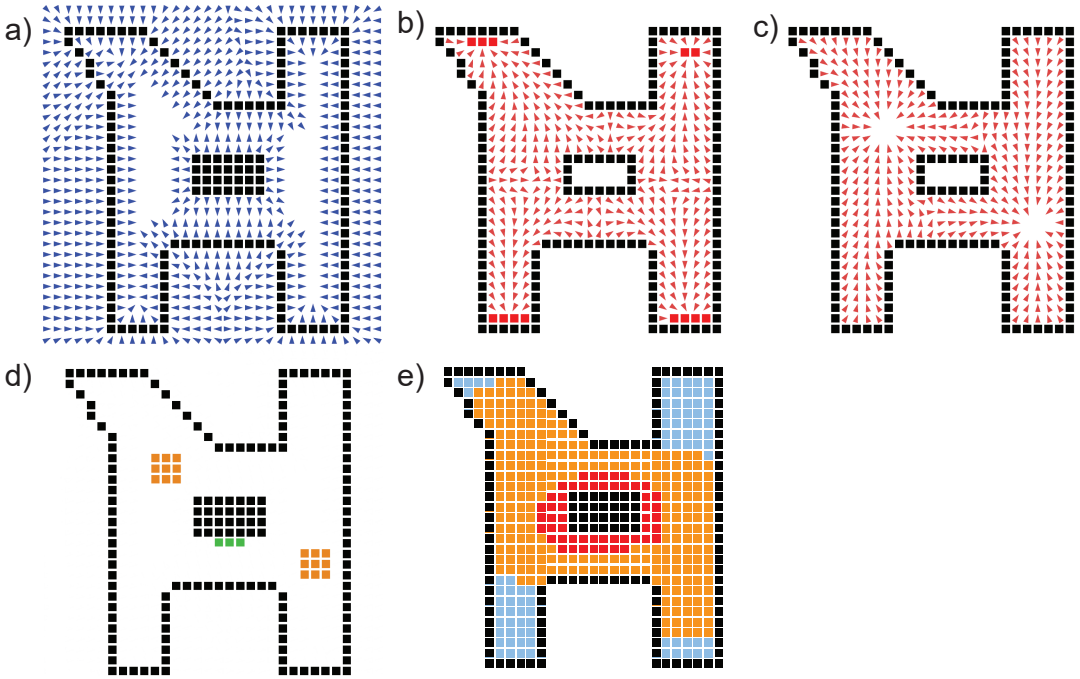

Figure 5.9 This figure shows the different types of maps used to represent the environment. (a) The CVF map is used in avoidance of static obstacles. (b) The exit NVF map guides the pedestrians to the nearest exit in the environment. (c) The disrobing NVF map guides the pedestrian to the nearest disrobing points. (d) Other information such as emitter (green blocks) , disrobing points (orange blocks). (e) The hot, warm and cold cordon zones are represented in as red, orange and blue blocks respectively.

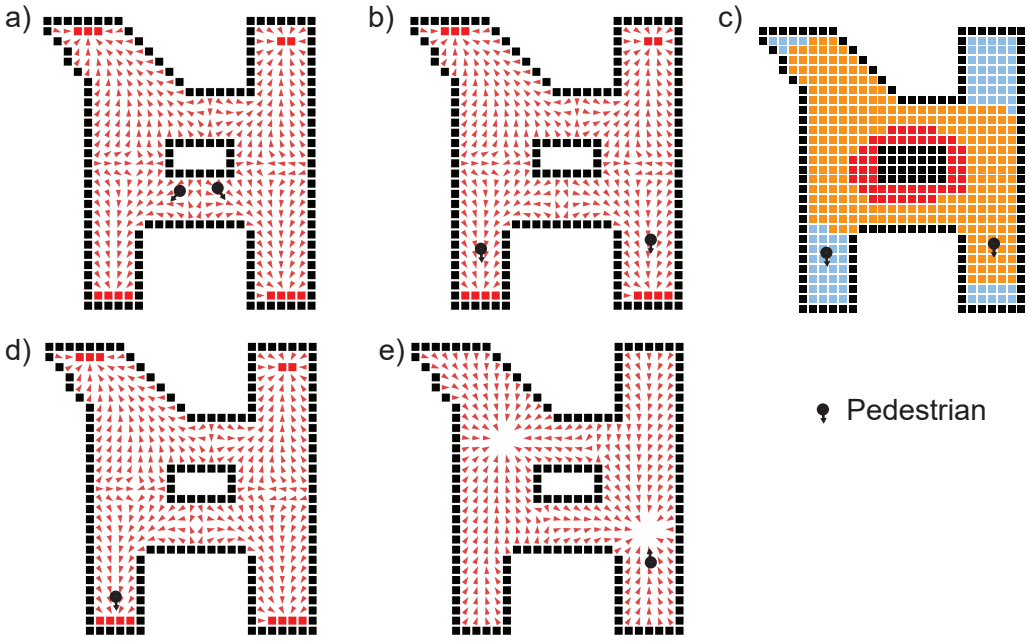

Figure 5.10 (a) Pedestrians enters the environment. (b) They then follow the evacuation map. (c) After a certain time, cordon zones are established, red is hot, orange is warm and blue is cold. (d) After the cordon has been established, pedestrian outside the warm zone carries on walking to the exit. (e) Pedestrians inside the warm or hot zones walks to the nearest disrobing point.

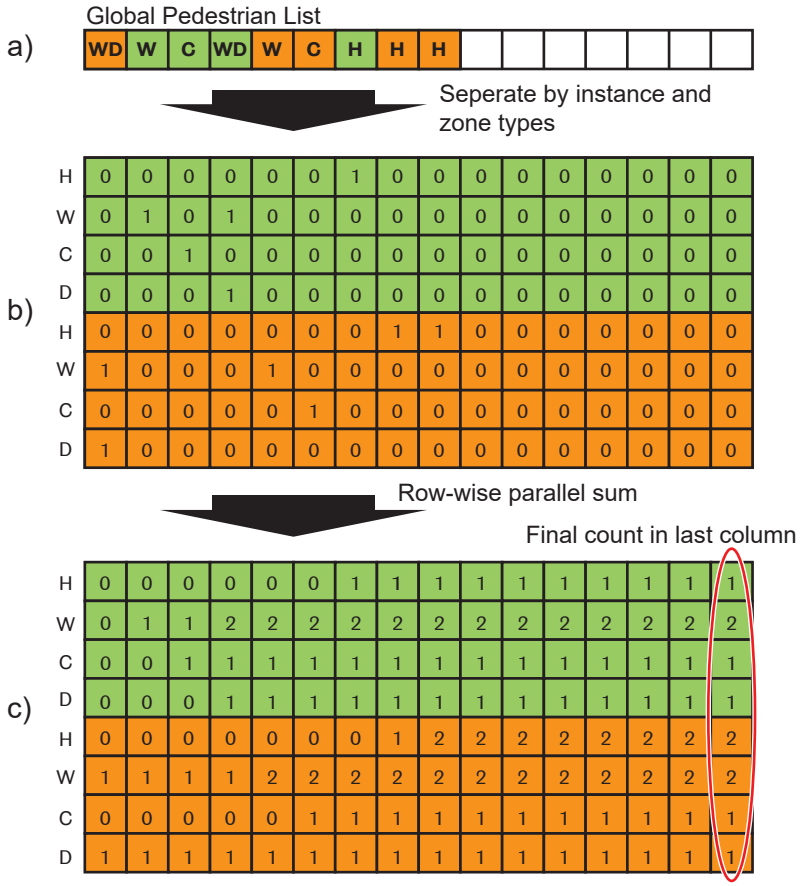

Figure 5.11 The colours orange and green represents separate simulations. Pedestrians keep track of the zones that they're currently in and whether they have obtained a disrobing kit. (a) An example of this pedestrian list whereby H, W, C and D refers to hot zone, warm zone, cold zone and disrobed respectively. (b) All of these properties are divided into separate arrays. Different simulations are also separated. (c) A parallel sum can be performed on all of these arrays in order to obtain the final counts.

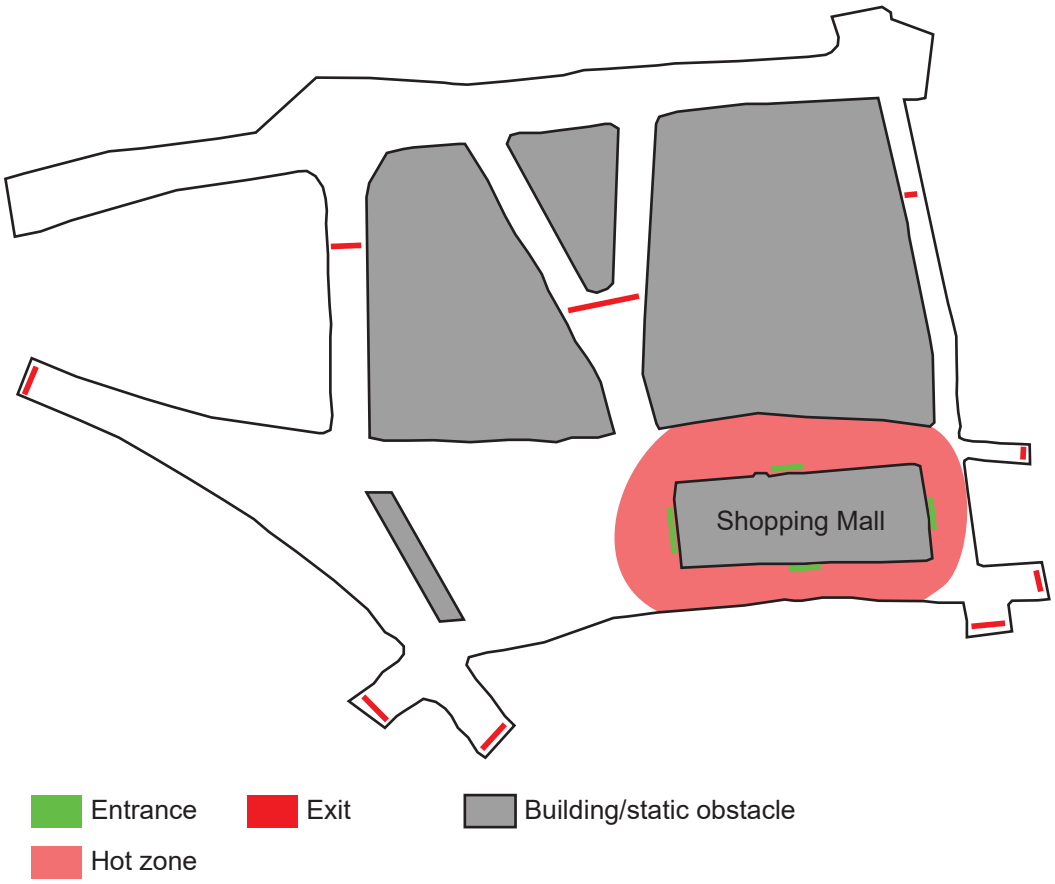

Figure 5.12 Nottingham shopping mall environment used for testing the multi-simulation system.

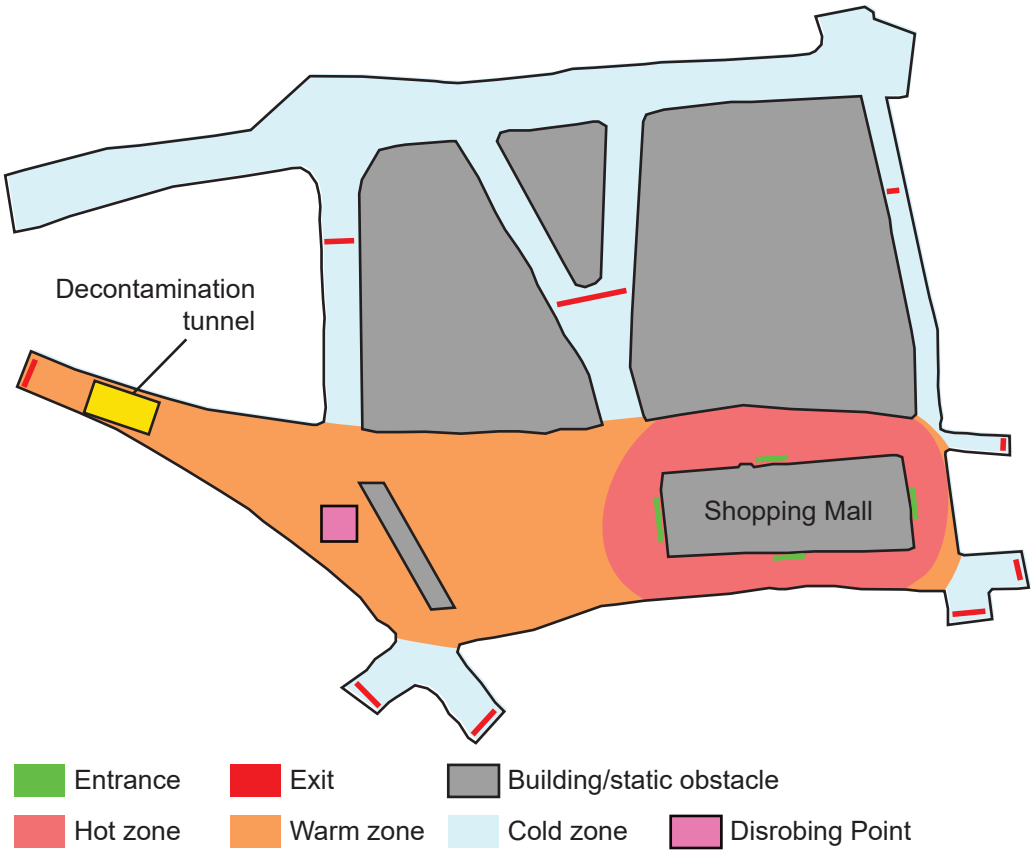

Figure 5.13 Nottingham shopping mall evacuation plan 1 - the decontamination tunnel is placed at the left most exit, with a disrobing point on the left side of the large obstacle left of the mall.

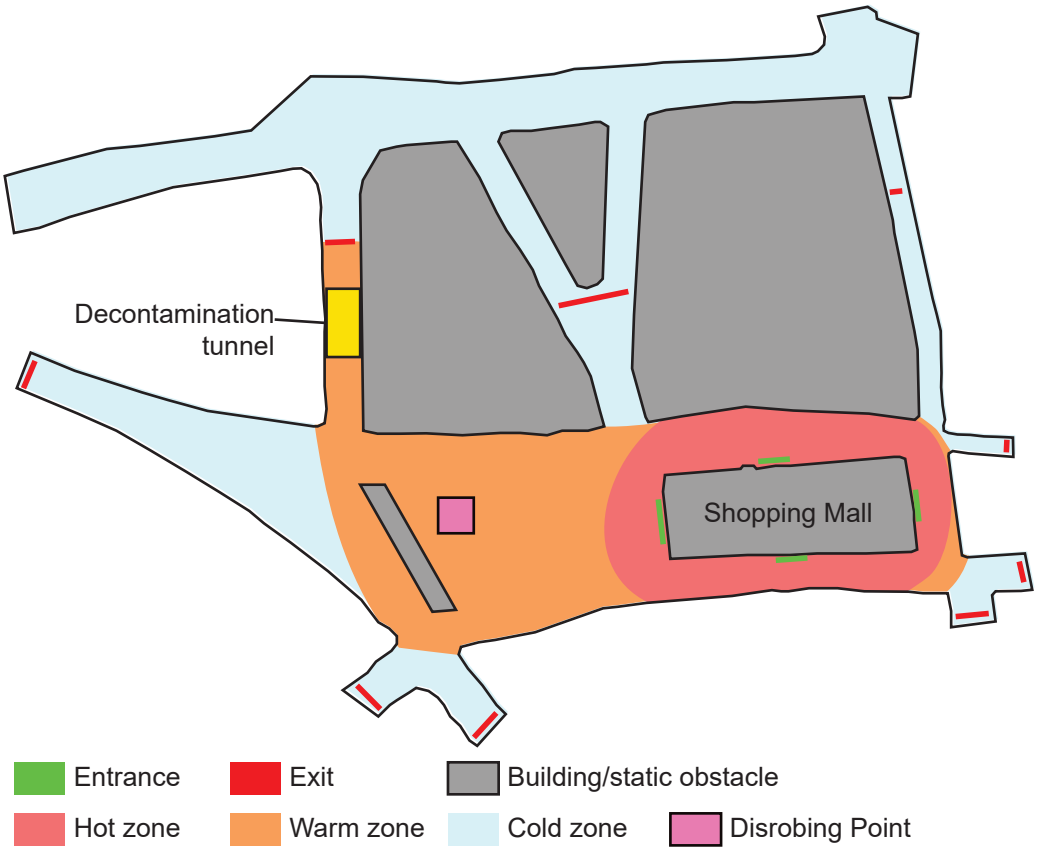

Figure 5.14 Nottingham shopping mall evacuation plan 2 - the decontamination tunnel is placed at one of the top exits, with a disrobing point on the left side of the Shopping Mall before the large obstacle.

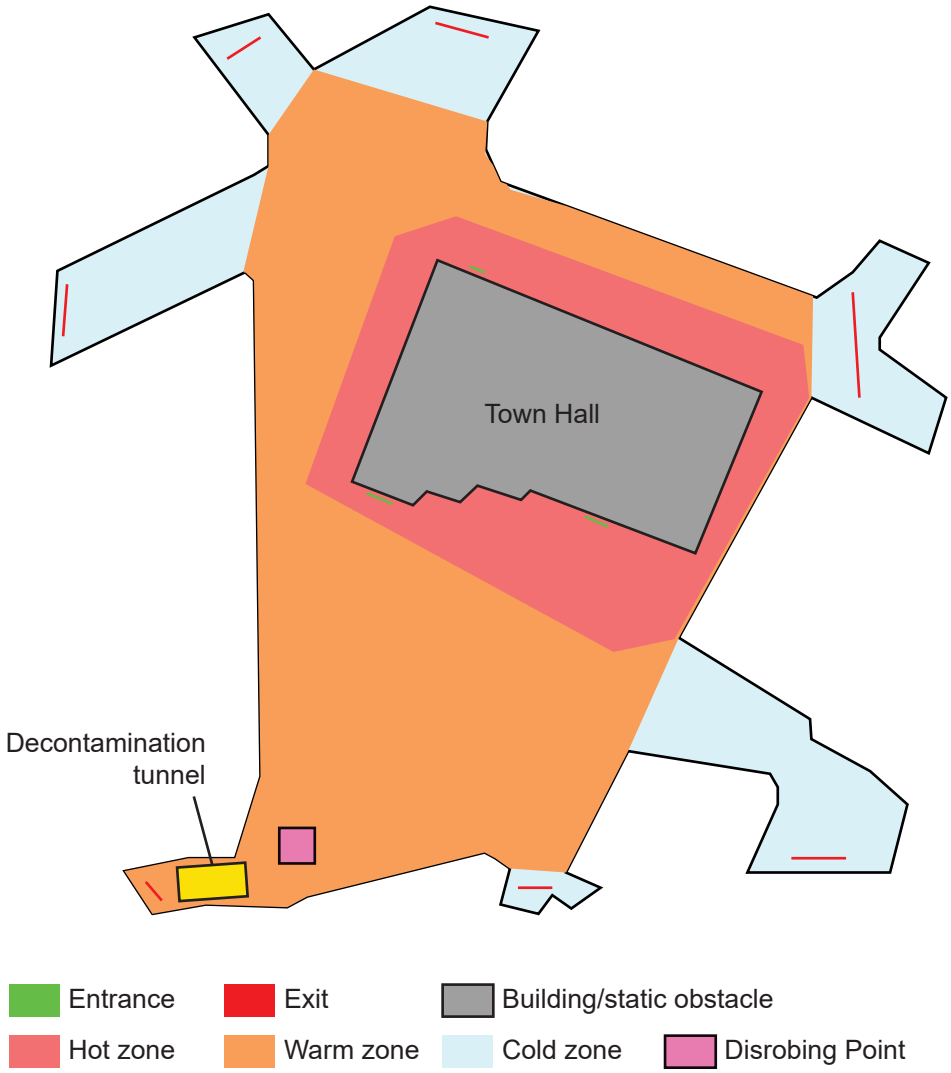

Figure 5.15 Sheffield town hall evacuation plan 1 - the decontamination tunnel is placed at the bottom left exit.

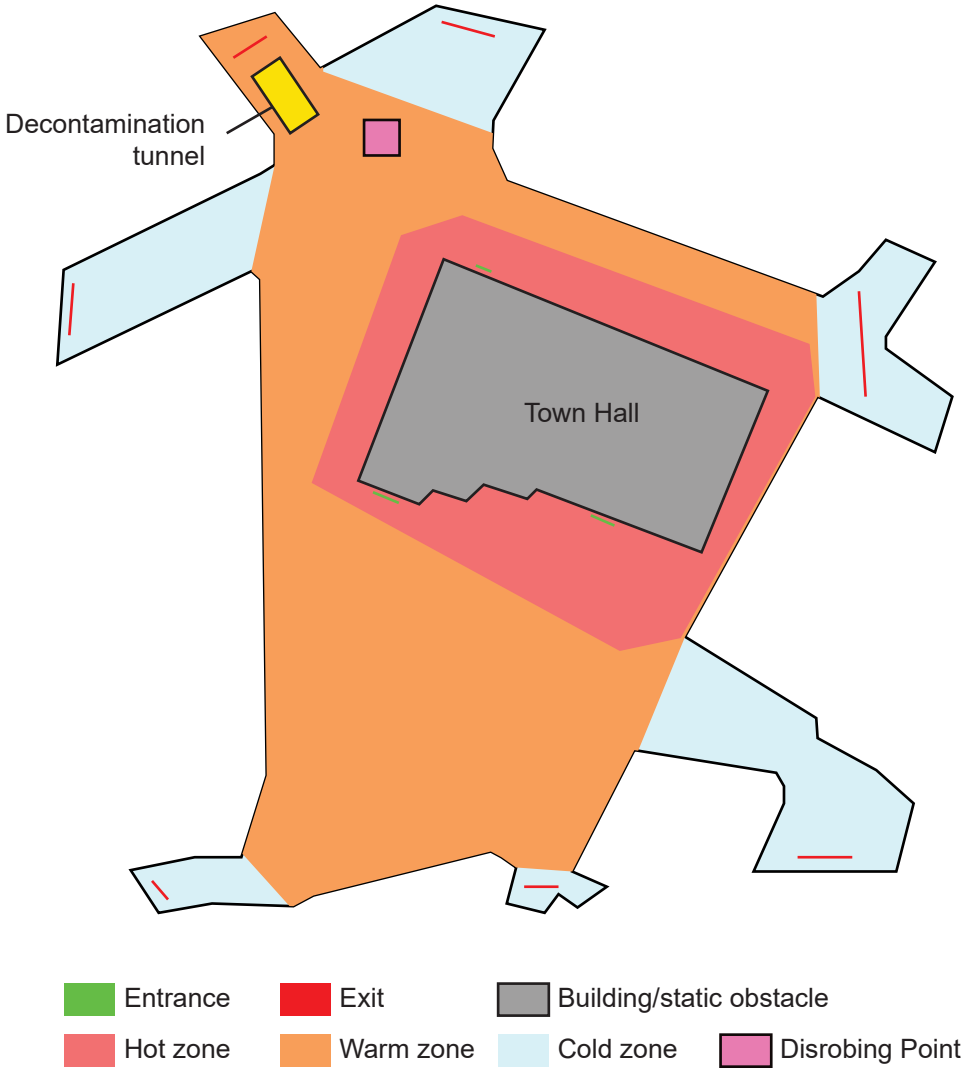

Figure 5.16 Sheffield town hall evacuation plan 2 - the decontamination tunnel is placed at the top left exit.

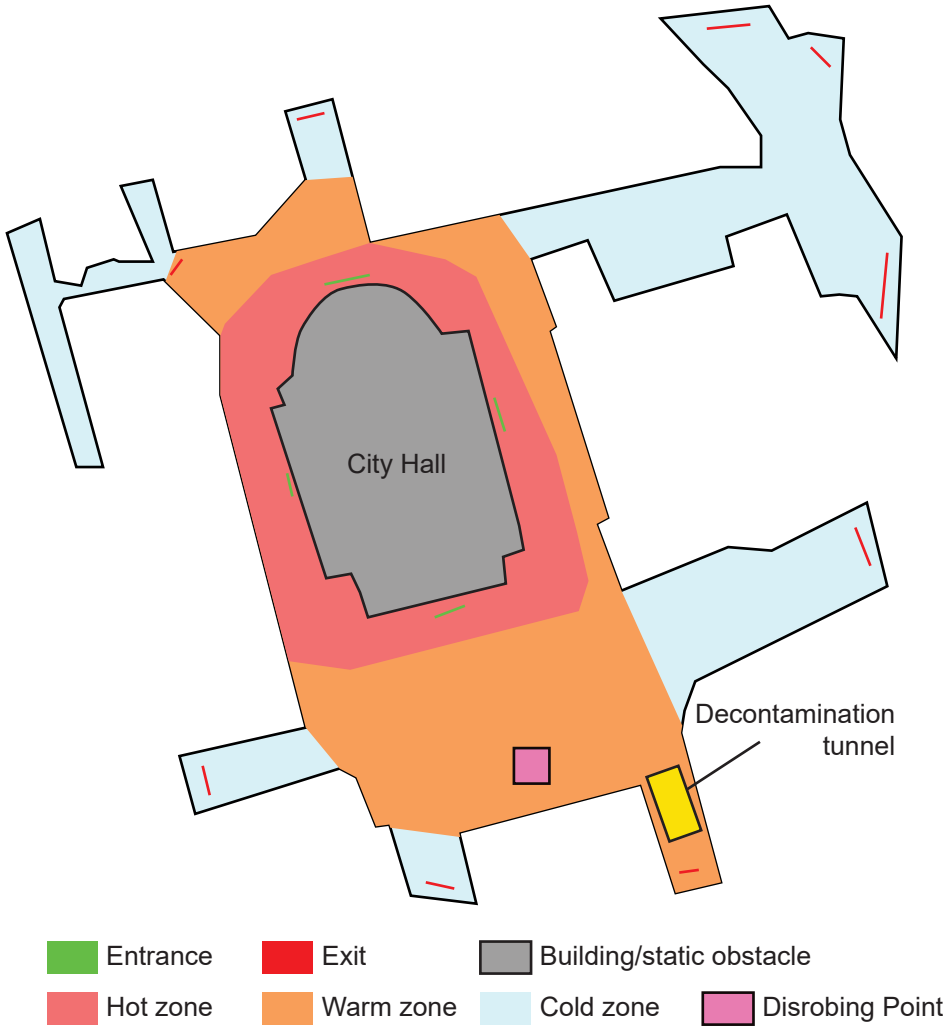

Figure 5.17 Sheffield city hall evacuation plan 1 - the decontamination tunnel is placed at the bottom right exit.

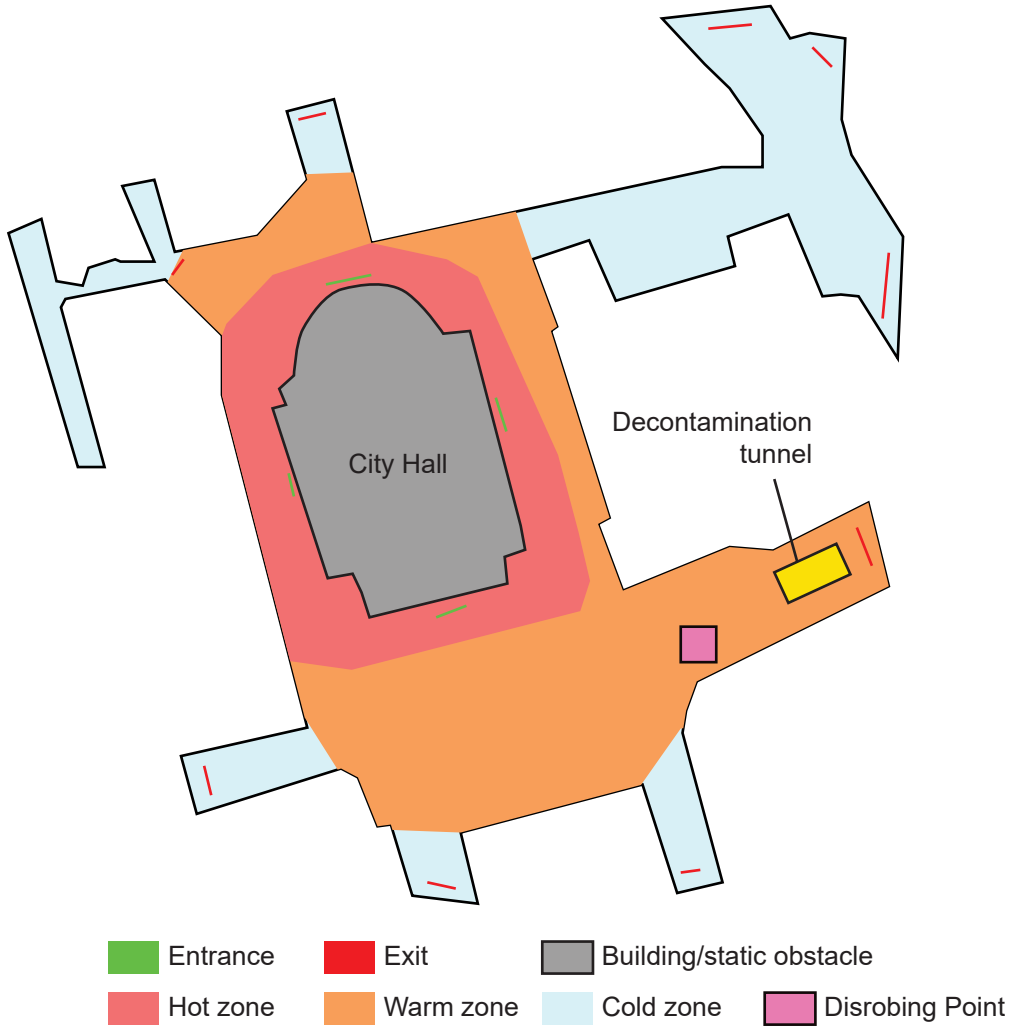

Figure 5.18 Sheffield city hall evacuation plan 2 - the decontamination tunnel is placed at a lower right exit.

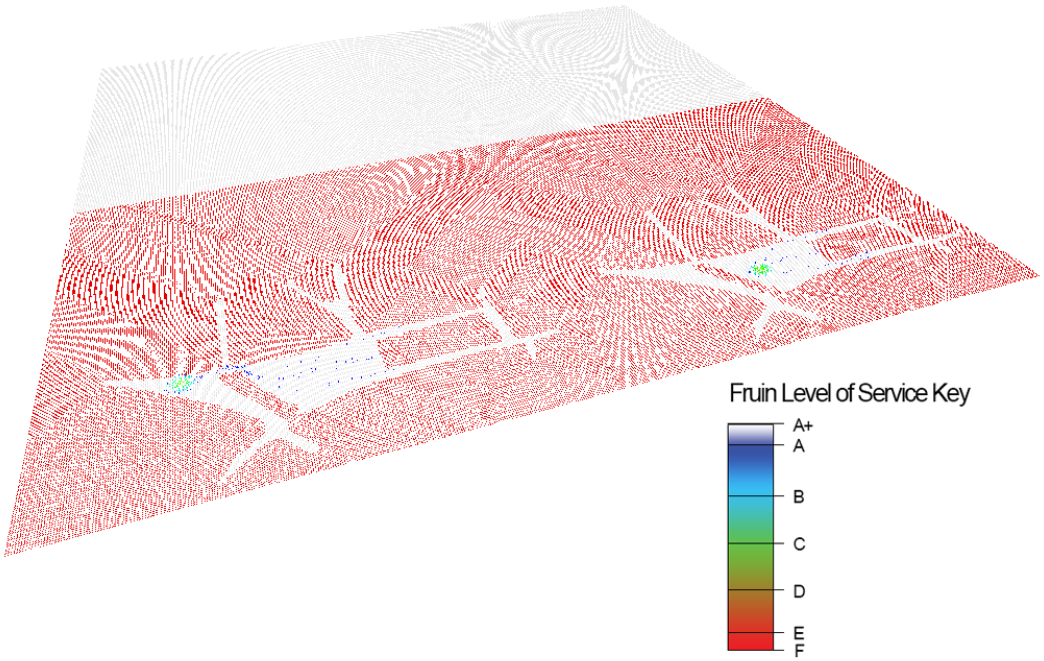

Figure 5.19 Two concurrent running simulations. The scene shows pedestrians gathering at the disrobing point. Nottingham shopping mall environment plan 1 (Fig. 5.13) is shown on the left and plan 2 (Fig. 5.14) is shown on the right.

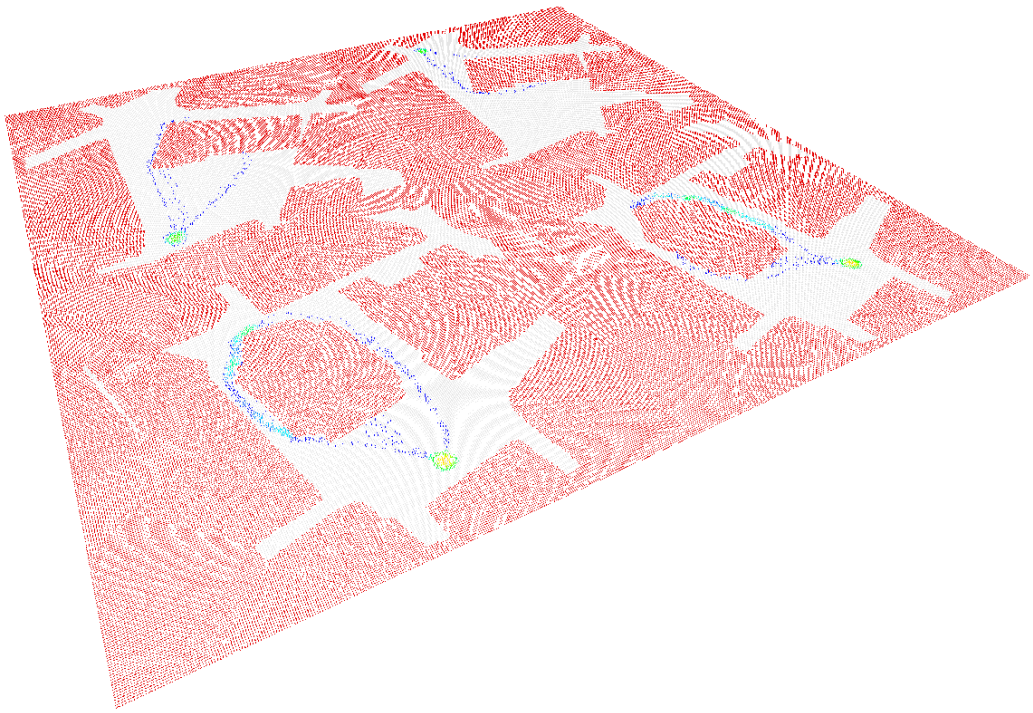

Figure 5.20 Four concurrent running simulations. The scene shows pedestrians gathering at the disrobing point. The Sheffield town hall plan 1 (Fig. 5.15) is shown running at the top left, and plan 2 (Fig. 5.16) is running at the top right. The Sheffield city hall plan 1 (Fig. 5.17) is shown running at the bottom left and plan 2 (Fig. 5.18) at the bottom right.

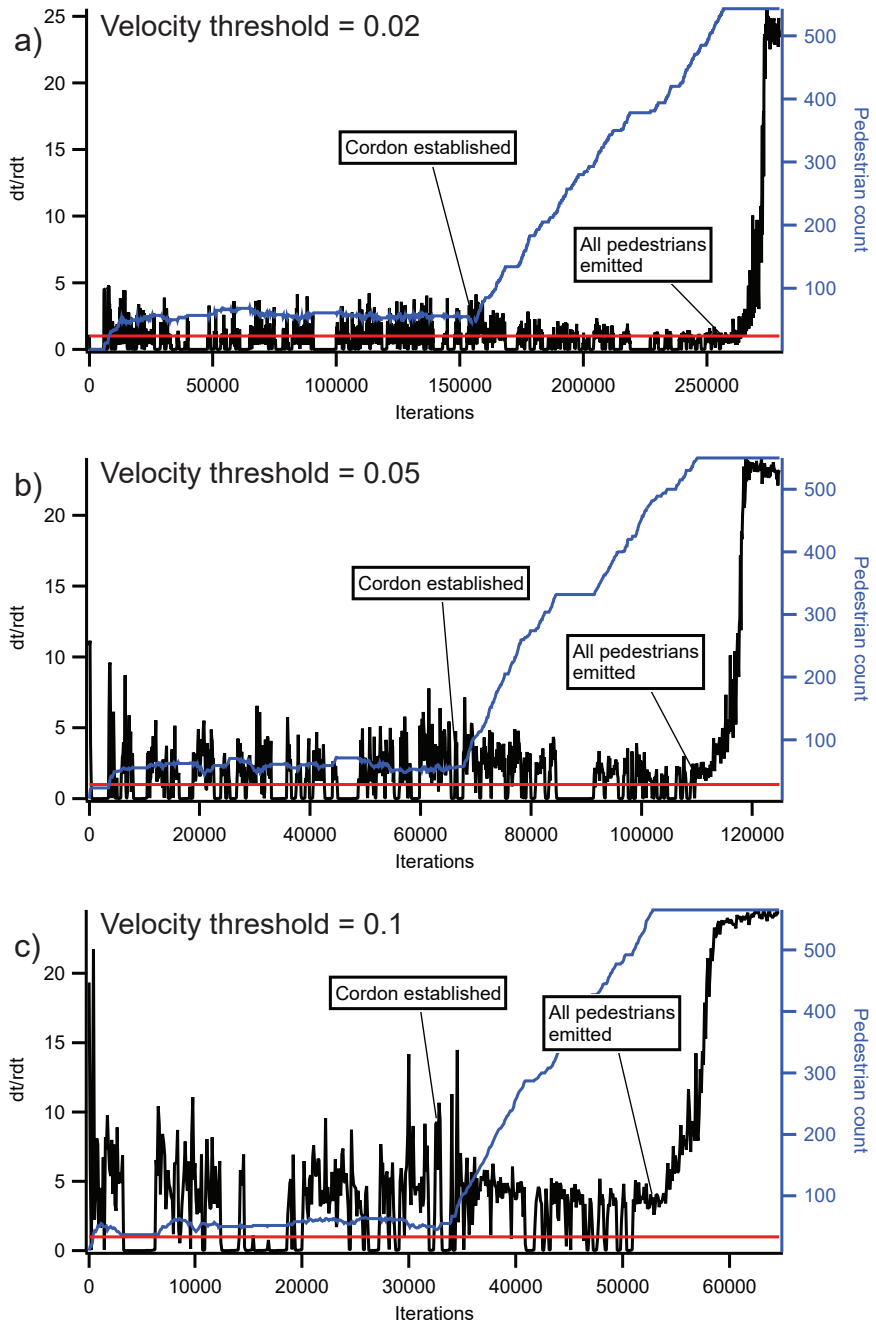

Figure 5.21 The three graphs shows how the real simulation speed (*dt*/*rdt*) changes throughout a simulation in normal conditions. The value *dt* stands for the dynamic time step and *rdt* stands for the amount of real time taken to process that step of the simulation. Both these values are averaged over 100 steps. Real simulation speed values over 1 (above the red line) means it is running faster than real time. The line in blue, along with the right axis, shows the number of pedestrians currently in the simulation.

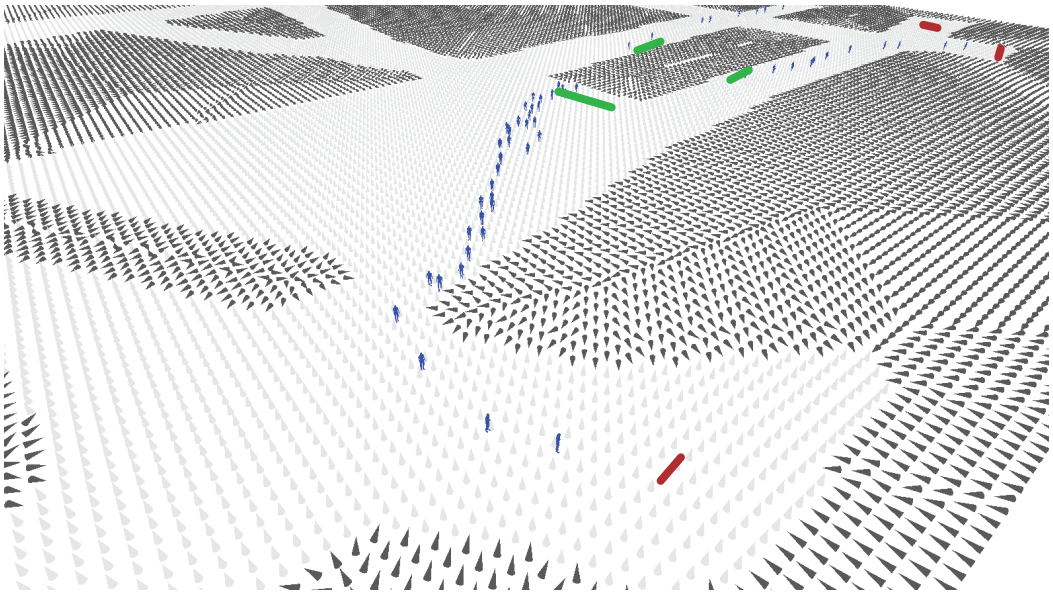

Figure 5.22 Nottingham shopping mall - start of evacuation. Pedestrians are emitted from the shopping mall (green lines) and walks to the nearest exit (red lines).

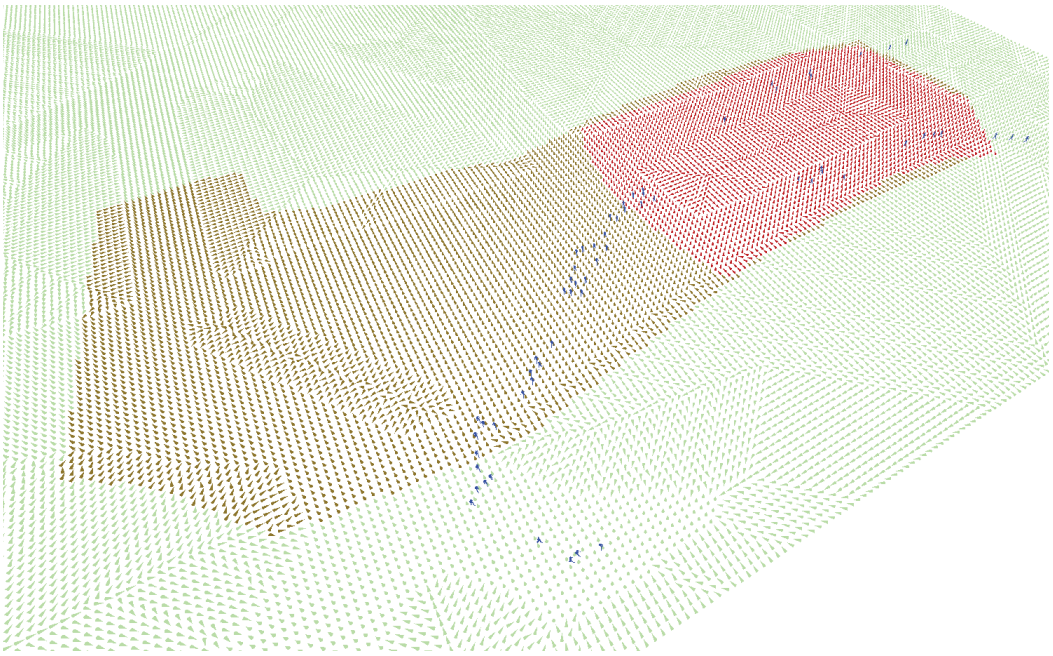

Figure 5.23 Nottingham shopping mall plan 2 - zones as visualised in the simulator. Green is the cold zone, orange is the warm zone and red is the hot zone.

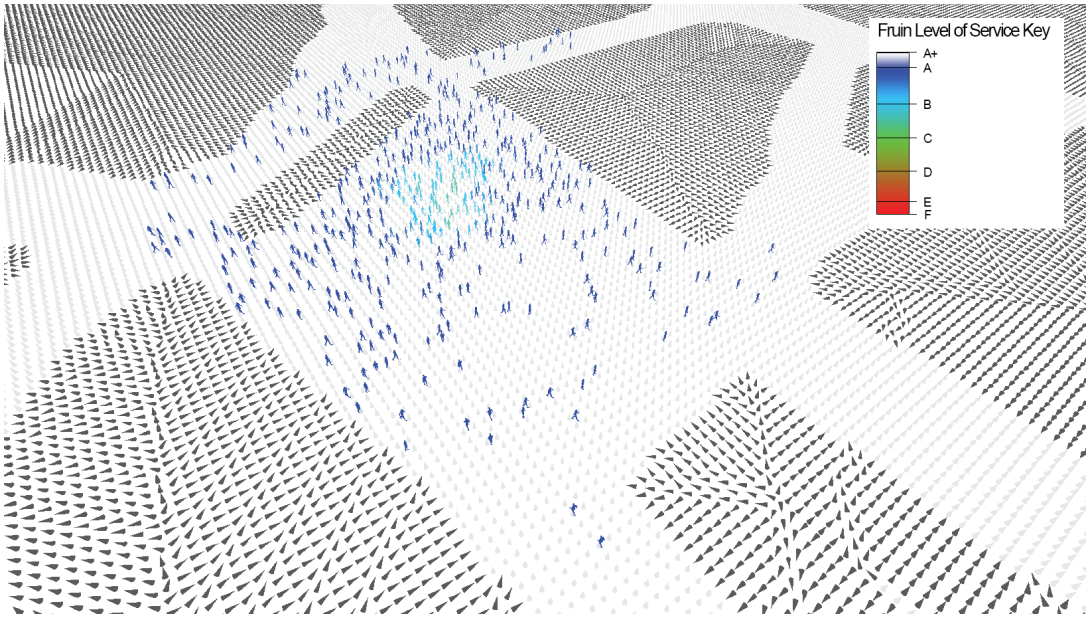

Figure 5.24 Nottingham shopping mall evacuation plan 1 - pedestrians gathering at the disrobing point. Pedestrians change colour according to their LoS value.

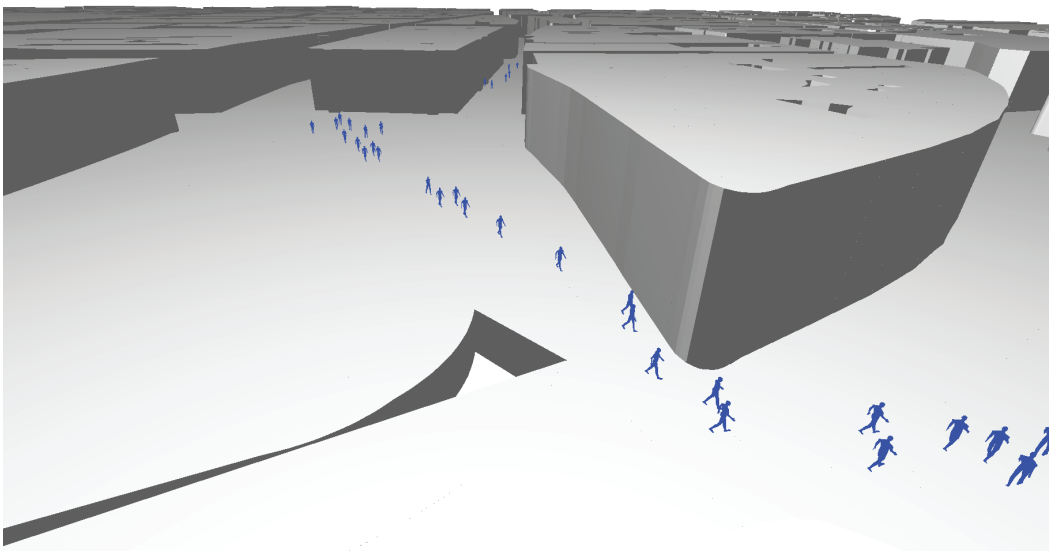

Figure 5.25 Nottingham shopping mall - simulation visualised with 3D plan of the town centre extracted from GIS data.

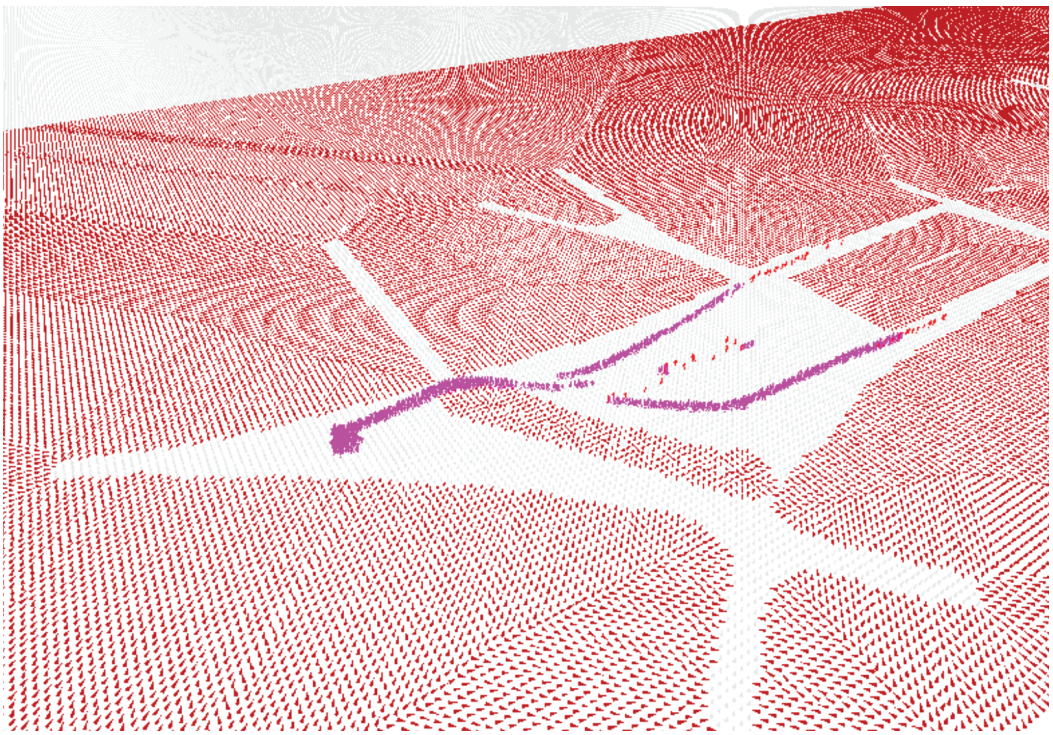

Figure 5.26 Nottingham shopping mall evacuation plan 2 - pedestrians gathering at the disrobing point. The red pedestrians are injured and walk slower. The pink pedestrians are not injured.

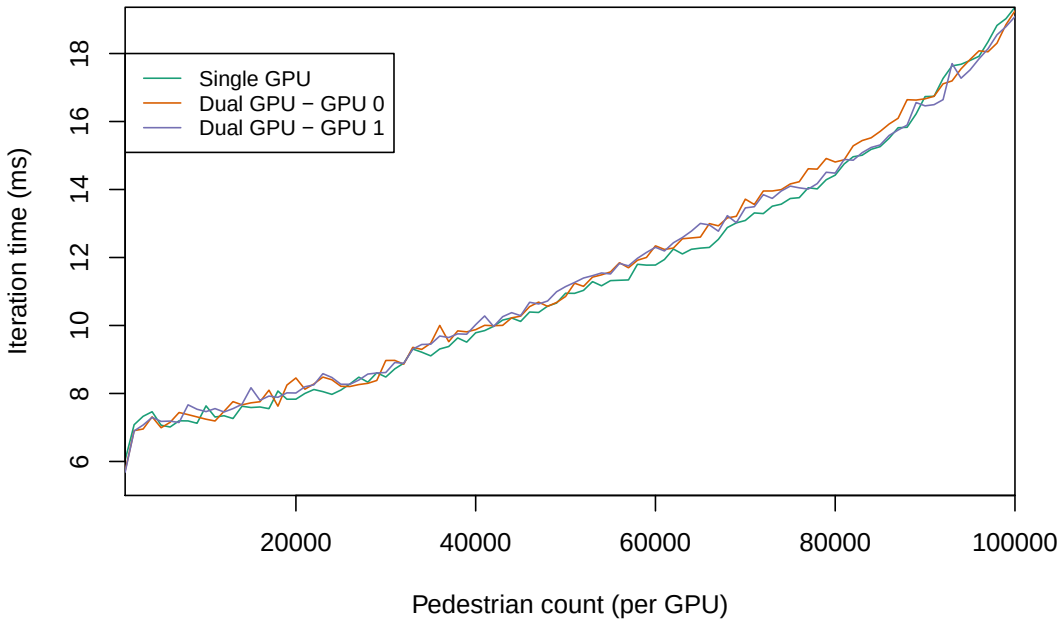

Figure 5.27 Comparison of running the Simulator running the Nottingham shopping mall plan 1 over a single GPU and two GPUs. Each GPU is running 4 instances of the scenario concurrently.

Chapter 6

Conclusions

This thesis has investigated the use of GPUs for the creation of pedestrian simulation systems based on the Agent-based Modelling (ABM) approach. Through the implementation of two pedestrian simulation systems – Concoursia (Chapter 4) and a multi-simulation system (Chapter 5) – the architectural complexities of using a GPU have been examined. In addition, two alternative navigation systems have been developed and investigated for use in GPU-based pedestrian simulation systems: a grid-based approach and a graph-based approach (both described in Chapter 3), with the graph-based approach being used in Concoursia.

The work in this thesis has demonstrated that GPUs can be used to create a fast, efficient and scalable pedestrian simulation system. Chapter 4 illustrated this by showing that Concoursia, a pedestrian simulation system built for the GPU, is able to run simulations that have large, complex, and multi-level environments. This was illustrated with a Train station scenario that exhibited complex pedestrian behaviours such as queueing and following multi-stage objectives, which could be run at speeds faster than real-time. However there are complexities that have to be considered when using a GPU which could impact on speed, efficiency and scalability. One of the main issues to deal with is memory access, which can mean that algorithms must be reengineered. In considering memory access patterns on the GPU, data structures should be laid out following a Structure of Arrays (SoA) pattern in order to increase cache hits on memory reads, thus lowering the overall number of reads. Also, when the SoA pattern is followed and there are complex inter-relationships between the data structures, using indices rather than memory pointers to link the structures together makes it simpler to retain these links when transferring the data structures from CPU to GPU. Another memory issue that potentially can affect a GPU-based ABM system is that it is inefficient to dynamically allocate global memory from code running on the device. Considerations must be made for algorithms that rely on data structures that,

on the CPU, are designed to dynamically shrink and grow (e.g. vectors, queues, and stacks). One method that can be used to resolve this is to alter the algorithms so that they always use a fixed amount of pre-allocated global memory. The work presented in Chapter 3 demonstrated the use of the fixed memory method by designing the graph-based navigation system to always require a fixed number of variables to keep track of navigation progress. The traversal of the R-tree for searching the Environment graph is also done using a stackless approach ([272]). In addition, the RVO algorithm in Chapter 4 uses a fixed-length array instead of a dynamic array in order to track obstacles.

Individual aspects of a complete pedestrian simulation system must also be re-considered for the GPU. Individual pedestrian behaviour in an ABM approach is suited to the thread-based multicore GPU, and can be handled using a framework such as FLAME GPU, which can also deal with local social force models. The other aspect that must be considered is global navigation which is important for complex behaviour. Chapter 3 of the thesis focussed on alternative pedestrian global navigation approaches, a grid-based approach and a graph-based approach, and how these could be used with FLAME GPU. The grid-based approach addressed an issue in other grid-based approaches where pedestrians are agents, but navigation is separate. Instead, grid-based approach in this thesis embraced the agent-based paradigm and modelled the environment as static agents that encode navigation and the pedestrian agents as dynamic. This means the grid-based approach could be integrated into any general ABM framework and could take advantage of existing platform optimisations. However, the approach is limited to scenarios where the environment size and number of objectives are small due to its large memory requirements. Therefore, it fails on scalability. The second approach considered, a searchable graph-based approach, is scalable. For two large scenarios, a shopping mall and a train station, it used 100 times less memory than the grid-based approach. In contrast to other graph-based approaches, it also provides dynamic re-planning during congestion. These advantages meant it was the choice for use in Concoursia. However, the implementation of the graph-based approach on the GPU faced the memory issues that had to be resolved as discussed above.

Another aspect to consider is how to deal with running many variations of a simulation, as might be required when exploring alternative decisions in a complex scenario. To address this, Chapter 5 demonstrated a prototype pedestrian multi-simulation tool that is able to run multiple independent simulations concurrently on machines with a single GPU, with multiple GPUs or on GPUs across a network. For each GPU, the tool merges multiple independent simulations and runs them as if they were one single large simulation. This requires careful memory management both

for running the simulations and also for gathering metrics for each simulation. For example, efficient parallel operations were used for separating the metrics back in to the coordinates of individual simulations. When running on a machine with two GPUs, the amount of simulation runs was effectively doubled with a negligible decrease in performance (1.61% on average). It is expected that this result will scale up to machines equipped with more GPUs, as there is essentially no inter-communication between the GPUs and minimal communication between CPU and GPU during the simulation run. In addition, as each network node runs simulations that are independent of each other, the only network communication required is for scheduling the simulation and retrieving the metrics. The system is thus expected to be able to scale up to a large number of nodes although further testing is required.

6.1 Limitations and Future Work

This thesis has shown that the GPU can be used to create fast, efficient and scalable pedestrian simulation systems that have the potential to solve the computability problem posed by a Decision Support System (DSS). However, the work presented in the thesis is only a part of a much larger system necessary for the creation of a truly integrated real-time DSS for managing pedestrian flows.

In the short term, many improvements can be made to the current system. The integration of Concoursia (Chapter 4) with the multi-simulation system (Chapter 5) would form a core processing platform that could be used when an optimiser is integrated on top of the system. It would be necessary to test how the system scales up when running on hundreds or thousands of nodes. Although no problems are expected when actually running the model itself, issues with management of the Simulators, or data storage of collected metrics may arise. In addition, other features and behaviours such as elevators, re-routing at blocked exits, social grouping and decision making could be added. The research to add complex cognitive behaviours to the pedestrian model on the GPU will continue to face the complexities due to the parallel nature of GPU's architecture. Recent advances such as unified memory can automatically manage the synchronisation of memory between the CPU and GPU and allow models to be larger than the GPU memory. As the memory transfer process is still being performed in the background, and now as an opaque process, researchers may find the process more difficult to optimise for. Existing techniques for optimising memory access such as keeping data structures coalesced (e.g. following the Structure of Arrays (SoA) pattern) or the allocation of memory will remain relevant for the foreseeable future. Once the pedestrian model is finalised, it is also important to verify that the

model is working and calibrated correctly. As a start, the software could be subjected to the NIST validation and verification test [103].

Medium term objectives would involve further research into the optimisation of the simulation. This is the other crucial part of a DSS. Ultimately the system should be able to suggest a number of plans for solving a problem and provide cost-benefit analysis associated with carrying out each one. A standard set of metrics could be established for a wide variety of cases. The system would require a systematic way for automatically ranking pedestrian simulations, and be able to combine and rank different type of metrics depending on the initial objective for running the analysis. It may be that exit time is the most crucial for evacuation but footfall or utilisation in certain parts of the environment is more relevant to retail. After the system is able to evaluate simulation results, it must then be able to suggest plans for optimising the scenario to achieve the desired result. These optimisations are numerous, can be domain-specific, and will vary in scope and complexity. They could involve things such as deciding to rope off an area, closing an exit, attempting to pre-warn people at the entrance or even the re-design of the building or facility. Each of these optimisations and suggestions must then be validated according to their own guidelines.

Longer term objectives are associated with the implementation of a robust large-scale automated pedestrian tracking system which still poses a significant challenge that requires further research. It can be used to validate and calibrate the pedestrian model as well as providing historical records to assist in future predictions. Although there exist a variety of technologies and tools as discussed in Section 2.4.1, none of them currently present a truly robust and cost-effective solution. New algorithms and hardware will need to be developed. It may require the development of new sensor technology or the combination of existing ones. Just the process itself will require many years of development and testing until it is ready to be deployed. The tracking of pedestrians also raises privacy implications and if the system is not inherently anonymous, data security, data and privacy laws, and even dealing with public opinion poses additional challenges to the project.

References

[1] Y. Cao, Q. Duan, and N. Zhang, "Optimized environment designing of nanjing south railway station based on pedestrian simulation," *Cross-Cultural Design*, Jan. 2016. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-40093-8_50

[2] R. Lohner, M. Baqui, E. Haug, and B. Muhamad, "Real-time micro-modelling of a million pedestrians," *Engineering Computations*, vol. 33, no. 1, pp. 217–237, 2016. [Online]. Available: http://dx.doi.org/10.1108/EC-02-2015-0036

[3] E. Galea, G. Sharp, P. Lawrence, and R. Holden, "Approximating the evacuation of the world trade center north tower using computer simulation," *Journal of Fire Protection Engineering*, vol. 18, no. 2, pp. 85–115, 2008. [Online]. Available: http://dx.doi.org/10.1177/1042391507079343

[4] A. Veeraswamy, E. R. Galea, L. Filippidis, P. J. Lawrence, and R. J. Gazzard, "The simulation of urban-scale evacuation scenarios: Swinley forest fire," in *Human Behaviour in Fire, Proceedings 6th Int Symposium*, 2015.

[5] N. Zhu, J. Wang, and J. Shi, "Application of pedestrian simulation in olympic games," *Journal of Transportation Systems Engineering and Information Technology*, vol. 8, no. 6, pp. 85 – 90, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570667209600076

[6] A. L. Lu, B. G. Ren, C. W. Wang, and D. C.-Y. Chan, "Application of sfca pedestrian simulation model to the signalized crosswalk width design," *Transportation Research Part A: Policy and Practice*, vol. 80, pp. 76 – 89, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0965856415002050

[7] Y. Zhou, J. Wang, D. Huang, and S. Sun, "Pedestrian simulation modeling for world expo 2010 shanghai," *Journal of Transportation Systems Engineering and Information Technology*, vol. 9, no. 2, pp. 141 – 146, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570667208600604

[8] Y. Chen, Y. Cai, P. Li, and G. Zhang, "Study on evacuation evaluation in subway fire based on pedestrian simulation technology," p. 9, 2015. [Online]. Available: http://dx.doi.org/10.1155/2015/357945

[9] T. Schelhorn, D. O'Sullivan, M. Haklay, and M. Thurstain-Goodwin, "Streets: An agent-based pedestrian model," 1999.

[10] M. Lyell and M. Becker, "Simulation of cognitive pedestrian agents: crowds in crisis situations," *Systemics, Cybernetics and Informatics*, vol. 4, no. 3, pp. 79–84, 2005.

[11] R. Thomas and S. Donikian, "A spatial cognitive map and a human-like memory model dedicated to pedestrian navigation in virtual urban environments," in *International Conference on Spatial Cognition*. Springer, 2006, pp. 421–438.

[12] J. M. Allbeck, "Carosa: A tool for authoring npcs," in *International Conference on Motion in Games*. Springer, 2010, pp. 182–193.

[13] N. Fridman and G. A. Kaminka, "Modeling pedestrian crowd behavior based on a cognitive model of social comparison theory," *Computational and Mathematical Organization Theory*, vol. 16, no. 4, pp. 348–372, 2010.

[14] E. Andresen, D. Haensel, M. Chraibi, and A. Seyfried, "Wayfinding and cognitive maps for pedestrian models," in *Traffic and Granular Flow'15*. Springer, 2016, pp. 249–256.

[15] S. R. Musse and D. Thalmann, "A Model of Human Crowd Behavior: Group Inter-Relationship and Collision Detection Analysis," in *Workshop Computer Animation and Simulation of Eurographics*, 1997, pp. 39–52.

[16] N. Pelechano, K. O'Brien, B. Silverman, and N. Badler, "Crowd Simulation Incorporating Agent Psychological Models, Roles and Communication," 2005.

[17] N. Pelechano and N. I. Badler, "Modeling crowd and trained leader behavior during building evacuation," *IEEE computer graphics and applications*, vol. 26, no. 6, 2006.

[18] L. Luo, S. Zhou, W. Cai, M. Y. H. Low, F. Tian, Y. Wang, X. Xiao, and D. Chen, "Agent-based human behavior modeling for crowd simulation," *Computer Animation and Virtual Worlds*, vol. 19, no. 3-4, pp. 271–281, 2008.

[19] M. Asano, T. Iryo, and M. Kuwahara, "Microscopic pedestrian simulation model combined with a tactical model for route choice behaviour," *Transportation Research Part C: Emerging Technologies*, vol. 18, no. 6, pp. 842–855, 2010.

[20] W. G. Van Toll, A. F. Cook, and R. Geraerts, "Real-time density-based crowd simulation," *Computer Animation and Virtual Worlds*, vol. 23, no. 1, pp. 59–69, 2012.

[21] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Physical Review E*, vol. 51, no. 5, pp. 4282–4286, May 1995.

[22] D. Helbing, I. Farkas, and T. Vicsek, "Simulating dynamical features of escape panic," *Nature*, vol. 407, no. 6803, pp. 487–490, 28 Sep. 2000.

[23] C. Burstedde, K. Klauck, A. Schadschneider, and J. Zittartz, "Simulation of pedestrian dynamics using a two-dimensional cellular automaton," *Physica A: Statistical Mechanics and its Applications*, vol. 295, no. 3–4, pp. 507–525, 15 Jun. 2001.

[24] R.-Y. Guo and H.-J. Huang, "A modified floor field cellular automata model for pedestrian evacuation simulation," vol. 41, no. 38, p. 385104, 22 Aug. 2008.

[25] T. Korhonen, S. Hostikka, S. Heliövaara, and H. Ehtamo, "Fds+evac: An agent based fire evacuation model," in *Pedestrian and Evacuation Dynamics 2008*, W. W. F. Klingsch, C. Rogsch, A. Schadschneider, and M. Schreckenberg, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 109–120.

[26] "Legion website," http://www.legion.com, accessed: 2017-03-06.

[27] M. Harmon and J. Joseph, "Evacuation planning tool (ept) for emergency, event and space planning," in *Pedestrian and Evacuation Dynamics*. Springer, 2011, pp. 785–788.

[28] G. Baglietto and D. R. Parisi, "Continuous-space automaton model for pedestrian dynamics," *Phys. Rev. E Stat. Nonlin. Soft Matter Phys.*, vol. 83, no. 5 Pt 2, p. 056117, May 2011.

[29] M. Chraibi, M. Freialdenhoven, A. Schadschneider, and A. Seyfried, "Modeling the desired direction in a force-based model for pedestrian dynamics," *Traffic and Granular Flow '11*, pp. 263–275, 2013.

[30] "Massmotion," http://www.oasys-software.com/products/engineering/massmotion.html, accessed: 2016-11-08.

[31] S. Paris, J. Pettré, and S. Donikian, "Pedestrian reactive navigation for crowd simulation: a predictive approach," *Computer Graphics*

Forum, vol. 26, no. 3, pp. 665–674, 2007. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2007.01090.x

[32] J. Pettré, J. Ondřej, A.-H. Olivier, A. Cretual, and S. Donikian, "Experiment-based modeling, simulation and validation of interactions between virtual walkers," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '09. New York, NY, USA: ACM, 2009, pp. 189–198. [Online]. Available: http://doi.acm.org/10.1145/1599470.1599495

[33] J. van den Berg, M. C. Lin, and D. Manocha, "Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation," in *IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, 2008, pp. 1928–1935.

[34] S. J. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, and P. Dubey, "Clearpath: Highly parallel collision avoidance for multi-agent simulation," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '09. New York, NY, USA: ACM, 2009, pp. 177–187. [Online]. Available: http://doi.acm.org/10.1145/1599470.1599494

[35] I. Karamouzas and M. Overmars, "A velocity-based approach for simulating human collision avoidance," in *Proceedings of the 10th International Conference on Intelligent Virtual Agents*, ser. IVA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 180–186. [Online]. Available: http://dl.acm.org/citation.cfm?id=1889075.1889097

[36] D. Wolinski, M. C. Lin, and J. Pettré, "Warpdriver: Context-aware probabilistic motion prediction for crowd simulation," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 164:1–164:11, Nov. 2016. [Online]. Available: http://doi.acm.org/10.1145/2980179.2982442

[37] A. Treuille, S. Cooper, and Z. Popović, "Continuum crowds," in *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*. New York, NY, USA: ACM, 2006, pp. 1160–1168.

[38] W. G. Wilson, "Resolving discrepancies between deterministic population models and individual-based simulations." *The American naturalist*, vol. 151, pp. 116–34, Feb 1998.

[39] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, and C. Greenough, "Exploitation of High Performance Computing in the FLAME Agent-Based

Simulation Framework," in *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*. IEEE, Jun. 2012, pp. 538–545.

[40] P. Richmond, S. Coakley, and D. Romano, "A High Performance Agent Based Modelling Framework on Graphics Card Hardware with CUDA," *Proceedings of The Eighth International Conference on Autonomous Agents and Multiagent Systems*, 2009.

[41] A. Grandison, Y. Cavanagh, P. J. Lawrence, and E. R. Galea, "Increasing the simulation performance of large-scale evacuations using parallel computing techniques based on domain decomposition," *Fire Technology*, vol. 53, no. 3, pp. 1399–1438, May 2017. [Online]. Available: https://doi.org/10.1007/s10694-016-0645-8

[42] W. Shao and D. Terzopoulos, "Autonomous pedestrians," in *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. New York, NY, USA: ACM Press, 2005, pp. 19–28.

[43] M. Haklay, D. O'Sullivan, M. Thurstain-Goodwin, and T. Schelhorn, ""so go downtown": Simulating pedestrian movement in town centres," *Environment and Planning B: Planning and Design*, vol. 28, no. 3, pp. 343–359, 2001. [Online]. Available: http://dx.doi.org/10.1068/b2758t

[44] K. Zia and A. Ferscha, *City Scale Evacuation: A High-Performance Multi-agent Simulation Framework*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 239–293. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36614-7_10

[45] W. Shao and D. Terzopoulos, "Autonomous pedestrians," in *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. New York, NY, USA: ACM Press, 2005, pp. 19–28. [Online]. Available: http://dx.doi.org/10.1145/1073368.1073371

[46] J. Shopf, J. Barczak, C. Oat, and N. Tatarchuk, "March of the Froblins: simulation and rendering massive crowds of intelligent and detailed creatures on GPU," in *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*. New York, NY, USA: ACM, 2008, pp. 52–101.

[47] J. Pettré, J. Laumond, and D. Thalmann, "A Navigation Graph for Real-Time Crowd Animation on Multilayered and Uneven Terrain," 2005.

[48] W. Shao, "Environmental Modeling for Autonomous Virtual Pedestrians," http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.130.1901, 2005.

[49] A. Sud, E. Andersen, S. Curtis, M. Lin, and D. Manocha, "Real-time path planning for virtual agents in dynamic environments," in *ACM SIGGRAPH 2008 Classes*, ser. SIGGRAPH '08. New York, NY, USA: ACM, 2008, pp. 55:1–55:9. [Online]. Available: http://doi.acm.org/10.1145/1401132.1401206

[50] R. Wein, J. P. van den Berg, and D. Halperin, "The visibility–voronoi complex and its applications," *Computational Geometry*, vol. 36, no. 1, pp. 66 – 87, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925772106000496

[51] A. Sud, R. Gayle, E. Andersen, S. Guy, M. Lin, and D. Manocha, "Real-time navigation of independent agents using adaptive roadmaps," in *VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology*. New York, NY, USA: ACM, 2007, pp. 99–106.

[52] A. Sud, R. Gayle, S. Guy, E. Andersen, M. Lin, and D. Manocha, "Real-time Simulation of Heterogeneous Crowds," 2007.

[53] G. Snook, *Game Programming Gems*. Charles River Media, 2000, ch. Simplified 3D movement and pathfinding using navigation meshes, p. 288–304.

[54] W. van Toll, R. Triesscheijn, M. Kallmann, R. Oliva, N. Pelechano, J. Pettré, and R. Geraerts, "A comparative study of navigation meshes," in *Proceedings of the 9th International Conference on Motion in Games*, ser. MIG '16. New York, NY, USA: ACM, 2016, pp. 91–100. [Online]. Available: http://doi.acm.org/10.1145/2994258.2994262

[55] W. van Toll, A. F. Cook, and R. Geraerts, "Navigation meshes for realistic multi-layered environments," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2011, pp. 3526–3532.

[56] A. Barnett, H. P. H. Shum, and T. Komura, "Coordinated crowd simulation with topological scene analysis," *Comp. Graph. Forum*, vol. 35, no. 6, pp. 120–132, 2016.

[57] C. Gloor, P. Stucki, and K. Nagel, "Hybrid Techniques for Pedestrian Simulations," in *Cellular Automata*, 2004, pp. 581–590.

[58] A. Kneidl, A. Borrmann, and D. Hartmann, "Generation and use of sparse navigation graphs for microscopic pedestrian simulation models," *Advanced Engineering Informatics*, vol. 26, no. 4, pp. 669–680, Oct. 2012.

[59] F. Haron, Y. M. Alginahi, M. N. Kabir, and A. I. Mohamed, "Software evaluation for crowd evacuation- case study: Al-masjid an-nabawi," *International Journal of Computer Science Issues(IJCSI)*, vol. 9, no. 6, 2012.

[60] "EXODUS website," http://fseg.gre.ac.uk/exodus/, accessed: 2017-03-06.

[61] "Urban analytics framework," http://www.crowddynamics.com/products/uaf.php, accessed: 2016-10-17.

[62] J. Hallett, "Introducing decision support systems," *Journal of the Operational Research Society*, vol. 46, no. 10, 1995.

[63] S. H. Ghodsypour and C. O'Brien, "A decision support system for supplier selection using an integrated analytic hierarchy process and linear programming," *International journal of production economics*, vol. 56, pp. 199–212, 1998.

[64] M. Klein and L. B. Methlie, "Knowledge-based decision support systems with applications in business: a decision support approach," 2009.

[65] A. Perini and A. Susi, "Developing a decision support system for integrated production in agriculture," *Environmental Modelling & Software*, vol. 19, no. 9, pp. 821 – 829, 2004, environmental Sciences and Artificial Intelligence. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364815203002007

[66] V. K. Varma, I. Ferguson, and I. Wild, "Decision support system for the sustainable forest management," *Forest Ecology and Management*, vol. 128, no. 1–2, pp. 49 – 55, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378112799002716

[67] J. Törnquist, "Computer-based decision support for railway traffic scheduling and dispatching: A review of models and algorithms," in *OASIcs-OpenAccess Series in Informatics*, vol. 2. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.

[68] P. Richmond and D. Romano, "A High Performance Framework For Agent Based Pedestrian Dynamics on GPU hardware," *European Simulation and Modelling*, 2008.

[69] P. Richmond, D. Walker, S. Coakley, and D. Romano, "High performance cellular level agent-based simulation with FLAME for the GPU," *Brief. Bioinform.*, 2010.

[70] T. Karmakharm, P. Richmond, and D. Romano, "Agent-based Large Scale Simulation of Pedestrians With Adaptive Realistic Navigation Vector Fields," in *Theory and Practice of Computer Graphics (TPCG) 2010*, 2010, pp. 67–74.

[71] T. Karmakharm and P. Richmond, "Large Scale Pedestrian Multi-Simulation for a Decision Support Tool," *Proceedings of Theory and Practice of Computer Graphics (TPCG)*, 13 Sep. 2012.

[72] G. Hoy, E. Morrow, and A. Shalaby, "Use of agent-based crowd simulation to investigate the performance of large-scale intermodal facilities," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2540, pp. 20–29, 2016. [Online]. Available: http://dx.doi.org/10.3141/2540-03

[73] C. Deissenberg, S. Vanderhoog, and H. Dawid, "EURACE: A massively parallel agent-based model of the European economy," *Appl. Math. Comput.*, vol. 204, no. 2, pp. 541–552, 15 Oct. 2008.

[74] H. V. D. Parunak, R. Savit, and L. Riolo, "Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and User's Guide," 1998.

[75] F. Reif, *Fundamentals of statistical and thermal physics*. Waveland Press, 2009, pp. 430–434.

[76] A. Adamatzky, Ed., *Game of Life Cellular Automata*. Springer London, 2010.

[77] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, vol. 21. New York, NY, USA: ACM, Jul. 1987, pp. 25–34.

[78] M.-L. Xu, H. Jiang, X.-G. Jin, and Z. Deng, "Crowd simulation and its applications: Recent advances," *Journal of Computer Science and Technology*, vol. 29, no. 5, pp. 799–811, 2014.

[79] A. Penn and A. Turner, "Space syntax based agent simulation," 2001.

[80] ——, "Encoding natural movement as an agent-based system: an investigation into human pedestrian behaviour in the built environment," 2001.

[81] P. A. Thompson and E. W. Marchant, "A computer model for the evacuation of large building populations," *Fire safety journal*, vol. 24, no. 2, pp. 131–148, 1995.

[82] E. D. Kuligowski, R. D. Peacock, and B. L. Hoskins, *A review of building evacuation models*. US Department of Commerce, National Institute of Standards and Technology Gaithersburg, MD, 2005.

[83] J. S. Roh, H. S. Ryou, W. H. Park, and Y. J. Jang, "Cfd simulation and assessment of life safety in a subway train fire," *Tunnelling and Underground Space Technology*, vol. 24, no. 4, pp. 447–453, 2009.

[84] E. Mas, S. Koshimura, F. Imamura, A. Suppasri, A. Muhari, and B. Adriano, "Recent advances in agent-based tsunami evacuation simulations: Case studies in indonesia, thailand, japan and peru," *Pure and Applied Geophysics*, vol. 172, no. 12, pp. 3409–3424, 2015. [Online]. Available: http://dx.doi.org/10.1007/s00024-015-1105-y

[85] N. Chooramun, P. Lawrence, and E. Galea, "Urban scale evacuation simulation using buildingexodus," *Interflam 2016*, vol. 2, pp. 1645–1656, 07 2016.

[86] E. A. Khan, M. A. Ahmed, E. H. Khan, and S. C. Majumder, "Fire emergency evacuation simulation of a shopping mall using fire dynamic simulator (fds)," *Journal of Chemical Engineering*, vol. 30, no. 1, pp. 32–36, 2017.

[87] D. Thalmann, "Populating virtual environments with crowds," in *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*. ACM, 2006, pp. 11–11.

[88] S. Dobbyn, R. McDonnell, L. Kavan, S. Collins, and C. O'Sullivan, "Clothing the masses: Real-time clothed crowds with variation." in *Eurographics (Short Presentations)*, 2006, pp. 103–106.

[89] J. Maïm, B. Yersin, and D. Thalmann, "Unique Character Instances for Crowds," *IEEE Comput. Graph. Appl.*, vol. 29, no. 6, pp. 82–90, 2009.

[90] T. Kwon, K. H. Lee, J. Lee, and S. Takahashi, "Group motion editing," in *ACM SIGGRAPH 2008 Papers*, ser. SIGGRAPH '08. New York, NY, USA: ACM, 2008, pp. 80:1–80:8. [Online]. Available: http://doi.acm.org/10.1145/1399504.1360679

[91] S. Takahashi, K. Yoshida, T. Kwon, K. H. Lee, J. Lee, and S. Y. Shin, "Spectral-based group formation control," *Computer Graphics Forum*, vol. 28, no. 2, pp. 639–648, 2009. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2009.01404.x

[92] J. Kim, Y. Seol, T. Kwon, and J. Lee, "Interactive manipulation of large-scale crowd animation," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 83:1–83:10, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2601097.2601170

[93] J. Henry, H. P. H. Shum, and T. Komura, "Interactive formation control in complex environments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 2, pp. 211–222, Feb 2014.

[94] A. Lerner, Y. Chrysanthou, and D. Lischinski, "Crowds by example," *Computer Graphics Forum*, vol. 26, no. 3, pp. 655–664, 2007. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2007.01089.x

[95] E. Ju, M. G. Choi, M. Park, J. Lee, K. H. Lee, and S. Takahashi, "Morphable crowds," *ACM Trans. Graph.*, vol. 29, no. 6, pp. 140:1–140:10, Dec. 2010. [Online]. Available: http://doi.acm.org/10.1145/1882261.1866162

[96] M. G. Choi, M. Kim, K. L. Hyun, and J. Lee, "Deformable motion: Squeezing into cluttered environments," *Computer Graphics Forum*, vol. 30, no. 2, pp. 445–453, 2011. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2011.01889.x

[97] M. Sung, M. Gleicher, and S. Chenney, "Scalable behaviors for crowd simulation," *Computer Graphics Forum*, vol. 23, no. 3, pp. 519–528, 2004. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2004.00783.x

[98] R. McDonnell, M. Larkin, S. Dobbyn, S. Collins, and C. O'Sullivan, "Clone attack! Perception of crowd variety," in *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*. New York, NY, USA: ACM, 2008, pp. 1–8.

[99] D. Helbing, "A mathematical model for the behavior of pedestrians," *Behav. Sci.*, vol. 36, no. 4, pp. 298–310, Oct. 1991.

[100] D. Helbing, P. Molnar, I. J. Farkas, and K. Bolay, "Self-organizing pedestrian movement," *Environ. Plann. B Plann. Des.*, 2001.

[101] M. Moussaïd, D. Helbing, S. Garnier, A. Johansson, M. Combe, and G. Theraulaz, "Experimental study of the behavioural mechanisms underlying

self-organization in human crowds," *Proceedings of the Royal Society of London B: Biological Sciences*, vol. 276, no. 1668, pp. 2755–2762, 2009. [Online]. Available: http://rspb.royalsocietypublishing.org/content/276/1668/2755

[102] M. Moussa\"id, N. Perozo, S. Garnier, D. Helbing, and G. Theraulaz, "The Walking Behaviour of Pedestrian Social Groups and Its Impact on Crowd Dynamics," *PLoS One*, vol. 5, no. 4, pp. e10 047+, 7 Apr. 2010.

[103] E. Ronchi, E. D. Kuligowski, P. A. Reneke, R. D. Peacock, and D. Nilsson, *The process of verification and validation of building fire evacuation models*. US Department of Commerce, National Institute of Standards and Technology, 2013.

[104] "Massive Software," http://www.massivesoftware.com/, accessed: 2017-10-11.

[105] "Miarmy," http://www.basefount.com/miarmy.html, accessed: 2017-03-06.

[106] B. Ulicny and D. Thalmann, "Crowd simulation for interactive virtual environments and vr training systems," in *Computer Animation and Simulation 2001*. Springer, 2001, pp. 163–170.

[107] N. Pelechano, C. Stocker, J. Allbeck, and N. Badler, "Being a part of the crowd: towards validating VR crowds using presence," in *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 136–142.

[108] D. Panzoli, C. Peters, I. Dunwell, S. Sanchez, P. Petridis, A. Protopsaltis, V. Scesa, and S. de Freitas, "A level of interaction framework for exploratory learning with characters in virtual environments," in *Intelligent Computer Graphics 2010*. Springer, 2010, pp. 123–143.

[109] "Total War," http://www.totalwar.com, accessed: 2017-03-06.

[110] E. d'Eon, D. Luebke, and E. Enderton, "Efficient rendering of human skin," in *Proceedings of the 18th Eurographics conference on Rendering Techniques*. Eurographics Association, 2007, pp. 147–157.

[111] C. O'Sullivan, J. Cassell, H. Vilhjalmsson, J. Dingliana, S. Dobbyn, B. Mcnamee, C. Peters, and T. Giang, "Levels of detail for crowds and groups," in *Computer Graphics Forum*, vol. 21, no. 4. Wiley Online Library, 2002, pp. 733–741.

[112] S. Dobbyn, J. Hamill, K. O'Conor, and C. O'Sullivan, "Geopostors: a real-time geometry/impostor crowd rendering system," in *Proceedings of the 2005 symposium on Interactive 3D graphics and games*. ACM, 2005, pp. 95–102.

[113] E. Millan and I. Rudomin, "Impostors and pseudo-instancing for gpu crowd rendering," in *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*. ACM, 2006, pp. 49–55.

[114] J. Pettré, P. d. H. Ciechomski, J. Maïm, B. Yersin, J.-P. Laumond, and D. Thalmann, "Real-time navigating crowds: scalable simulation and rendering," *Comput. Animat. Virtual Worlds*, vol. 17, no. 3-4, pp. 445–455, 1 Jul. 2006.

[115] B. Ulicny, P. d. H. Ciechomski, and D. Thalmann, "Crowdbrush: interactive authoring of real-time crowd scenes," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 2004, pp. 243–252.

[116] R. F. Fahy, "Exit89: an evacuation model for high-rise buildings," *Fire Safety Science*, vol. 3, pp. 815–823, 1991.

[117] Y. Min and Y. Yu, "Calculation of mixed evacuation of stair and elevator using evacnet4," *Procedia Engineering*, vol. 62, pp. 478–482, 2013.

[118] D. Helbing, I. J. Farkas, P. Molnar, and T. Vicsek, "Simulation of pedestrian crowds in normal and evacuation situations," in *Pedestrian and Evacuation Dynamics*, 1 Jan. 2002, vol. 21, pp. 21–58.

[119] F. Alonso-Marroquín, J. Busch, C. Chiew, C. Lozano, and A. Ramírez-Gómez, "Simulation of counterflow pedestrian dynamics using spheropolygons," *Phys. Rev. E Stat. Nonlin. Soft Matter Phys.*, vol. 90, no. 6, p. 063305, Dec. 2014.

[120] N. Chooramun, P. J. Lawrence, and E. R. Galea, "An agent based evacuation model utilising hybrid space discretisation," *Safety Science*, vol. 50, no. 8, pp. 1685 – 1694, 2012, evacuation and Pedestrian Dynamics. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925753511003419

[121] L. F. Henderson, "The statistics of crowd fluids," *Nature*, vol. 229, no. 5284, pp. 381–383, 5 Feb. 1971.

[122] R. Hughes, "The flow of large crowds of pedestrians," *Mathematics and Computers in Simulation*, vol. 53, no. 4-6, pp. 367–370, 2000, cited By

63. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0006212847&partnerID=40&md5=80619defa91e265adab489070cdecc6c

[123] ——, "A continuum theory for the flow of pedestrians," *Trans. Res. Part B: Methodol.*, vol. 36, no. 6, pp. 507–535, Jul. 2002.

[124] ——, "The flow of human crowds," *Annual Review of Fluid Mechanics*, vol. 35, pp. 169–182, 2003, cited By 270. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0346245136&doi=10.1146%2fannurev.fluid.35.101101.161136&partnerID=40&md5=5081dbff962e7f5d68627ae10b4db4e5

[125] S. Seer, C. Rudloff, T. Matyus, and N. Brändle, "Validating social force based models with comprehensive real world motion data," *Transportation Research Procedia*, vol. 2, pp. 724–732, 2014.

[126] M. Moussaïd, D. Helbing, and G. Theraulaz, "How simple rules determine pedestrian behavior and crowd disasters," *Proceedings of the National Academy of Sciences*, vol. 108, no. 17, pp. 6884–6888, 2011. [Online]. Available: http://www.pnas.org/content/108/17/6884.abstract

[127] M. Moussaïd and J. D. Nelson, "Simple heuristics and the modelling of crowd behaviours," in *Pedestrian and Evacuation Dynamics 2012*, U. Weidmann, U. Kirsch, and M. Schreckenberg, Eds. Cham: Springer International Publishing, 2014, pp. 75–90.

[128] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998. [Online]. Available: http://ijr.sagepub.com/content/17/7/760.abstract

[129] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, *Reciprocal n-Body Collision Avoidance*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–19. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-19457-3_1

[130] J. Snape, S. J. Guy, D. Vembar, A. Lake, and others, "Reciprocal collision avoidance and navigation for video games," *Conf., San Francisco*, 2012.

[131] N. Pelechano, J. M. Allbeck, and N. I. Badler, "Controlling individual agents in high-density crowd simulation," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ser. SCA '07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, pp. 99–108.

[132] F. Qiu and X. Hu, "Modeling group structures in pedestrian crowd simulation," *Simulation Modelling Practice and Theory*, vol. 18, no. 2, pp. 190–205, 2010.

[133] Y. Murakami, K. Minami, T. Kawasoe, and T. Ishida, "Multi-agent simulation for crisis management," in *Knowledge Media Networking, 2002. Proceedings. IEEE Workshop on*. IEEE, 2002, pp. 135–139.

[134] J. Ondřej, J. Pettré, A.-H. Olivier, and S. Donikian, "A synthetic-vision based steering approach for crowd simulation," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 123:1–123:9, Jul. 2010.

[135] "STEPS software," http://www.steps.mottmac.com/, accessed: 2017-03-06.

[136] J. P. Yuan, Z. Fang, Y. C. Wang, S. M. Lo, and P. Wang, "Integrated network approach of evacuation simulation for large complex buildings," *Fire Saf. J.*, vol. 44, no. 2, pp. 266–275, Feb. 2009.

[137] S. Lemercier and J.-M. Auberlet, "Towards more behaviours in crowd simulation," *Computer Animation and Virtual Worlds*, vol. 27, no. 1, pp. 24–34, 2016, cav.1629. [Online]. Available: http://dx.doi.org/10.1002/cav.1629

[138] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, no. 10, pp. 560–570, Oct. 1979. [Online]. Available: http://doi.acm.org/10.1145/359156.359164

[139] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg, "Real-time robot motion planning using rasterizing computer graphics hardware," in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '90. New York, NY, USA: ACM, 1990, pp. 327–335. [Online]. Available: http://doi.acm.org/10.1145/97879.97915

[140] T. Lozano-Pérez, *Spatial Planning: A Configuration Space Approach*. New York, NY: Springer New York, 1990, pp. 259–271. [Online]. Available: http://dx.doi.org/10.1007/978-1-4613-8997-2_20

[141] J. J. Kuffner, Jr., "Goal-directed navigation for animated characters using real-time path planning and control," in *Proceedings of the International Workshop on Modelling and Motion Capture Techniques for Virtual Environments*, ser. CAPTECH '98. London, UK, UK: Springer-Verlag, 1998, pp. 171–186. [Online]. Available: http://dl.acm.org/citation.cfm?id=647338.723378

[142] S. Bandi and D. Thalmann, "Path finding for human motion in virtual environments," *Comput. Geom.*, vol. 15, no. 1-3, pp. 103–127, Feb. 2000.

[143] D. Delling, P. Sanders, D. Schultes, and D. Wagner, *Engineering Route Planning Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 117–139. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02094-0_7

[144] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: The case for a*," *Int. J. Geogr. Inf. Sci.*, vol. 23, no. 4, pp. 531–543, Apr. 2009. [Online]. Available: http://dx.doi.org/10.1080/13658810801949850

[145] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[146] S. Patil, J. Van Den Berg, S. Curtis, M. C. Lin, and D. Manocha, "Directing crowd simulations using navigation fields," *IEEE transactions on visualization and computer graphics*, vol. 17, no. 2, pp. 244–254, 2011.

[147] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 29 May 2006.

[148] S. Chenney, "Flow tiles," in *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2004, pp. 233–242.

[149] X. Jin, J. Xu, C. C. L. Wang, S. Huang, and J. Zhang, "Interactive Control of Large-Crowd Navigation in Virtual Environments Using Vector Fields," *IEEE Comput. Graph. Appl.*, vol. 28, no. 6, pp. 37–46, Nov. 2008.

[150] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 995–1001 vol.2.

[151] X. Yu, J. A. Goldak, and L. Dong, "Constructing 3-D discrete medial axis," in *SMA '91: Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*. New York, NY, USA: ACM, 1991, pp. 481–489.

[152] W. H. Hesselink and J. B. Roerdink, "Euclidean skeletons of digital image and volume data in linear time by the integer medial axis transform." *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 12, pp. 2204–2217, Dec. 2008.

[153] H. Xia and P. Tucker, "Distance Solutions for Medial Axis Transform," in *Proceedings of the 18th International Meshing Roundtable*, B. Clark, Ed. Springer Berlin Heidelberg, 2009, pp. 247–265.

[154] W. van Toll and R. Geraerts, "Dynamically pruned a* for re-planning in navigation meshes," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015, pp. 2051–2057.

[155] G. Reeb, "Sur les points singuliers d'une forme de pfaff complètement intégrable ou d'une fonction numérique [on the singular points of a completely integrable pfaff form or of a numerical function]," *Comptes Rendus Acad. Sciences*, vol. 222, pp. 847–849, 1946.

[156] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii, "Topology matching for fully automatic similarity estimation of 3d shapes," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 203–212. [Online]. Available: http://doi.acm.org/10.1145/383259.383282

[157] R. Oliva and N. Pelechano, "Neogen: Near optimal generator of navigation meshes for 3d multi-layered environments," *Computers & Graphics*, vol. 37, no. 5, pp. 403 – 412, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0097849313000435

[158] M. Mononen, "Recast navigation," https://github.com/recastnavigation/recastnavigation, 2014, accessed: 2017-03-03.

[159] B. Yersin, J. Maím, P. D. H. Ciechomski, S. Schertenleib, and D. Thalmann, "Steering a virtual crowd based on a semantically augmented navigation graph," in *In Proceedings of the First International Workshop on Crowd Simulation*, 2005, pp. 169–178.

[160] P. N. Seneviratne and J. F. Morrall, "Analysis of factors affecting the choice of route of pedestrians," *Transportation Planning and Technology*, vol. 10, no. 2, pp. 147–159, 1985. [Online]. Available: http://dx.doi.org/10.1080/03081068508717309

[161] A. W. Agrawal, M. Schlossberg, and K. Irvin, "How far, by which route and why? a spatial analysis of pedestrian preference," *Journal of Urban Design*, vol. 13, no. 1, pp. 81–98, 2008. [Online]. Available: http://dx.doi.org/10.1080/13574800701804074

[162] R. G. Golledge, "Path selection and route preference in human navigation: A progress report," in *International Conference on Spatial Information Theory*. Springer, 1995, pp. 207–222.

[163] I. Armeni and K. Chorianopoulos, "Pedestrian navigation and shortest path: Preference versus distance," in *Ambient Intelligence and Smart Environments*, 2014, pp. 647–652.

[164] E. Kuligowski, R. D Peacock, and B. Hoskins, "A review of building evacuation models, 2nd edition, nist technical note 1680," 01 2010.

[165] "Pathfinder software," https://www.thunderheadeng.com/pathfinder/, accessed: 2017-03-06.

[166] P. Thompson, J. Wu, and E. Marchant, "Simulex 3.0: Modelling evacuation in multi-storey buildings," *Fire Safety Science*, vol. 5, pp. 725–736, 1997.

[167] "Simwalk," http://www.simwalk.com, accessed: 2017-03-06.

[168] M. Harmon and J. Joseph, "Evacuation planning tool (ept) for emergency, event and space planning," in *Pedestrian and Evacuation Dynamics*. Springer, 2011, pp. 785–788.

[169] R. Kukla, J. Kerridge, A. Willis, and J. Hine, "Pedflow: Development of an autonomous agent model of pedestrian flow," *Transportation research record: Journal of the transportation research board*, no. 1774, pp. 11–17, 2001.

[170] M. Bensilum and D. A. Purser, "Grid flow: An object-oriented building evacuation model combining pre-movement and movement behaviours for performance-based design," *Fire Safety Science*, vol. 7, pp. 941–952, 2003.

[171] "MASSEgress website," http://http://eil.stanford.edu/pengao/ResentFocus/index.html, accessed: 2017-03-06.

[172] S. F. Railsback, S. L. Lytinen, and S. K. Jackson, "Agent-based simulation platforms: Review and development recommendations," *Simulation*, vol. 82, no. 9, pp. 609–623, 2006.

[173] R. Löhner, "On the modeling of pedestrian motion," *Applied Mathematical Modelling*, vol. 34, no. 2, pp. 366 – 382, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0307904X09001395

[174] A. Rousset, B. Herrmann, C. Lang, and L. Philippe, "A survey on parallel and distributed multi-agent systems for high performance computing simulation," *Computer Science Review*, vol. itansdafsdf, pp. –, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574013715300435

[175] G. Cordasco, R. De Chiara, A. Mancuso, D. Mazzeo, V. Scarano, and C. Spagnuolo, "Bringing together efficiency and effectiveness in distributed simulations: The experience with d-mason," *SIMULATION*, vol. 89, no. 10, pp. 1236–1253, 2013. [Online]. Available: http://sim.sagepub.com/content/89/10/1236.abstract

[176] N. Collier and M. North, *Repast HPC: A Platform for Large-Scale Agent-Based Modeling*. John Wiley & Sons, Inc., 2012, pp. 81–109. [Online]. Available: http://dx.doi.org/10.1002/9781118130506.ch5

[177] E. S. Angelotti, E. E. Scalabrin, and B. C. Avila, "Pandora: a multi-agent system using paraconsistent logic," in *Computational Intelligence and Multimedia Applications, 2001. ICCIMA 2001. Proceedings. Fourth International Conference on*, 2001, pp. 352–356.

[178] "Anylogic," http://www.anylogic.com/, accessed: 2016-11-01.

[179] "Development and structure of prometheus : the canadian wildland fire growth simulation model," http://www.firegrowthmodel.ca/prometheus/overview_e.php, accessed: 2017-02-20.

[180] E. S. Berner, *Clinical decision support systems*. Springer, 2007.

[181] A. Kneidl, M. Thiemann, D. Hartmann, and A. Borrmann, "Combining pedestrian simulation with a network flow optimization to support security staff in handling an evacuation of a soccer stadium," in *Proceedings of European Conference Forum*, 2011.

[182] L. Feng and E. Miller-Hooks, "A network optimization-based approach for crowd management in large public gatherings," *Transportation Research Part C: Emerging Technologies*, vol. 42, pp. 182 – 199, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0968090X1400031X

[183] S. Hoogendoorn, W. Daamen, D. Duives, and F. van Wageningen-Kessels, *Optimal crowd evacuation*. TRB, 2013.

[184] D. R. Bish, H. D. Sherali, and A. G. Hobeika, "Optimal evacuation planning using staging and routing," *Journal of the Operational Research Society*, vol. 65, no. 1, pp. 124–140, 2014. [Online]. Available: http://dx.doi.org/10.1057/jors.2013.3

[185] E. Galea, D. Cooney, L. Filippidis *et al.*, "Active dynamic signage system: A full-scale evacuation trial," 2015.

[186] "Estimates of station usage - office of rail and road," http://orr.gov.uk/statistics/published-stats/station-usage-estimates, accessed: 2017-03-8.

[187] M. Alzantot and M. Youssef, "Uptime: Ubiquitous pedestrian tracking using mobile phones," in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2012, pp. 3204–3209.

[188] P. Mirowski, T. K. Ho, S. Yi, and M. MacDonald, "Signalslam: Simultaneous localization and mapping with mixed wifi, bluetooth, lte and magnetic signals," in *International Conference on Indoor Positioning and Indoor Navigation*, Oct 2013, pp. 1–10.

[189] Y. Malinovskiy, N. Saunier, and Y. Wang, "Analysis of pedestrian travel with static bluetooth sensors," *Transportation Research Record: Journal of the Transportation Research Board*, no. 2299, pp. 137–149, 2012.

[190] N. Abedi, A. Bhaskar, and E. Chung, "Bluetooth and wi-fi mac address based crowd data collection and monitoring: benefits, challenges and enhancement," *36th Australasian Transport Research Forum (ATRF)*, 2013.

[191] L. Schauer, M. Werner, and P. Marcus, "Estimating crowd densities and pedestrian flows using wi-fi and bluetooth," in *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, ser. MOBIQUITOUS '14. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014, pp. 171–177. [Online]. Available: http://dx.doi.org/10.4108/icst.mobiquitous.2014.257870

[192] N. Shlayan, A. Kurkcu, and K. Ozbay, "Exploring pedestrian bluetooth and wifi detection at public transportation terminals," in *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*. IEEE, 2016, pp. 229–234.

[193] A. Lesani and L. F. Miranda-Moreno, "Development and testing of a real-time wifi-bluetooth system for pedestrian network monitoring and data extrapolation," in *Transportation Research Board 95th Annual Meeting*, no. 16-5665, 2016.

[194] J. Weppner, B. Bischke, and P. Lukowicz, "Monitoring crowd condition in public spaces by tracking mobile consumer devices with wifi interface," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, ser. UbiComp '16. New York, NY, USA: ACM, 2016, pp. 1363–1371. [Online]. Available: http://doi.acm.org/10.1145/2968219.2968414

[195] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[196] X. Zeng, W. Ouyang, B. Yang, J. Yan, and X. Wang, "Gated bi-directional cnn for object detection," in *European Conference on Computer Vision*. Springer, 2016, pp. 354–369.

[197] R. Bodor, B. Jackson, and N. Papanikolopoulos, "Vision-based human tracking and activity recognition," in *Proc. of the 11th Mediterranean Conf. on Control and Automation*, vol. 1. Citeseer, 2003.

[198] F. Xu, X. Liu, and K. Fujimura, "Pedestrian detection and tracking with night vision," *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 1, pp. 63–71, 2005.

[199] P. Chong and Y. H. Tay, "A novel pedestrian detection and tracking with boosted hog classifiers and kalman filter," in *Research and Development (SCOReD), 2016 IEEE Student Conference on*. IEEE, 2016, pp. 1–5.

[200] D. Ribeiro, A. Mateus, J. C. Nascimento, and P. Miraldo, "A real-time pedestrian detector using deep learning for human-aware navigation," *arXiv preprint arXiv:1607.04441*, 2016.

[201] A. Dehghan and M. Shah, "Binary quadratic programing for online tracking of hundreds of people in extremely crowded scenes," *arXiv preprint arXiv:1603.09240*, 2016.

[202] K. Teknomo, Y. Takeyama, and H. Inamura, "Tracking system to automate data collection of microscopic pedestrian traffic flow," *arXiv preprint arXiv:1609.01810*, 2016.

[203] E. Goubet, J. Katz, and F. Porikli, "Pedestrian tracking using thermal infrared imaging," in *Proceedings of SPIE*, vol. 6206, 2006, pp. 797–808.

[204] A. Leykin and R. Hammoud, "Pedestrian tracking by fusion of thermal-visible surveillance videos," *Machine Vision and Applications*, vol. 21, no. 4, pp. 587–595, 2010.

[205] G. Lian, J. Lai, and W.-S. Zheng, "Spatial–temporal consistent labeling of tracked pedestrians across non-overlapping camera views," *Pattern Recognition*, vol. 44, no. 5, pp. 1121–1136, 2011.

[206] F. Tan, L. Huang, C. Zhai, M. Song, R. Zhuang, and W. Liu, "Specific object re-identification across non-overlapping camera views in traffic accidents," *Transportation Planning and Technology*, vol. 39, no. 8, pp. 759–767, 2016.

[207] T. Xiao, H. Li, W. Ouyang, and X. Wang, "Learning deep feature representations with domain guided dropout for person re-identification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1249–1258.

[208] "Titan supercomputer," https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7/, accessed: 2017-03-06.

[209] "Piz diant, cscs," http://www.cscs.ch/computers/piz_daint_piz_dora/index.html, accessed: 2016-11-01.

[210] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, "Medical image processing on the gpu–past, present and future," *Medical image analysis*, vol. 17, no. 8, pp. 1073–1094, 2013.

[211] K. Kim, J. Gu, S. Tyree, P. Molchanov, M. Nießner, and J. Kautz, "A lightweight approach for on-the-fly reflectance estimation," *arXiv preprint arXiv:1705.07162*, 2017.

[212] J. G. X. Y. S. De and M. J. Kautz, "Dynamic facial analysis: From bayesian filtering to recurrent neural network," 2017.

[213] M. Yan, I. Frosio, S. Tyree, and J. Kautz, "Sim-to-real transfer of accurate grasping with eye-in-hand observations and continuous control," *arXiv preprint arXiv:1712.03303*, 2017.

[214] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield, "Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness," *arXiv preprint arXiv:1705.02550*, 2017.

[215] S. Laine and T. Aila, "Temporal ensembling for semi-supervised learning," *arXiv preprint arXiv:1610.02242*, 2016.

[216] T. Karras, T. Aila, S. Laine, A. Herva, and J. Lehtinen, "Audio-driven facial animation by joint end-to-end learning of pose and emotion," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 94, 2017.

[217] J. Rodríguez-Navarro and A. Susín Sánchez, "Non structured meshes for cloth gpu simulation using fem," in *3rd Workshop in Virtual Reality Interactions and Physical Simulation*. EUROGRAPHICS, 2006, pp. 1–7.

[218] T. Shimokawabe, T. Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo, A. Nukada, N. Maruyama, and S. Matsuoka, "An 80-fold speedup, 15.0 tflops full gpu acceleration of non-hydrostatic weather model asuca production code," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–11.

[219] M. Burtscher and K. Pingali, "An efficient cuda implementation of the tree-based barnes hut n-body algorithm," in *GPU computing Gems Emerald edition*. Elsevier, 2011, pp. 75–92.

[220] J. Bédorf, E. Gaburov, and S. P. Zwart, "A sparse octree gravitational n-body code that runs entirely on the gpu processor," *Journal of Computational Physics*, vol. 231, no. 7, pp. 2825–2839, 2012.

[221] A. J. Crespo, J. M. Domínguez, B. D. Rogers, M. Gómez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, and O. García-Feal, "Dualsphysics: Open-source parallel cfd solver based on smoothed particle hydrodynamics (sph)," *Computer Physics Communications*, vol. 187, pp. 204–216, 2015.

[222] "Cpu, gpu and mic hardware characteristics over time," https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time, accessed: 2017-12-01.

[223] "Nvidia dgx-1," https://www.nvidia.com/en-us/data-center/dgx-1, accessed: 2017-12-01.

[224] L. Zhang, B. Jiang, Y. Wu, C. Strouthos, P. Z. Sun, J. Su, and X. Zhou, "Developing a multiscale, multi-resolution agent-based brain tumor model by graphics processing units," *Theoretical Biology and Medical Modelling*, vol. 8, no. 1, p. 46, 2011.

[225] Z. Wang, J. D. Butner, R. Kerketta, V. Cristini, and T. S. Deisboeck, "Simulating cancer growth with multiscale agent-based modeling," in *Seminars in cancer biology*, vol. 30. Elsevier, 2015, pp. 70–78.

[226] R. M. D'Souza, M. Lysenko, S. Marino, and D. Kirschner, "Data-parallel algorithms for agent-based model simulation of tuberculosis on graphics processing units," in *Proceedings of the 2009 spring simulation multiconference*. Society for Computer Simulation International, 2009, p. 21.

[227] A. Ulbinaitė and Y. Le Moullec, "Towards an abm-based framework for investigating consumer behaviour in the insurance industry," *Ekonomika*, vol. 89, 2010.

[228] P. Heywood, S. Maddock, J. Casas, D. Garcia, M. Brackstone, and P. Richmond, "Data-parallel agent-based microscopic road network simulation using graphics processing units," *Simulation Modelling Practice and Theory*, 2017.

[229] *Simulation of large crowds in emergency situations including gaseous phenomena*, 2005.

[230] A. Rahman, N. A. W. A. Hamid, A. R. Rahiman, and B. Zafar, "Towards accelerated agent-based crowd simulation for hajj and umrah," in *Agents, Multi-Agent Systems and Robotics (ISAMSR), 2015 International Symposium on*, Aug 2015, pp. 65–70.

[231] M. Joselli, E. B. Passos, M. Zamith, E. Clua, A. Montenegro, and B. Feijó, "A neighborhood grid data structure for massive 3d crowd simulation on gpu," in *2009 VIII Brazilian Symposium on Games and Digital Entertainment*, Oct 2009, pp. 121–131.

[232] I. Rudomin, B. Hernández, O. De Gyves, L. Toledo, I. Rivalcoba, and S. Ruiz, "Gpu generation of large varied animated crowds," *Computación y Sistemas*, vol. 17, no. 3, 2013.

[233] H. Mroz and J. WĄS, "Discrete vs. continuous approach in crowd dynamics modeling using gpu computing," *Cybernetics and Systems*, vol. 45, no. 1, pp. 25–38, 2014.

[234] B. Hernández, H. P'erez, I. Rudomin, S. Ruiz, O. de Gyves, and L. Toledo, "Simulating and Visualizing Real-Time Crowds on GPU Clusters," *Computación y Sistemas*, vol. 18, pp. 651 – 664, 12 2014.

[Online]. Available: http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S1405-55462014000400002&nrm=iso

[235] M. Lysenko and R. M. D'Souza, "A framework for megascale agent based model simulations on graphics processing units," *Journal of Artificial Societies and Social Simulation*, vol. 11, no. 4, p. 10, 2008. [Online]. Available: http://jasss.soc.surrey.ac.uk/11/4/10.html

[236] "Flame gpu website," http://www.flamegpu.com/, accessed: 2017-11-01.

[237] P. Richmond, "FLAME GPU Technical Report and User Guide (CS-11-03)," Department of Computer Science, University of Sheffield, Tech. Rep., 2011.

[238] "Report on the sunway taihulight," http://www.netlib.org/utk/people/JackDongarra/PAPERS/sunway-report-2016.pdf, accessed: 2016-11-20.

[239] "Ibm supercomputer overtakes japan's fujitsu as world's fastest," http://www.techspot.com/news/49026-ibm-supercomputer-overtakes-japans-fujitsu-as-worlds-fastest.html, accessed: 2016-11-01.

[240] M. Wijerathne, L. Melgar, M. Hori, T. Ichimura, and S. Tanaka, "Hpc enhanced large urban area evacuation simulations with vision based autonomously navigating multi agents," *Procedia Computer Science*, vol. 18, pp. 1515 – 1524, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050913004626

[241] D. Pianini, M. Viroli, F. Zambonelli, and A. Ferscha, "Hpc from a self-organisation perspective: The case of crowd steering at the urban scale," in *High Performance Computing Simulation (HPCS), 2014 International Conference on*, July 2014, pp. 460–467.

[242] T. Mao, H. Jiang, J. Li, Y. Zhang, S. Xia, and Z. Wang, "Parallelizing continuum crowds," in *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '10. New York, NY, USA: ACM, 2010, pp. 231–234. [Online]. Available: http://doi.acm.org/10.1145/1889863.1889914

[243] K. Zia, A. Riener, K. Farrahi, and A. Ferscha, "A new opportunity to urban evacuation analysis: Very large scale simulations of social agent systems in repast hpc," in *Principles of Advanced and Distributed Simulation (PADS), 2012 ACM/IEEE/SCS 26th Workshop on*, July 2012, pp. 233–242.

[244] R. Dulam, M. Lalith, M. Hori, T. Ichimura, and S. Tanaka, "Development of an hpc enhanced multi agent simulation code for tsunami evacuation," *Journal of Japan Society of Civil Engineers, Ser. A2 (Applied Mechanics (AM))*, vol. 68, no. 2, pp. 513–521, 2012.

[245] A. Gutierrez-Milla, F. Borges, R. Suppi, and E. Luque, "Crowd dynamics modeling and collision avoidance with openmp," in *2015 Winter Simulation Conference (WSC)*, Dec 2015, pp. 3128–3129.

[246] A. Grandison, Y. Muthu, P. Lawrence, and E. R. Galea, "Simulating the evacuation of very large populations in large domains using a parallel implementation of the buildingexodus evacuation model," in *Interflam 2007 11th International Fire Science and Engineering Conference*. Greenwich: Interscience Communications, 2007, vol. 1, pp. 259–270, this paper forms part of the published proceedings from 11th International Fire Science & Engineering Conference, Interflam 2007, 3-5th September 2007, Royal Holloway College, University of London, UK. [Online]. Available: http://gala.gre.ac.uk/1095/

[247] B. Y. Y. Mohedeen, "Domain partitioning and software modifications towards the parallelisation of the buildingexodus evacuation software," 2011.

[248] Y. Hirokawa, N. Nishikawa, T. Asano, M. Terai, and T. Matsuzawa, "A study of real-time and 100 billion agents simulation using the boids model," *Artificial Life and Robotics*, pp. 1–6, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10015-016-0308-3

[249] D. Jacobsen, J. C. Thibault, and I. Senocak, "An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters," in *American Institute of Aeronautics and Astronautics (AIAA) 48th Aerospace Science Meeting Proceedings*, 2010.

[250] R. Abdelkhalek, H. Calandra, O. Coulaud, J. Roman, and G. Latu, "Fast seismic modeling and Reverse Time Migration on a GPU cluster," in *High Performance Computing & Simulation, 2009. HPCS '09. International Conference on*. IEEE, Jun. 2009, pp. 36–43.

[251] H. Pérez, B. Hernández, I. Rudomín, and E. Ayguadé, *Scaling Crowd Simulations in a GPU Accelerated Cluster*. Cham: Springer International Publishing, 2016, pp. 461–472. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-32243-8_32

[252] M. Holcombe, S. Adra, M. Bicak, S. Chin, S. Coakley, A. I. Graham, J. Green, C. Greenough, D. Jackson, M. Kiran *et al.*, "Modelling complex biological systems using an agent-based approach," *Integrative Biology*, vol. 4, no. 1, pp. 53–64, 2012.

[253] M. Burkitt, D. Walker, D. Romano, and A. Fazeli, "Using computational modeling to investigate sperm navigation and behavior in the female reproductive tract," *Theriogenology*, vol. 77, no. 4, pp. 703–716, 2012.

[254] H. Kaul, Z. Cui, and Y. Ventikos, "A multi-paradigm modeling framework to simulate dynamic reciprocity in a bioreactor," *PLoS One*, vol. 8, no. 3, p. e59671, 2013.

[255] X. Li, A. Upadhyay, A. Bullock, T. Dicolandrea, J. Xu, R. Binder, M. Robinson, D. Finlay, K. Mills, C. Bascom *et al.*, "Skin stem cell hypotheses and long term clone survival–explored using agent-based modelling," *Scientific reports*, vol. 3, p. 1904, 2013.

[256] T. Balanescu, A. J. Cowling, H. Georgescu, M. Gheorghe, M. Holcombe, and C. Vertan, "Communicating stream X-machines systems are no more than X-machines," in *In Twelfth International Symposium on Fundamentals of Computation Theory (FCT'99), Iasi*, 1999, pp. 494–507.

[257] S. Coakley, R. Smallwood, and M. Holcombe, "Using X-Machines as a Formal Basis for Describing Agents in Agent-Based Modelling," *Proceedings of the 2006 Springg Simulation Multiconference*, pp. 33–40, Apr. 2006.

[258] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, T. R. o. the SBML Forum:, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novère, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang, "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models," *Bioinformatics*, vol. 19, no. 4, pp. 524–531, 1 Mar. 2003.

[259] A. A. Cuellar, C. M. Lloyd, P. F. Nielsen, D. P. Bullivant, D. P. Nickerson, and P. J. Hunter, "An Overview of CellML 1.1, a Biological Model Description Language," *Simulation*, vol. 79, no. 12, pp. 740–747, 1 Dec. 2003.

[260] S. Eilenberg, *Automata, Languages, and Machines*. Orlando, FL, USA: Academic Press, Inc., 1974.

[261] "Flame flexible large-scale agent modelling environment," http://www.flame.ac.uk, accessed: 2016-11-01.

[262] P. Richmond and D. Romano, "Template-Driven Agent-Based Modeling and Simulation with CUDA," in *GPU Computing Gems Emerald Edition*, 1st ed., ser. Applications of GPU Computing Series, W.-M. W. Hwu, Ed. Morgan Kaufmann, 7 Feb. 2011, ch. 21, pp. 313–324.

[263] J. McIlveen, S. Maddock, P. Heywood, and P. Richmond, "PED: Pedestrian Environment Designer," in *Computer Graphics and Visual Computing (CGVC)*, C. Turkay and T. R. Wan, Eds. The Eurographics Association, 2016.

[264] S. Le Grand, "Broad-Phase Collision Detection with CUDA," *GPU Gems 3*, pp. 697–721, 2008.

[265] A. Bleiweiss, "Multi agent navigation on the gpu," in *Games Developpement Conference*, 2009, pp. 39–42.

[266] J. Pan, C. Lauterbach, and D. Manocha, "g-planner: Real-time motion planning and global navigation using gpus." in *AAAI*, 2010.

[267] A. Demeulemeester, C.-F. Hollemeersch, P. Mees, B. Pieters, P. Lambert, and R. Van de Walle, "Hybrid path planning for massive crowd simulation on the gpu," in *Motion in Games*, J. M. Allbeck and P. Faloutsos, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 304–315.

[268] G. Caggianese and U. Erra, "Gpu accelerated multi-agent path planning based on grid space decomposition," *Procedia Computer Science*, vol. 9, pp. 1847–1856, 2012.

[269] F. M. Garcia, M. Kapadia, and N. I. Badler, "Gpu-based dynamic search on adaptive resolution grids," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1631–1638.

[270] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: an efficient and robust access method for points and rectangles," in *ACM Sigmod Record*, vol. 19, no. 2. Acm, 1990, pp. 322–331.

[271] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '84. New York, NY, USA: ACM, 1984, pp. 47–57.

[272] S. Popov, J. Günther, H. P. Seidel, and P. Slusallek, "Stackless KD-Tree Traversal for High Performance GPU Ray Tracing," *Comput. Graph. Forum*, vol. 26, no. 3, pp. 415–424, Sep. 2007.

[273] K. Bolay, "Nichtlineare phänomene in einem fuid-dynamischen verkehrsmodell," Master's thesis, University of Stuttgart, 1998.

[274] R. D. Peacock, J. D. Averill *et al.*, *Pedestrian and evacuation dynamics*. Springer Science & Business Media, 2011.

[275] "Qt project website," https://www.qt.io/, accessed: 2017-03-06.

[276] J. J. Fruin, *Pedestrian planning and design*. Metropolitan Association of Urban Designers and Environmental Planners, 1971.

[277] V. Viswanathan, C. E. Lee, M. H. Lees, S. A. Cheong, and P. M. A. Sloot, "Quantitative comparison between crowd models for evacuation planning and evaluation," *The European Physical Journal B*, vol. 87, no. 2, p. 27, 2014. [Online]. Available: http://dx.doi.org/10.1140/epjb/e2014-40699-x

[278] D. Wolinski, S. J Guy, A.-H. Olivier, M. Lin, D. Manocha, and J. Pettré, "Parameter estimation and comparative evaluation of crowd simulations," in *Computer Graphics Forum*, vol. 33, no. 2. Wiley Online Library, 2014, pp. 303–312.

[279] J. L. Berrou, J. Beecham, P. Quaglia, M. A. Kagarlis, and A. Gerodimos, *Calibration and validation of the Legion simulation model using empirical data*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 167–181. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-47064-9_15

[280] "Cudpp," http://cudpp.github.io/, accessed: 2017-03-08.

[281] T. Kardi, "Microscopic pedestrian flow characteristics: Development of an image processing data collection and simulation model," Ph.D. dissertation, Tohoku University, 2002.