UNIVERSITAT POLITÉCNICA DE CATALUNYA

DOCTORAL THESIS

Crowd Simulation and Visualization

Author:
Hugo Perez

Supervisor:
Dr. Isaac Rudomin Goldberg,
Dr. Eduard Ayguade

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Architecture

in the

Barcelona Supercomputing Center
Computer Sciences

March 18, 2019

I would like to dedicate this thesis to my family in Mexico.

Acknowledgements

I would like to acknowledge Dr. Isaac Rudomin and Dr. Eduard Ayguade for discussing and supervising my thesis work. Thanks to Dr. Vassil Alexandrov for his support and valuable feedback.

I want to express my appreciation to the OmpSs development team and the BSC technical support team for their excellent work.

My especial thanks to Movidius-Intel, and in particular to David Moloney, Jonathan Byrne, and Dexmont Pena, for their important collaboration and support during the research internship in Ireland.

I appreciate the confidence and support of my work team in Mexico: REDDES and SIDEC. Thanks to Dr. Alfredo Careaga for his support in the professional and personal aspect. Thanks to my dear Ola who has endured my fatigue and bad mood. Thanks to my dear friends Agueda and Mario.

I am also grateful to my colleagues who have been part of the research team at some point Benjamin Hernandez, Leonel Toledo, Genoveva Vargas-Solar, Javier Espinosa-Oviedo, Alberto Ochoa, Ivan Rivalcoba, Israel Tabarez-Paz, Sergio Ruiz, Anton Aguilar, Eduardo Cabrera for their collaboration with ideas and stimulating discussions, for their help and support all this time.

Recognition to the institutions that supported this work:: CONACyT doctoral fellowship 285730, BSC-CNS Severo Ochoa program (SEV-2011-00067), CUDA Center of Excellence at BSC, the Spanish Ministry of Economy and Competitivity under contract TIN2015-34557, and the SGR programme (2014-SGR-1051) of the Catalan Government.

UNIVERSITAT POLITÉCNICA DE CATALUNYA

Abstract

Facultad de Informática de Barcelona
Computer Sciences

Doctor of Philosophy in Computer Architecture

Crowd Simulation and Visualization

by Hugo Perez

Large-scale simulation and visualization are essential topics in areas as different as sociology, physics, urbanism, training, entertainment among others.

This kind of systems requires a vast computational power and memory resources commonly available in High Performance Computing HPC platforms. Currently, the most potent clusters have heterogeneous architectures with hundreds of thousands and even millions of cores. The industry trends inferred that exascale clusters would have thousands of millions.

The technical challenges for simulation and visualization process in the exascale era are intertwined with difficulties in other areas of research, including storage, communication, programming models and hardware. For this reason, it is necessary prototyping, testing, and deployment a variety of approaches to address the technical challenges identified and evaluate the advantages and disadvantages of each proposed solution.

The focus of this research is interactive large-scale crowd simulation and visualization. To exploit to the maximum the capacity of the current HPC infrastructure and be prepared to take advantage of the next generation. The project develops a new approach to scale crowd simulation and visualization on heterogeneous computing cluster using a task-based technique. Its main characteristic is hardware agnostic. It abstracts the difficulties that imply the use of heterogeneous architectures like memory management, scheduling, communications, and synchronization — facilitating development, maintenance, and scalability.

With the goal of flexibility and take advantage of computing resources as best as possible, the project explores different configurations to connect the simulation with the visualization engine. The kind of system we are developing has an essential use in emergencies. Therefore, we create urban scenes as realistic as possible, in this way users will be ready to face real events.

Path planning for large-scale crowds is a challenge to solve, due to the inherent dynamism in the scenes and vast space search. A new path-finding algorithm was developed. It has a hierarchical approach which offers different advantages: it divides the search space reducing the problem complexity, it can obtain a partial path instead of wait for the complete one, which allows a character to start moving and compute the rest asynchronously, it can reprocess only a part if necessary with different levels of abstraction.

Within the project, new models of behavior based on data analytics were developed. It was developed the infrastructure to be able to consult various data sources such as social networks, government agencies or transport companies such as Uber. Every time there is more geolocation data available and better computation resources which allow performing analysis of greater depth, this lays the foundations to improve the simulation models of current crowds.

Using and supporting simulations and visualization makes it possible to address real-time dynamic crowd observation and organization. Decision makers can consider the state of the crowd, the way it occupies space and the way individuals form or leave groups. With new technology, it might be possible to configure and reconfigure urban areas dynamically to prevent disasters and to help people to behave

safely in public events.

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

BSC	**B**arcelona **S**upercomputing **C**enter
HPC	**H**igh **P**erformance **C**omputing
GPU	**G**raphics **P**rocessing **U**nit
ABM	**A**gent **B**ase **M**odels
ANN	**A**rtificial **N**eural **N**etworks
SNN	**S**piking **N**eural **N**etworks
SVM	**S**upport **V**ector **M**achine
DAT	**D**ata **A**ssociation based **T**racking
CTM	**C**orrelated **T**opic **M**odel
ORDSS	**O**perations **R**esearch based **D**ecision **S**upport **S**ystems
CPD	**C**ompressed **P**ath **D**atabase
BFS	**B**readth **F**irst **S**earch
LAN	**L**ocal **A**rea **N**etwork

Chapter 1

Introduction

1.1 Introduction

Nowadays there are 502 cities with over one million inhabitants, 74 with more than 5 million and 29 megacities with populations above 10 million; in total, there are around 3.9 billion human beings living in urban areas [1]. As a result, it is crucial to develop computer models that describe the behavior of populations.

Crowd simulations allow safe application of the scientific method and may aid in the analysis of certain subsets of the crowd phenomena, such as disease propagation, building evacuation during emergencies, traffic modeling or social evolution [2], [3]. Prediction before, during and after daily crowd events may reduce risks and and avoid catastrophic situations. Figure 1.1 shows the main criteria for possible applications.

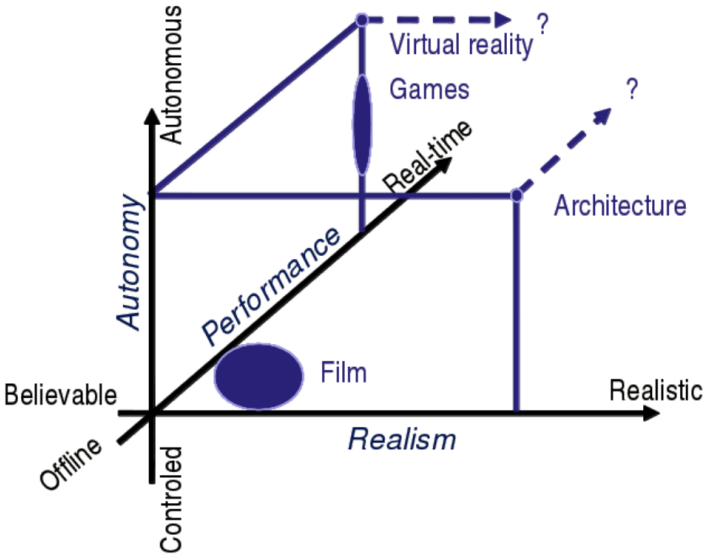


FIGURE 1.1: Possible applications can vary along these three main criteria. From [4]

Simulating crowded places helps architects to design safe public spaces. However, everyday human behavior imply complex simulations, particularly walking in groups, waiting for children or elderly members of the group, or stopping to talk, or even tying shoelaces. Indeed, humans react to stress and crowding in different ways depending on their size, age, gender, and other physical characteristics; on their psychological characteristics; on whether they are alone or in groups such as

family. Their behavior also changes per culture dependent characteristics such as proxemics, and even by how people of different cultures understand instructions. Ignoring these fundamental aspects dehumanizes people.

This kind of systems requires a vast computational power and memory resources commonly available on High-Performance Computing HPC platforms. According to the list of the most potent clusters in the world Top500 [5], it is a common practice the use of heterogeneous architectures combining conventional processors with accelerators. Currently, the most popular accelerators are Intel Xeon Phi and NVIDIA Graphical Processing Units GPU's. At this moment, these clusters have hundreds of thousands and even millions of processors.

Right now, we are in a stage of transition between the petascale and exascale era. The industry trends inferred that exascale clusters would have thousands of millions of cores [6]. The technical challenges for the simulation and visualization process in the exascale are intertwined with difficulties in other areas of research, including storage, communication, programming models and hardware. For this reason, it is important prototyping, testing, and deployment of a variety of approaches to address the technical challenges identified and evaluate the advantages and disadvantages of each proposed solution.

The work done in this thesis is focused on: Large scale crowd simulation and visualization on heterogeneous computing clusters. Represent the most realistic behavior and scenarios based on crowdsourced data.

1.2 Problem Definition

Interactive simulations of hundreds or thousands of agents are currently common, but simulations with hundreds of thousands or millions of agents are still an issue to be resolved [7]. The complexity, realism, and size of the simulation and visualization systems are proportional to the processing capacity. Therefore, the manufacture of increasingly powerful computers raises the requirements and characteristics of these systems.

The project development is focused on:

- Scale crowd simulation and visualization on heterogeneous cluster computing.
- Represent the most realistic behavior and scenarios based on crowdsourced data.

Below the needs of the project in each of these axes.

1.2.1 Scale crowd simulation and visualization on heterogeneous cluster computing.

Due to the good results that can be obtained on a single workstation with GPU's and since it is relatively new the use of GPU's on the scene of HPC, have been little explored the simulation and visualization of crowds on a GPU cluster. However, since the resources of one machine are inevitably exhausted, the development of a scalable system is necessary. The GPU cluster gives us three main advantages, accelerates data processing during simulation, allows us to perform machine learning

algorithms and perform in situ visualization.

The use of a computing cluster has extra inherent processes associated, such as the distribution of work, memory management, communication, and synchronization, plus the use of advanced techniques to harness data locality, overlap computation with communication between others [8]. For this reason, the programming task is slow and difficult.

Programming models are in constant development with the objective of facilitating the development, maintenance and application portability, besides exploiting to the maximum the computing resources. Some of them are CUDA, OpenCL, OpenACC, OpenMP, OpenMP-Superscalar OmpSs, MPI, and others. It is important to know their features, test, compare and even combine them to get the best results.

The system architecture is conditioned by the hardware resources available, so it is important to analyze the advantages that different schemes can provide. For example, perform partial offscreen rendering on different nodes where the simulation is done, or use streaming data with a remote visualization engine. Each system architecture has trade-offs. Priority is given to schemes that allow us to couple the simulation and visualization to allow:

- Coupling the simulation and visualization to allow:
  - To interact with the system.
  - To explore results during runtime.
  - To allow to stop and save partial results.
- To avoid moving data through the memory hierarchy to obtain results more quickly and save resources.
- To distribute the work and maintain load balancing

Interactivity is an indispensable feature for some applications such as video games or disaster prediction, and offers the following advantages:

- The simulation can be stopped, if the results are not as expected, saving time and resources.
- Get results immediately, accelerating the researcher work.

In situ visualization refers to visualize the results on the same machine where the simulation occurs [9]. Therefore allows us to get the results faster and saves energy since data movement from the chip out through memory hierarchies to storage has significant power costs. Besides these advantages, this feature becomes indispensable when the data generated by the simulation are so extensive that you can not store them. For this reason, the visualization must be coupled with simulation to analyze results and save partial data as the simulation progresses [10].

1.2.2 Represent the most realistic behavior and scenarios based on crowd-sourced data.

There is extensive work on modeling the behavior of crowds, in Chapter 2 there is a description of the main simulation models. Most of the algorithms used in these

models have been developed to run on one machine, with conventional processors. If we want to take advantage of the processing power of a GPU cluster, these algorithms must be adapted, and in some cases, they need to be redesigned.

Validate simulation models is difficult due to different factors:

- To validate against real data involves collecting information from hundreds or thousands of people (agents).
- Some scenarios are panic situations that can not be replicated.
- Human behavior is complex and depends on many factors so that the same person may act differently in similar situations.

The work done in the validation of models is limited [4] [11], even the more recent studies only consider small scenarios, with dozens of people. So far, there has been no widely accepted conventions for how to present results and evaluation of a steering algorithm.

For these reasons, it is necessary to study the current models, based on them develop new ones taking their best features. In the development of these new models, the project considers that they should be highly parallelizable and scalable to be executed on a computing cluster.

Also, the emergence of new platforms [12] [13] [14] to collect data from thousands of people around the world on their displacement in different scenarios, gives us the possibility to design and validate new models from these data. In the other hand projects like OpenStreetMaps OSM allows to know the physical characteristics of practically anyplace in the world. Through the participation of volunteers the project gathers geografical information, that later is validated for the same users. This data complemented with other sources like gubernamental institutions allows to represent realistic scenarios.

1.3 Motivation

In 2008 for the first time in humankind history, the number of people living in urban areas was higher than in rural areas. Figure 1.2 shows the urban and rural population from 1960 onwards. Urban settings are a relatively new phenomenon in human history. Over the second half of the 20th century, there was a rapid growth in urban areas. The forecast indicates that the urban population will continue to increase over the next few years above the rural population, as can be seen in Figure 1.3. Most countries are expected to be majority urban.

Big cities attract the rural community because they represent a positive influence on economic activity and culture. The migratory phenomenon added to the global growth in the population, have resulted in huge chaotic cities, where every day there are conditions of risk. It is increasingly common to see situations such as those shown in image 1.4. In cities like Tokyo, Delhi or Mexico City are everyday situations.

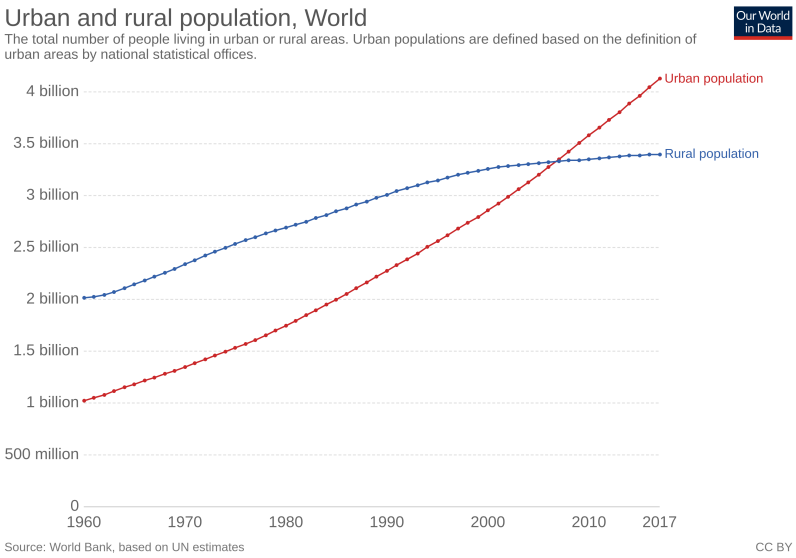


FIGURE 1.2: Urban vs rural population. From [1]

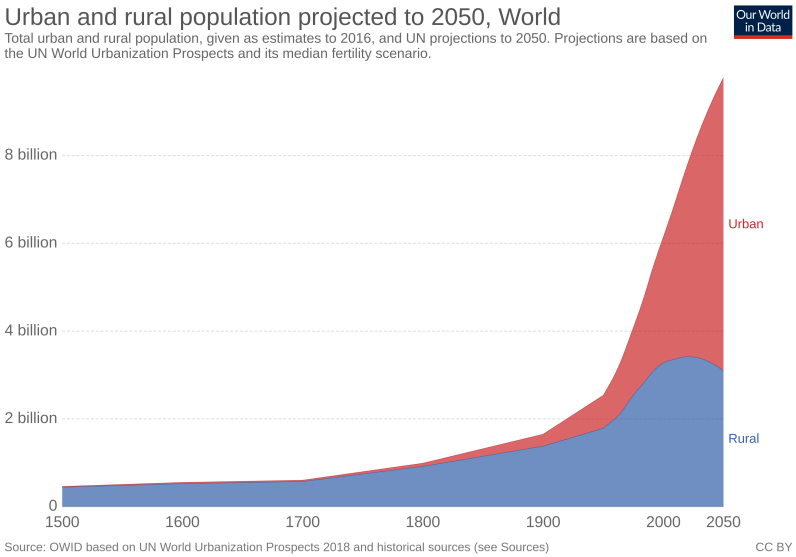


FIGURE 1.3: Urban and rural population projected to 2050. From [1]

It represents a challenge for urban authorities and planners, who need to use technological tools that allow them to monitor activities in the city, with the aim of avoiding risk situations, taking preventive and corrective measures to improve public infrastructure.

These technological tools aim to model the behavior of the population under different scenarios that would be impossible to prove in real situations. The development of simulation and visualization systems that integrate these models allow an analysis before, during and after the events, which enables reducing risks and improving logistics.

(A) Metro in Mexico City, anyday, anytime.

(B) Typical vehicular traffic of a big city.

FIGURE 1.4: Daily crowded events that occur in a city

In this context, data management and visualization techniques can support real-time observation for understanding the behavior of people and thus help in the development of security strategies (e.g., by supporting on-line and post-mortem analytics for guiding the process of recommendation and decision making in real-time). Large scale crowd simulation and visualization combine computer graphics, artificial intelligence and high-performance computing among other areas.

1.4 Research Questions

The complexity of the current hardware architectures of supercomputing equipment makes us question whether it is possible to implement large-scale crowd simulation and visualization on heterogeneous computer architectures and if the simulation can scale efficiently on multiple nodes and provide valuable insights and discovery in real life problems from diverse application areas. The project explores If it is possible to take advantage of the computing resources available in high-performance computing infrastructure considering these characteristics:

- Capable to use heterogeneous architectures as next-generation technology demands.
- Scalable to a cluster computing.
- Make efficient use of computing resources.

Have coincided the availability of large volumes of data describing the behavior of people and the technological tools that allow us to process and analyze this data. Another issue to explore is the use of real data to develop new models of behavior or improve current ones, also, to verify if these data are useful to give realism to the scenes.

1.5 Contributions

The main contributions of this thesis consist of: It was developed a new approach to scale crowd simulation and visualization on heterogeneous computing clusters using a task-based technique. Its main characteristic is that it is hardware agnostic. It abstracts the difficulties that imply the use of heterogeneous architectures like memory management, scheduling, communications, and synchronization — facilitating development, maintenance, and scalability [15], [16].

Visualization transforms results in graphical and color representations that can improve human cognition in data or results' analysis. The selection of a visualization mechanism depends on the simulation and available HPC infrastructure. With the goal of flexibility and to take advantage of computing resources as best as possible, three different configurations were designed to connect the simulation with the visualization engine: streaming, in-situ, and web [17], [15].

A new hierarchical path-finding algorithm was developed which offers significant advantages for crowd simulation: it can return a partial path faster which allows to a character start moving while computing the rest asynchronously, it can reprocess only a part with different levels of abstraction, which is ideal for dynamic scenes. It has a low memory consumption, and it is suitable to improve applying parallelism. We develop new behavior models based on real data applying data analytics and explore different tools to recreate realistic urban scenarios [18]. We investigate how anthropometry and cultural diversity may impact in the behavior of crowds in panic situations [19].

1.6 Organization

The remainder of this document is structured as follows.

Chapter 2 covers the background and state of the art about the main behavior models and programming models used in HPC.

Chapter 3 presents a case of study to analyze how the use of novel programming models can facilitate the development of crowd simulation systems on heterogeneous architectures. Since industry trends in the coming years imply the availability of cluster computing with hundreds to thousands of cores per chip and the use of accelerators, programming presents a challenge due to this heterogeneous architecture.

Chapter 4 presents a task-based approach for crowd simulation using OmpSs, CUDA, and MPI, which allows taking the full advantage of computational resources available in heterogeneous clusters and reduce complexity in its development.

Chapter 5 presents three different configurations to connect the simulation with the visualization engine: streaming, in-situ, and web. We explain the trade-offs associated with each setup.

Chapter 6 presents a new hierarchical path-finding algorithm that reduces the search space, identifying the areas of interest in high levels of abstraction, allowing the tree to be pruned as it is constructed, using a heuristic based on density and distance the algorithm gives priority to the shortest paths in areas with fewer obstacles.

Chapter 7 presents a methodology that allows us to integrate real data into crowd simulation and visualization. The project goal is to represent the most realistic scenarios in a city. Crowdsourced location data is used to compute spatiotemporal people and vehicle flows, while the map and geometric data describe specific real places.

Chapter 8 proposes an approach to explore the impact of anthropometry and cultural diversity in the behavior of crowds in panic situations. The method includes techniques for reproducing and simulating the behavior of the crowd to generate models that can help decision-makers to control such conditions.

Finally Chapter 9 presents the conclusions and future work.

Chapter 2

Background and State of the Art

This chapter provides the context in the principal axes that integrate the project.

2.1 Scale crowd simulation and visualization on heterogeneous cluster computing

A long time ago (2005), our research group led by Dr. Isaac Rudomin, started to explore moving our crowd simulation from the CPU to the GPU. The reason was that simulation performance significantly increased in comparison with desktop CPUs. Allowing to perform larger crowd simulations with high performance at a fraction of the computing cost of a grid or CPU cluster-based alternatives. Another advantage was that massive crowds could be visualized in real time as agent data is already located on the GPU hardware. Our group was one of the first to do that [20], but it started a trend [21], [22]. By 2012 it was required even larger crowds. By now GPU computing is part of heterogeneous supercomputing clusters such as Marenostrum, the supercomputer at Barcelona Supercomputing Center BSC, and the group started studying how to use this kind of hardware for our research. This trend is even more evident today with Summit at Oakridge, as well as the plans for future hardware by supercomputing centers around the world.

The project uses ABM for crowd simulation handling each agent individually and applying general behavior rules in a single-rule-multiple-agents fashion, which suits the GPU's Single Instruction Multiple Data (SIMD) model. Before CUDA and OpenCL programming models, and while precursor GPGPU languages such as Brook and Sh were in active development, GPUs were already being used for general purpose computing (GPGPU). Pioneers in this regard are Rudomin, Millan and Hernandez [20] that through XML scripting defined Finite State Machines to determine the behavior of agents implemented in fragment shaders. This was possible even when neither the GPU hardware architecture nor the programming language was oriented to this type of tasks.

The emergence of CUDA in 2007, and OpenCL in 2008, and a new GPU unified hardware architecture, has allowed developers to take advantage of the GPU resources using a C-like language syntax. Today, there are a vast variety of projects that uses the computing power of the GPU for the simulation and visualization of crowds [23]. Despite the widespread use of CUDA and OpenCL, they have the following drawbacks: they require explicit device selection, data transfers between host and devices, flow control through queues and events. Besides, there is no an implicit way to use the CPU and GPU resources concurrently and effectively; Therefore, programmers need to use OpenMP + CUDA/OpenCL to control each resource within

a node or a cluster explicitly by using MPI for inter-node communication.

The need for combining these programming models makes evident the programming complexity of heterogeneous HPC systems. In an effort to reduce such complexity, the author uses OpenMP Superscalar - OmpSs [24], a task-based programming model developed by the BSC. It uses a set of directives and library routines that can be embedded in high-level programming language code to develop concurrent applications. OmpSs' main goal is to provide a productive environment for application development in modern HPC systems. OmpSs is built on top of the Mercurium [25] a source-to-source compiler, which emits calls to the Nanos++ [26] runtime system. Nanos++ uses the information provided by user annotations to build a task dependency graph dynamically at runtime, which is used to schedule tasks in a data-flow way.

An alternative programming model targeted to reduce the programming complexity of heterogeneous systems is OpenACC. Launched in 2010, OpenACC is a collection of compiler directives used to specify loops and regions of code which execution is offloaded to an accelerator in standard C, C++ or FORTRAN. The advantage offered over OmpSs is the automatic generation of CUDA kernels; however, it still has limited functionality. There is work in progress to implement similar characteristics in OmpSs [27]. OmpSs and OpenACC developers are part of the OpenMP Standard Group; they influence the design of OpenMP. Therefore, the best features of both are integrated into OpenMP as they evolve.

2.1.1 A brief introduction to OmpSs

OmpSs's philosophy is to include different types of accelerators such as Nvidia GPU, Xeon Phi, Mali GPU, etc, and remain open to include future technologies, in this way is the programming model who meets the new architectures not the applications. OmpSs offers the following advantages: Automates scheduling, memory management, synchronization and communication, simplifies syntax and portability.

It is possible to define tasks inline or outline. Listing 2.4 shows the syntaxis. Listing 2.2 shows an example of a task inlined. Listing 2.3 shows an example of a task outlined.

LISTING 2.1: OmpSs Task definition.

```
1 #pragma omp task [ input (...)] [ output (...)] \
2 [ inout (...)] [ concurrent (...)]
3  { function or code block }
```

LISTING 2.2: OmpSs Task definition inline.

```
1 int main ( )
2 {
3   int X[100];
4   #pragma omp task
5   for (int i =0; i< 100; i++) X[i]=i;
6   ...
7 }
```

LISTING 2.3: OmpSs Task definition outline.

```
1  #pragma omp task
2  void foo (int Y[size], int size) {
3    int j;
4    for (j=0; j < size; j++) Y[j]= j;
5  }
6  int main()
7  {
8    int X[100];
9    foo (X, 100) ;
10   ...
11 }
```

The strategy used by the author to translate a sequential algorithm into one task-based parallel, consist in encapsulating the main steps into functions, changing these functions into tasks and defining data dependencies between them. Data dependencies definition is necessary to guarantee data locality and automatic scheduling. OmpSs also allows incremental code parallelization, and thus code testing can be performed during the migration process.

The **target** directive allows using an accelerator by example a GPU. Listing 2.4 shows its syntaxis.

LISTING 2.4: OmpSs syntax to define a task.

```
1 #pragma omp target device ({ smp | cuda | opencl })
2 {[ndrange(workDim, gworkSize, lworkSize)]}
3 [implements( function_name )]
4 {copy_deps | [copy_in( array_spec ,...)] \\
5 [copy_out(...)] [copy_inout(...)] }
6 { function or code block }
```

From Listing 2.4 (code line 1), the **target** directive indicates the device and programming model used for the associated task; the **ndrange** clause, in code line 2, is used to configure the number of dimensions (workDim), the global size of each dimension (gworkSize) and the number of work-items for each workgroup (lworkSize).

Clause **implements** (code line 3) allows the definition of multiple task implementations of the same function, for example, one in the CPU and one in the GPU; the runtime decides which is better during execution and can even execute both in parallel.

Clause **copy_deps** (code line 4) ensures that the device has a valid and consistent copy of the data. Therefore, the programmer does not need to do explicit memory allocations or data transfers between the host and devices. The runtime system is responsible for doing it.

A synchronization point is defined by using **taskwait** (see listing 2.5), all data is copied from the device to the CPU memory. Similarly, **taskwait on**, synchronize the execution and transfers specific variables. With the **no flush** directive synchronization occurs, but data is not transferred to CPU memory.

LISTING 2.5: Synchronization point definition.

```
1 #pragma omp taskwait [on (...)] [noflush]
```

Using OmpSs with multiple GPUs offers the following advantages:

- It is not necessary to select the device for each operation.
- There is no need to allocate special memory in the host (pinned memory) or the device.
- It is not necessary to transfer data between host and devices, OmpSs maintains memory coherence efficiently and reduces transfers using data regions.
- Programmers do not need to declare streams, OmpSs implements them by default.
- All available devices (GPUs) are used by default, therefore, the application scales through all the devices.

OmpSs does all these operations automatically. There are different runtime variables to control how OmpSs allocates resources:

- NX_SMP_WORKERS sets the number of CPU cores used for computation. All available CPU cores are used by default.
- NX_GPUS sets the number of GPUs used for computation. All available GPUs are used by default.
- NX_GPUPREFETCH sets the number of CUDA streams used for computation. One CUDA stream is used by default.
- NX_GPUOVERLAP overlaps communication with computation when active. Active by default.
- NX_GPU_CONCURRENT_EXEC executes kernels concurrently when active. Active by default.

2.2 Represent the most realistic behavior and scenarios based on crowdsourced data.

Pedestrians have been empirically studied for more than four decades [28], [29]. Initial methods were based on direct observation, photographs, and time-lapse films to develop a level-of-service concept [30], design elements of pedestrian facilities [31], [32], [33], or planning guidelines [34]. Most research about panic was carried out by social psychologists. It has been of empirical nature (e.g., [35], [36], [37], [38]).

Simulation models have been proposed for addressing crowd dynamics modeling, route choice behavior of pedestrians, and emergency and evacuation situations. Experimental efforts have revealed quantitative details of pedestrian interactions, which have been modeled by mathematical equations. The analysis of crowds using mathematical and simulations lead to important conclusions for developing logistics strategies. For example, self-organized patterns of motion demonstrate that “intelligent” collective dynamics can be based on simple local interactions; under extreme conditions, coordination may break down giving rise to critical crowd condition such as the so-called “freezing-by-heating” and “faster-is-slower” effects, but also the transition to “turbulent” crowd dynamics.

This section describes existing works according to four perspectives. Techniques used for observing high-density crowds which it is understand as possible data harvesting methods 2.2.1. Simulation of crowds that apply mathematical and artificial intelligence methods for reproducing the way crowds evolve under different conditions 2.2.2. Finally, humanitarian logistics that integrate existing methods into systems that support decision making 2.2.3.

2.2.1 Observing High-Density Crowds

The computer vision community has started addressing different research problems related to crowds. Existing approaches focus on crowd detection, and detection and tracking of individuals in a crowd. These approaches have mainly used videos to identify the volume, shape, and color of individuals [39], contours [40], trajectories [41], or to count the number of people in the crowd [29]. However, it is still an open issue due to the high number of characters, significant occlusion problems and the combination of motion directions in a scene.

Ali et Al. [42] addresses this problem by modeling moving crowds as aperiodic dynamical systems manifested by a time-dependent flow field. A flow field in a general scene consists of regions with qualitatively different dynamics, reflecting the motion patterns emerging from the spatiotemporal interactions of the participants between each other and with the physical world [43].

There are approaches to classify a crowd concerning density. In a scene with a sparse or moderate crowd, most pedestrians can be fully observed and detected. Persons are then tracked by combining detections into tracklets and associating the tracklets into long trajectories [31], [44].

In [45], [46], [47] it is proposed the Data Association based Tracking (DAT) algorithm, which is extended by using shape and appearance models [48], body-part detectors [49], or a boosting algorithm to train the parameters [50]. However, in a denser crowd where the pedestrians are heavily occluded by the others, frame-based detection is highly unstable, and the identification of trajectories is very expensive.

To address this limitation, it is possible to use local feature points to track individual within a crowd. Brostow [41], Li [47], and Sugimura [51] were among the first to do this. They assume that the feature points belonging to the same person are close in space and their motion exhibit high correlation over time. The most important limitation of this approach is that targets moving together with the same speed cannot be identified as separate persons, whereas a local body movement is usually wrongly estimated as a different target.

Optical flow algorithms have been widely used for tracking pedestrians. Initial approaches assume that optical flow on the target is uniform, and the target is tracked by computing the mean flow around the target location [52], [53]. Some approaches use foreground-background segmentation to get a precise target region. In crowded scenes, targets are frequently occluded, and their region changes over time. Thus, the target location does not change linearly. This requires a propagation model that adaptively changes over time and space with non-linear behaviors like the solutions proposed by Rodriguez [40] and Kratz [54].

In extremely dense crowd situations, the most promising tracking algorithms use motion information in local areas. For example, Ali and Shah [42] assume that pedestrians in the crowd behave like particles in the flow. This assumption leads to a solution only able to track pedestrians moving in the same direction as the crowd. Rodriguez [40] divides the video into short clips, and for each local area in the clip, flow vectors are quantized into four categories based on the direction where vectors are heading. The quantized clips are trained within the Correlated Topic Model (CTM), generating a set of topics. For each new frame, a probability distribution over the topics is measured, and the probability of the motion is then derived. The new target position is estimated as a combination of the observation and the tracker prediction.

2.2.2 Simulating Crowds

Crowd simulation can be expressed at two different levels: macroscopic and microscopic. At the macroscopic level, the crowd is modeled as a whole, i.e., global interactions with the environment and the crowd itself. On the other hand, crowd simulation at the microscopic level exposes the interactions of the individuals that integrate the group. Through the progressive assembly of agents into crowds, expect them to result in global-scale behaviors.

A first macroscopic modeling approach that seems suited to reproduce spatiotemporal patterns of motion was proposed by Henderson [55], and Hughes [56], who conjectured that pedestrian crowds behave similarly to gases or fluids. Realistic gas-kinetic or fluid-dynamic theory for pedestrians contain corrections due to their specific interactions (i.e., avoidance and deceleration maneuvers).

The project approach for crowd simulation is Agent-Based Models (ABM). ABM's are computational models that simulate the actions and interactions of autonomous agents; it handles each agent individually and apply general behavior rules in a single-rule-multiple-agents fashion.

The "social force model" [57], [58], is maybe the most well-known of these models. This method operates on crowds composed of basic pedestrians obeying physics-based laws. The agents' behavior is based on forces, called social forces. These forces, applied to a pedestrian, can be attractive acting as their destination, or repulsive such as obstacles or strangers. However, it is also possible to introduce other social interactions just as the presence of friends or storefronts, which are modeled through attractive forces that fade over time. It is then possible to give the pedestrians some more decision-making abilities.There are also cellular automata for pedestrian dynamics models [33], [59], [60] and AI-based models [61], [62]. Below a brief description of the main behavior models based on AI.

Reynolds [63] proposed one of the first solutions for large groups of entities with emergent behavior being an extension of particle systems. It is based on three basic rules: separation, alignment, and cohesion. These rules maintain together, in a direction and free of collisions a group of boids or bird-like objects.

The Predictive/Velocity-based models [64] propose methods where agents compute

the set of velocities that lead to a collision with an obstacle. To move on routes without collision, agents choose velocities outside this domain. This concept is expanded by Van den Berg [65], [66], introducing the notions of Velocity Obstacles (VO) and Reciprocal Velocity Obstacles (RVO) and the notion of Optimal Reciprocal Collision Avoidance (ORCA).

In the everyday exercise of controlling their locomotion, human beings rely on the visual information obtained from the perception of the environment to achieve collision-free navigation. Cognitive science allows extracting relatively succinct information from perception to reach safe locomotion, e.g., using the distance of obstructions in candidate lines of sight, pedestrians apply simple heuristics to adapt their walking speeds and directions. Based on these observations Pettre propose Synthetic Vision models [67], [68]. These models predict individual trajectories and collective patterns of motion in good quantitative agreement with a large variety of empirical and experimental data.

When observed, the individuals in a crowd seem to adopt an "automatic behavior" that can be interpreted as the result of a learning process based on trial and error [69]. For example, pedestrians have a preferred side of walking since an asymmetrical avoidance behavior turns out to be profitable [50], [69], [58]. Claim that the formation of a behavioral convention can be described using evolutionary game theory. Based on quantitative measurements studies have shown that the behavior in conflict situations can be described by a superposition of forces [70]. In this sense, Lewin [71] proposes a mathematical model based on the idea that behavioral changes are guided by social fields or social forces [58], [57].

Rudomin et al. [23] describes advanced methods for simulating, generating, animating and rendering crowds such as encoding neighbor information on world maps, i.e., in the environment rather than the agents and thereby reduces the problem of determining neighbor contributions to O(n). It discusses local collision avoidance methods that use truncated Voronoi diagrams followed by a ray marching technique over a radius of vision to sense the viewing area in the direction of the agent.

Based on scatter and gather operations, describe the development of two data-parallel techniques for proximity queries that are suitable for simulating thousands of agents in real time with accurate collision avoidance. They first perform proximity queries in the graphics hardware, followed by some method of collision avoidance (Reynolds, Helbing, RVO, Synthetic Vision or any other heuristics).

The scatter-based method supports the idea that an agent should be able to find its closest neighbors with only one texture fetch. To accomplish this, each agent paints in a world map an area where it is visible. Each pixel in the environment map holds a list with the IDs of neighbor agents. A Layered frame buffer (LFB) is used to generate the nearest neighbor lists and render the environment map for the scatter-based technique. Contrary to the scatter-based, in the gather-based, each agent only writes its ID in the corresponding cell in the world map. To find their neighbors, every agent scans over an area within the world map. In this way, each agent generates its neighbor list. Avoiding the need to create an LFB.

2.2.3 Humanitarian Logistics

Humanitarian logistics specializes in organizing supplies during natural disasters or complex emergencies to affected areas. During these events, technology is a crucial factor to achieve better results. For instance, by implementing tracking systems, and humanitarian logistics software, it is possible to provide real-time supply-chain information. It enhances decision making, increasing the quickness of the relief operations and achieving better coordination.

Existing humanitarian logistics systems are operations research-based decision support systems (ORDSS) for crowd management, which employ a range of tools from operations research, analytics, and crowd dynamics [72]. At its core, such systems implement a scheduling tool and a real-time video tracking system. The video tracking system measures and balance infrastructure utilization. The ORDSS provides extensive real-time reports to support pilgrim flows.

2.3 Current Platforms

Crowd simulation and visualization is a very active research field which has led to the development of commercial platforms used mostly for the simulation of massive events, film scenes, video games, among other purposes. They are designed to run on a single workstation, so the size of the simulation (the number of agents) and its complexity is limited by the memory and processing capacity. Some examples of such systems are:

- Legion [73] uses speed, density, and space utilization maps to qualitatively and quantitatively analyze the use of space over time. Legion oversimplifies the behavioral representation of individuals by using four parameters and one decision rule based on the least effort to simulate individual behaviors.
- Massive [74] was originally developed for its use in The Lord Of The Rings film trilogy. Subsequently, it was adopted as a standard tool for film and TV productions around the world. It provides crowd-related visual effects and autonomous character animation.
- Golaem Studios [75] develops character animation software for VFX artists, animation and game studios that need to populate backgrounds and midgrounds for films, TV series, commercials, and games.

The development of a scalable system is necessary to overcome memory and processing limitations. Below there is a description of projects, some designed to run on clusters, others are research oriented or provide open source modules and libraries.

- TRANSIMS (Transportation Analysis SIMulation System) [76] is an activity and agent-based model. It creates an entire region and simulates the second-by-second movement of all travelers and vehicles. It has a visualization module called TransimsVIS and a GUI TRANSIMS RTE; it is developed by the Transportation Research and Analysis Computer Center (TRACC) in the USA.
- FLAME (Flexible Large-scale Agent Modelling Environment) [77] is a framework to develop agent-based simulations which automatically produces parallelizable code to execute on different parallel hardware architectures. FLAME-GPU was developed as an extension of FLAME. FLAME-GPU is designed to

be executed in a workstation and is used to accelerate the simulation using the Graphic Processing Unit GPU. Both are developed by Sheffield University.

- Pandora [78] is an open-source framework designed to provide a simulation environment for social scientists. It includes a C++ and Python environment that splits the workload of simulation across computer nodes using MPI and also uses multiple cores relying on OpenMP. It is developed by the BSC.

These projects are designed to run on CPU clusters and do not take advantage of accelerators. Current research on crowd simulation has shown excellent results can be obtained on a single workstation with GPUs. For this reason, FLAME and Pandora frameworks are starting to take advantage of them. However, the use of accelerators in HPC is relatively new, and the simulation and visualization of crowds on GPU clusters need to be explored further.

Crowd simulation and visualization have also been implemented on cloud computing facilities using techniques such as MapReduce and tools like Hadoop [79]. The number of simulated agents is lower than HPC based crowd simulations due to restrictions on computing power and communication latency. Nevertheless, it offers advantages such as resource elasticity and decentralized computation. In the future, mixed implementations combining HPC and cloud computing facilities should be considered to ensure system availability.

Chapter 3

Crowd Simulation on Heterogeneous Architectures

3.1 Introduction

Industry trends in the coming years imply the availability of cluster computing with hundreds to thousands of cores per chip and the use of accelerators. Programming presents a challenge due to this heterogeneous architecture. Therefore, using novel programming models that facilitate this process is necessary.

In this chapter, it is analyzed and compared the use of two programming models in a crowd simulation case study: OmpSs and CUDA. OmpSs allows to take advantage of all the resources available per node by combining the use of CPU and GPU while taking care of memory management, scheduling, communications and synchronization automatically. Experimental results obtained in one node of Minotauro Cluster at Barcelona Supercomputing Center with Tesla K80 GPUs are presented.

Crowd simulation is usually expressed at macroscopic and microscopic levels [2]. Macroscopic modeling describes global interactions with the environment and the crowd itself. On the other hand, microscopic level exposes the interactions between individuals within a group; each agent should be processed individually to simulate a crowd.

The project focuses in modeling crowds at the microscopic level using Agent-Based Models (ABM). An ABM simulates the actions and interactions of autonomous agents; it promotes the progressive assembly of agents into groups resulting in global-scale behaviors [80].

The rest of the chapter is organized as follows: Section 3.2 presents an overview of the algorithm, Section 3.3 shows the implementation, Section 3.4 presents the results obtained from our case study. Finally, Section 3.5 presents the summary of this chapter.

3.2 Algorithm Overview

Several behaviors can be observed within a crowd, one of them is wandering behavior where agents move randomly in a space avoiding collisions against obstacles or other agents. The author has chosen this behavior for its simplicity as it allows us to introduce with ease the reader to crowd simulation using task-based parallelism. A

naïve implementation of wandering behavior has

$$O(N^2) \tag{3.1}$$

complexity, i.e., each agent position must be compared with all the remaining positions to detect collisions. Current techniques reduce the complexity of the algorithm using geometrical or hierarchical approaches [23], [81]. In this case, the complexity of the algorithm is reduced by partitioning the navigable space into a grid that cut down the search space. Additional reduction of the search space is done by defining a search radius for each agent, thus complexity decreases to:

$$O(\frac{N}{t} * r)$$

Where N is the number of agents, t is the number of tiles in the grid, and r is the search radius.

The algorithm uses three vectors, the first one stores agent's position, speed and direction. The second one contains a unique agent identifier and the tile to which the agent belongs. The third one stores the navigation space grid, where each cell has the agent id if it is occupied or zero if it is free.

It is important to emphasize that crowd simulations at microscopic level, interactions between the agents are local, i.e., an agent just perceives and avoids nearby obstacles, in contrast to n-body gravitational problems, where all interactions between all entities need to be considered. For our case study, the navigation space is discretized using a grid. See figure 3.1. Algorithm 1 shows the sequential version. Initial values for position, direction, and speed for each agent are set up. Before update agents position, are checked the cells around in a radius of 5 cells. Evaluation starts in the agent's motion direction and switches every 45 degrees covering a total of eight directions counterclockwise. See figure 3.2. Then, the agent is moved in the direction in which there are fewer obstacles.

Algorithm 1: Crowd simulation sequential algorithm

```
1 Agents_initialization;
2 for i ← 0 to Iterations do
3 |   Update_agent_position;
4 end
```

Processing these tasks in parallel requires tiling and stencil computations. First, the navigation space and agents information is divided into tiles. Once the world is divided into tiles, stencil computations are performed. Then, the tasks performed in each tile can be executed in parallel by either a CPU or GPU, see figure 3.3.

In this case, the calculation of the agent's new position requires information about its surrounding cells. While agents inside each zone have all the necessary information, agents that are on the borders do not have such information. Therefore it is required to exchange it with neighboring tiles. The size of the area to exchange between tiles is defined according to the radius that the agents use to avoid collisions. To minimize the exchange of data between tiles, only the information of occupied cells is transferred. When an agent crosses a border and moves from one tile to another, the agent's information is sent to the new tile.

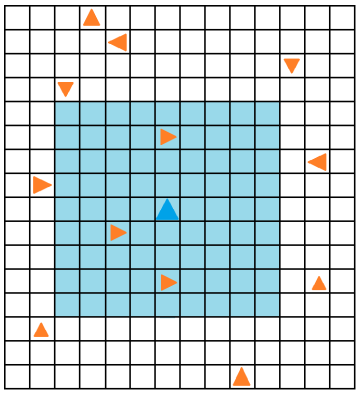


FIGURE 3.1: Navigation space discretization.

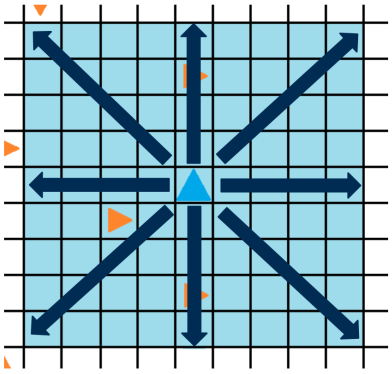


FIGURE 3.2: Search radius for collision avoidance.

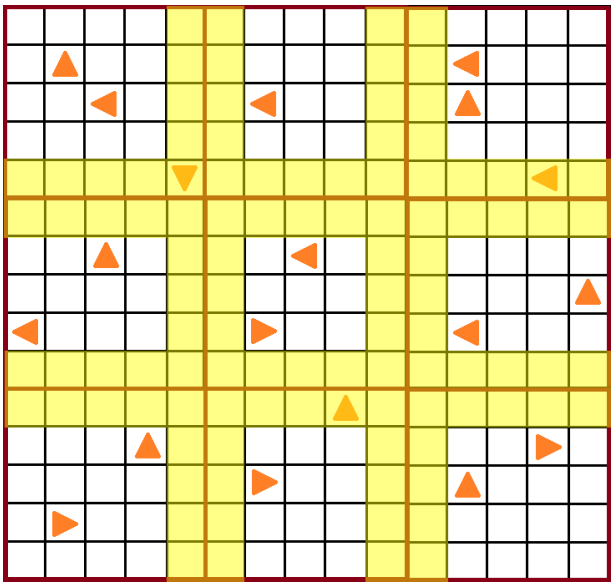


FIGURE 3.3: Dividing the domain in tiles

Algorithm 2 indicates the main functions for the parallel version. First, the buffers to exchange information between tiles are defined (agents information and world cells status). Since the space navigation is divided, the system manages local

coords inside each tile besides the global coords. For this reason, it is necessary to compute an offset for each tile to switch between global and local coords. It is configured a communication topology between the neighbors for each tile. The number of tiles is variable; we test with 4, 9 and 16 tiles. The neighbors of each tile are assigned regarding the total number of tiles. Then equal number of agents in each tile are initialized. the halos are filled with the data about the ocuppied cells to exchange navigation space information *world_halos* with the other tiles. Then, iteratively the system exchanges the world_halos with the neighboring tiles, and update the agent's position. During this step, the system identifies if an agent goes to a diferent tile according to his new position. The system fills the *agents_halos* with the data about agents which move to another tile. Exchange with the neighbors the agents_halos, fill the world_halos and repeat.

Algorithm 2: Crowd simulation parallel algorithm

```
1  Setup_exchanged_data_buffers();
2  Compute_offset();
3  Configure_topology();
4  Agents_initialization();
5  Fill_world_halos();
6  for i ← 0 to Iterations do
7      Exchange_world_halos();
8      Update_agent_position();
9      Fill_agents_halos();
10     Exchange_agents_halos();
11     Fill_world_halos();
12 end
```

The author uses a simple case study because the objective is to explore how to parallelize and subsequently scale a simulation in a heterogeneous computing cluster. For this reason, it has certain limitations like it does not resolve conflicts when two agents want to move to the same cell. To solve this issue, there are different techniques by example use atomic operations on CUDA or alleviate this issue using the distance to the camera to resolve conflicts just for agents near to her. However the use of these techniques are out of the scope of our case study.

3.3 Implementation

In this section, it is described the implementation of a crowd simulation using two programming models CUDA and OmpSs and present a comparison between them.

3.3.1 CPU Version

The following describes how the algorithm shown above is parallelized in the CPU. As a starting point, the CPU sequential version of the algorithm was used. Translating the sequential algorithm into one based on tasks consisted in encapsulating the main steps into functions, turn these functions to tasks and define data dependencies between them. Data dependencies definition is important so the Nanos runtime system can use data locality for automatic scheduling. Also, OmpSs allows us to parallelize a sequential code incrementally and preliminary test it on each new task.

As an example Listing 3.1 illustrates the OmpSs-CPU code to convert updatePosition function to a task. This task will read and update some variables; thus, inout clause is used. The agent's vector is read and updated with the new agent position; the ids vector is read to query the agent id and the current tile it belongs to. Then, the ids vector is updated with the new tile according to the new agent position. Finally, the world vector is updated to mark the grid cells as free or busy according to the agents' movement. All the tiles are executed in parallel inside the for loop.

LISTING 3.1: Conversion to a task of updatePosition function

```
1  //Task definition
2  #pragma omp task inout(
3  ([agents_total_buffer]agents)[0;current_agents],
4  ([agents_total_buffer]ids)[0;current_agents],
5  [world_cells_block]world )
6  void updatePosition(
7      float4 *agents,
8      float4 *ids,
9      int *world,
10     ...) { ... }
11
12 //Task execution
13 bool runSimulation( )
14 { ...
15 for ( int i = 0 ; i < num_tiles ; i ++)
16     updatePosition(agents[i], ids[i], world[i], ...);
17     ...
18     #pragma omp taskwait
19     ...
20 }
```

Another OmpSs feature is its ability to maintain memory consistency only for a particular region of data, e.g., the agents' vector could have 100 elements (agents_total_buffer = 100), but may operate just on the first 10 (current_agents = 10), then OmpSs guarantees memory consistency over these 10 elements. This makes memory management more efficient.

Profiling and analysis of the previous code were done using Extrae [82] and Paraver [83], respectively; the next traces were generated using these tools. OmpSs allows to do a flexible use of resources, it is possible to define the number of CPUs to use in each execution, through a runtime variable. Figure 3.4 shows an example trace of the simulation using eight threads. Notice from Listing 3.1 only the updatePosition function was converted into a task. With only one task, it is easier to understand this trace. On the other hand, Figure 3.5 shows the execution of the simulation using twelve threads; in this case, all the functions were already converted into tasks and were scheduled automatically by the runtime system.

3.3.2 CUDA First Version

The first version of the simulation using CUDA (CUDA1GPU-V1) implements the algorithm described earlier with no domain decomposition and only uses one GPU. All data is transferred from main memory to the GPU memory at the beginning and

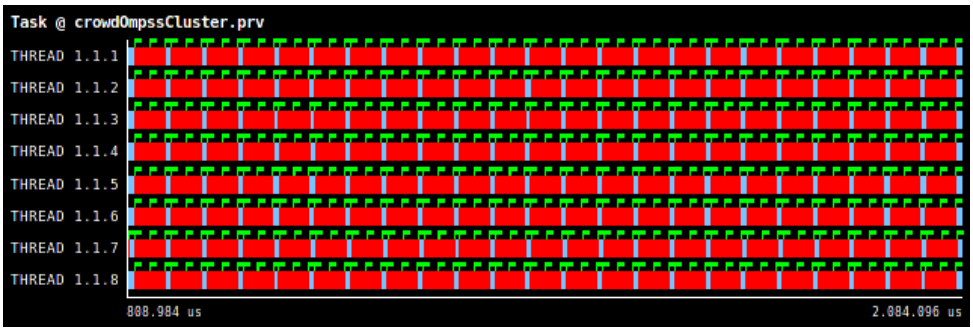


FIGURE 3.4: Simulation executed with eight threads and only one task

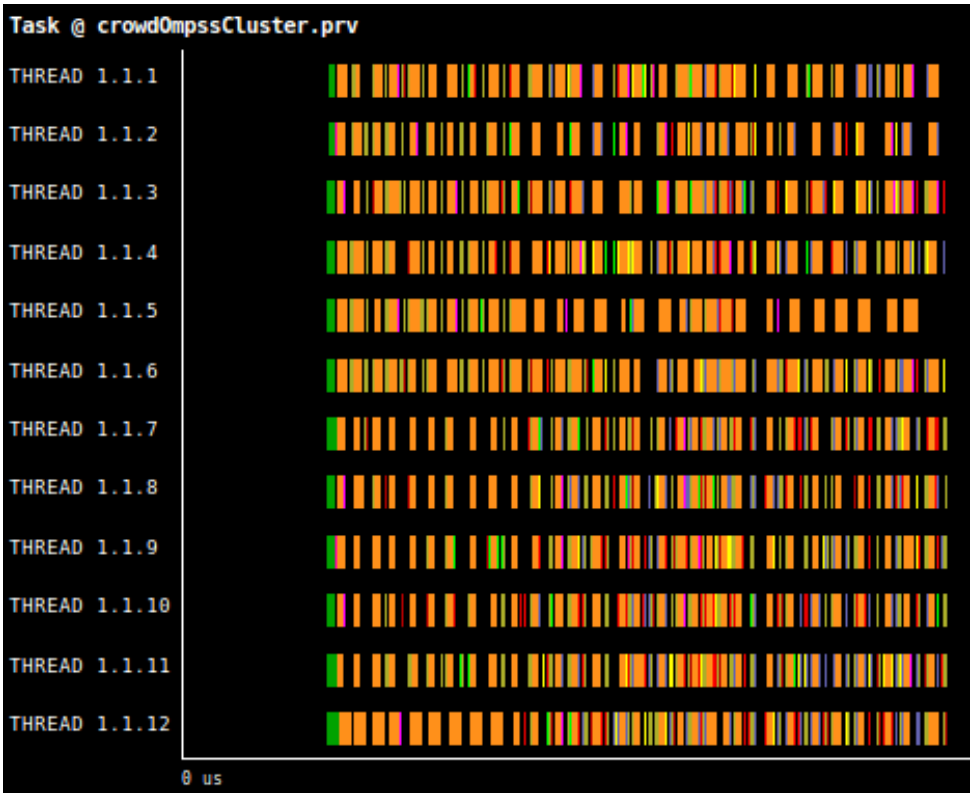


FIGURE 3.5: Simulation executed with twelve threads and all functions converted into tasks

at the end of the execution. This version is used as a reference to analyze how additional steps that involve dividing the domain affect the simulation performance. Preliminary results showed a speedup of 83x when compared to the sequential version of the algorithm. They are good results. However, the number of characters are constrained to the size of the GPU memory.

3.3.3 CUDA Second Version

The CUDA1GPU-V1 code was updated, so domain decomposition of the simulation was possible. This new code (CUDA1GPU-V2) supports larger problems even if data does not fit in GPU memory and allow to use more than one GPU. In this case,

a simulation step consists in transferring a subset of the data from CPU to GPU, then the GPU computes new positions and transfers the results back to CPU which sends a new data subset to the GPU to be computed. These steps are repeated until all the data is processed. A speedup similar to the CUDA1GPU-V1 code is achieved by using CUDA streams to process kernels in parallel and overlap communication with computation. The additional steps performed by CUDA 1GPU-V2 code when compared to the CUDA 1GPU-V1 code are:

- Create a data structure to manage the data in the GPU.
- Allocate special host memory (pinned memory) to handle more than one CUDA stream.
- Split the data between the GPUs.
- Create streams to control the flow of operations.
- Select the GPU in each operation.
- Synchronize operations.

3.3.4 OmpSs GPU Version

CUDA 1GPU-V2 code is taken as a basis for the development of the OmpSs-GPU code. An overall view of the OmpSs-GPU code is illustrated in Listing 3.2. In this case target directive indicates the updatePosition task will be executed by the GPU using the CUDA 1GPU-V2 kernel. OmpSs will take care of the memory management according to directive copy_deps. There is an explicit definition of data dependency for variables agents, ids and world, which are the input and output variables of the task. Task execution is done just by calling updatePosition as it was done for the OmpSs-CPU code.

LISTING 3.2: Converting updatePosition function into a task to be executed as a CUDA Kernel

```
1  // Kernel definition as a task
2  #pragma omp target device (cuda) \
3  ndrange(2, (int) sqrt(count_agents_total), \
4  (int)sqrt(count_agents_total), 8, 8) copy_deps
5  #pragma omp task inout( \
6  ([agents_total_buffer] agents ) [0; current_agents], \
7  ([agents_total_buffer] ids ) [0: current_agents], \
8  [world_cells_block] world )
9  extern C global void updatePosition
10     float4 *agents,
11     float4 *ids,
12     int *world,
13 ...);
14 // Kernel execution
15 bool runSimulation ( )
16 {
17     ...
18     for ( int i = 0 ; i < num_tiles; i ++)
19     { ...
```

```
20            updatePosition(agents[i], ids[i], world[i], ... );
21          ...
22      }
23  #pragma omp taskwait
24      ...
25  }
```

Figure 3.6 shows a trace illustrating how OmpSs manage the hardware resources. In this case, full node allocation is done using twelve CPU threads where Thread 1.1.2 and 1.1.3 are used to control the GPUs. As mentioned earlier, memory management, scheduling, communications and synchronization are done automatically.

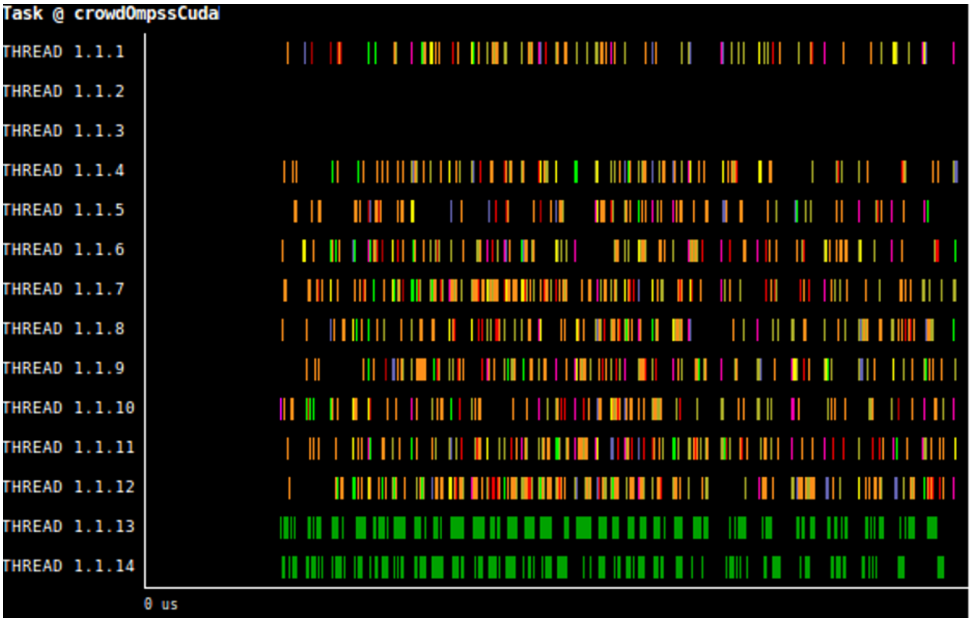


FIGURE 3.6: Thread 1.1.2 and 1.1.3 are CPU threads in charge of one GPU each one. Thread 1.1.13 and 1.1.14 are GPU threads. CPU Threads and GPU threads are executing tasks in parallel

3.4 Results

The test was performed on one node of the Minotauro Cluster at Barcelona Supercomputing Center. Each node has two processors Intel Xeon E5-2630 with 8 cores each processor. With 128 of RAM memory. They also have 2 NVIDIA K80 GPU cards. Running Linux operating system. The software used was: CUDA 7.5, mercurium compiler version 2.1.0, and nanos runtime system version 0.14.1. The speed-up is calculated concerning the sequential version performance. The tests show how different versions perform with a variable number of agents and tiles in a world discretized in 27852x27852 cells.

Figure 3.7 show all versions. From this figure, the author highlights general trends. CUDA version 1 achieves greater speed-up with few agents. Because domain decomposition implies additional work, that is not compensated with a small number

of agents. However, it is necessary to divide the navigation space to use more than one GPU and compute a greater number of agents, not limited to the memory size of the GPU. From four million the system gets better speed-up with CUDA version 2. The system gets better speed-up using just CUDA than CUDA and OmpSs, because there is an overhead involved in creating OmpSs' tasks. However, the difference in the speed-up decreases according to the number of agents. The number of blocks influences performance. Below the author analyzes in-depth every version.

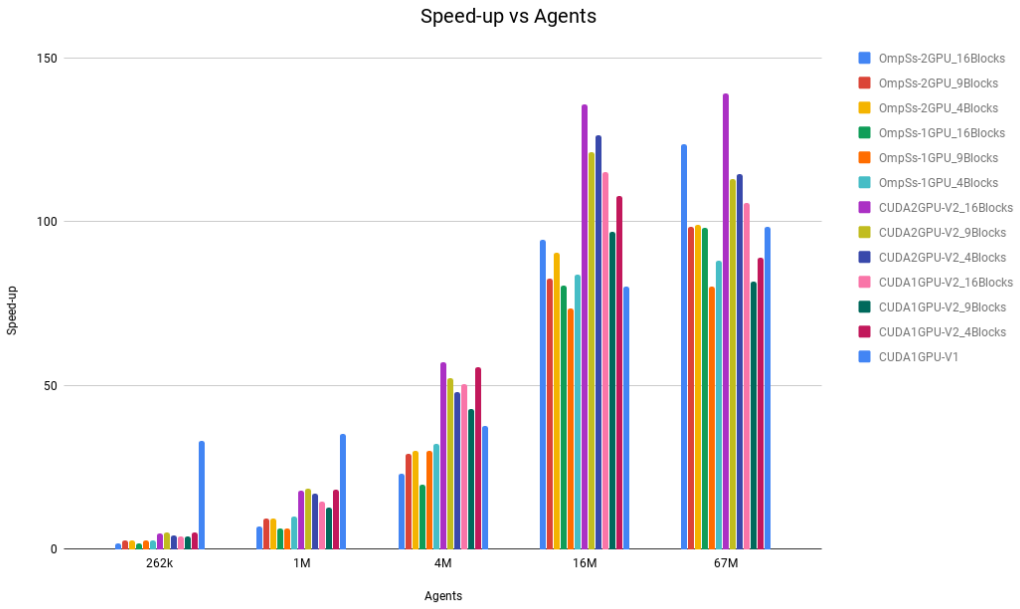


FIGURE 3.7: This graph shows the speed-up for all our experiments.

Figure 3.8 shows the speed-up for CUDA version 2, using one and two GPUs with three different number of blocks. The performance is similar using one or two GPUs until 4 million agents, from 16 million using two GPUs get better results. For one GPU using four blocks is the best configuration until four million agents, for greater quantities better results are obtained using 16 blocks. Using two GPUs get similar results indifferent of the number of blocks for a small number of agents. From four million the system gets better results using 16 blocks. The trend is a greater number of blocks for a greater number of agents.

Figure 3.9 shows the speed-up for OmpSs-CUDA version using one and two GPUs. Similar to CUDA there is no advantage of using two GPUs for a small number of agents. From 16 million it starts to scale, for 67 million is clear two GPUs, and 16 blocks achieve the best results.

Figure 3.10 shows a comparison in a speed-up for OmpSs vs CUDA using one GPU. As it was mentioned before the use of OmpSs implies an overhead that is not

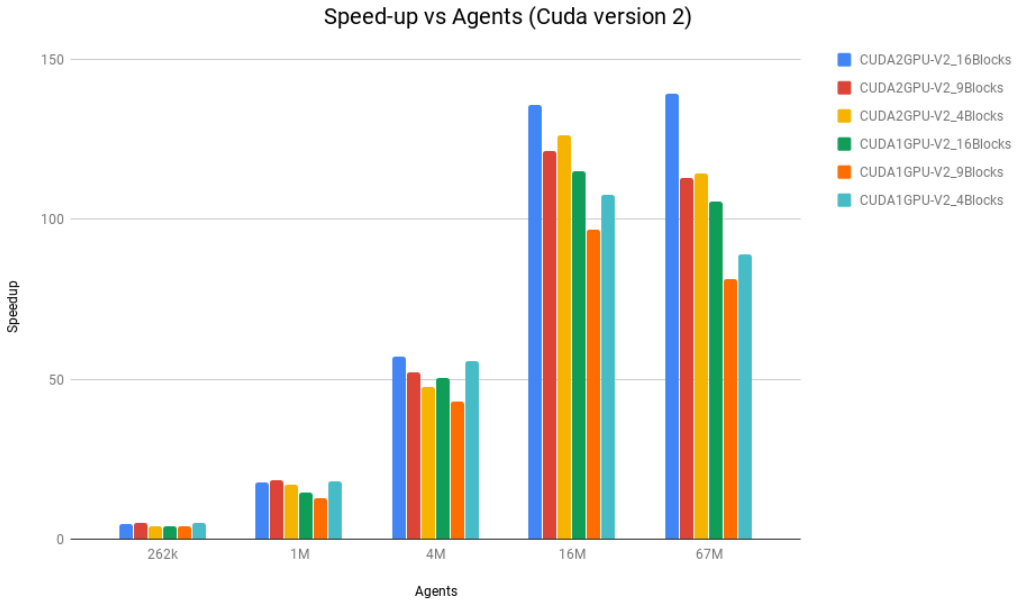


FIGURE 3.8: This graph shows a comparison for CUDA version 2 using one and two GPUs.

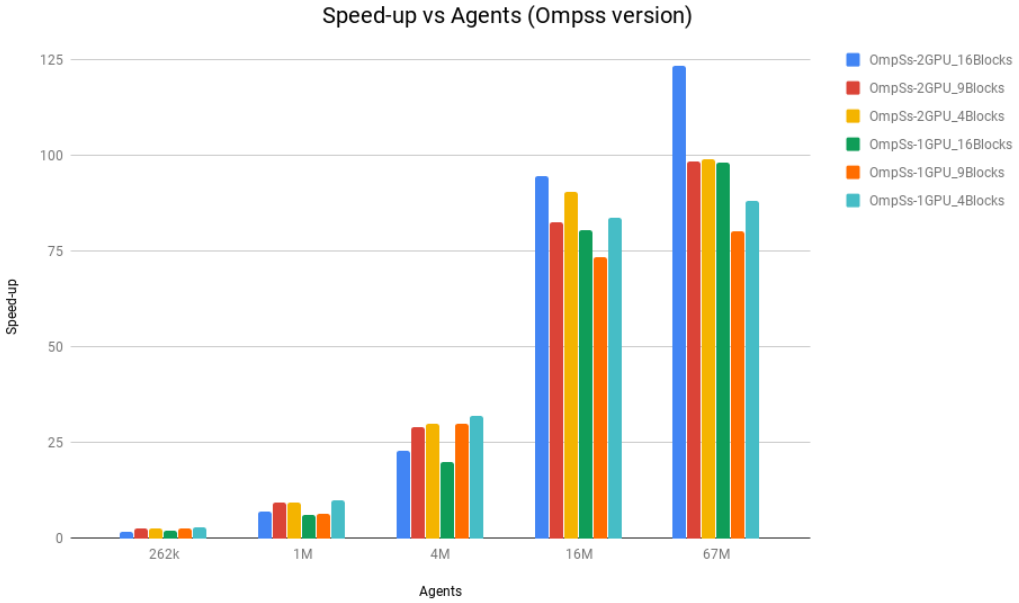


FIGURE 3.9: This graph shows the speed-up for OmpSs-CUDA version using one and two GPUs.

compensated with a small number of agents. However, the difference in performance reduces according to the number of agents.

Figure 3.11 shows a comparison in speed-up for OmpSs vs CUDA using two

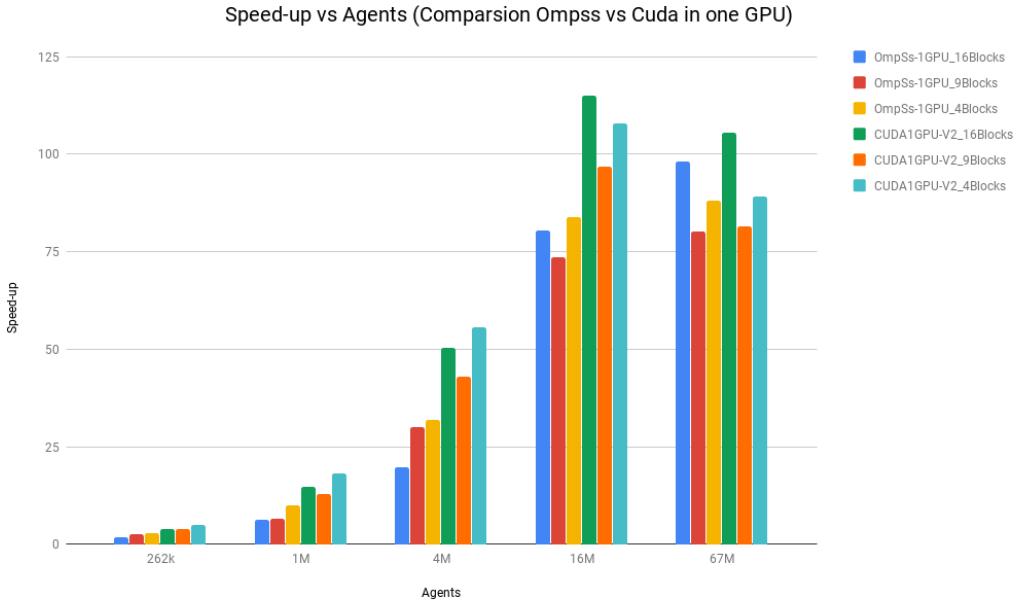


FIGURE 3.10: This graph shows a comparison in speed-up for OmpSs vs CUDA using one GPU.

GPUs. The trend is the same with two GPUs the difference in performance reduce according to the number of agents. There is a trade-off between speed-up and the benefits of using OmpSs.

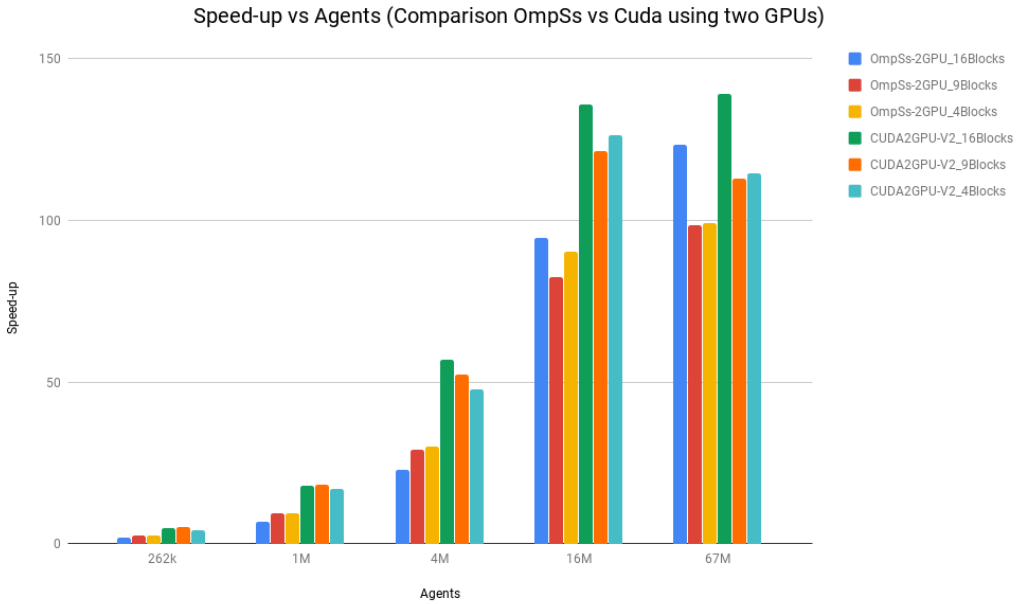


FIGURE 3.11: This graph shows a comparison in speed-up for OmpSs vs CUDA using two GPUs.

3.5 Summary

In this chapter, it was described an algorithm for an interactive crowd simulation using a task-based approach using OmpSs and CUDA. OmpSs allows to parallelize the algorithm incrementally and perform partial tests of each function being converted into a task. OmpSs takes care of memory management, scheduling, communications, and synchronization automatically. The main changes have been the subdivision of the domain and the exchange of information between tiles. Although these techniques incur extra processing, they have allowed taking advantage of the CPU and GPU resources. There is a trade-off between speed-up and the benefits of use OmpSs.

Programming with CUDA requires explicit operations like device selection, data transfers between host and devices, flow control through queues and events, among others. This kind of operations makes the programming difficult and prone to errors. On the other hand, OmpSs facilitates the use of all available GPUs in one node, makes flexible use of resources and exploits its full capacity, and scale the system to multiple CPUs or GPUs without modifying the program.

Chapter 4

Scaling Crowd Simulations in a GPU Accelerated Cluster

4.1 Introduction

Heterogeneous clusters provide various levels of parallelism that need to be exploited successfully to harness their computational power. At node-level, Pthreads, OpenMP directives, among other libraries, provide access to CPU level parallelism and OpenACC, CUDA or OpenCL libraries do the same for accelerators. Once node-level parallelism is achieved, additional scalability is accomplished by allocating computation in several nodes, using the message passing interface (MPI).

Programmers need to combine these programming models in their code and fully optimize it for each level of parallelism which turns into a complex task. These two processes also may suggest programming heterogeneous systems is an ad hoc activity.

During the last years, the Barcelona Supercomputer Center (BSC-CNS) has led a task-based approach to simplify code development efforts on these architectures. As a result, programmers can use OmpSs and MPI to leverage the computational power of heterogeneous clusters.

Our particular endeavors have been focused on the design, development, and analysis of crowd simulations in these systems. Our first efforts [17] combined CUDA and MPI models for in-situ crowd simulation and rendering on GPU clusters; also OmpSs was adopted [15]. The author proposed a task-based algorithm that allows the full use of all levels of parallelism available in a heterogeneous node. OmpSs allowed us to analyze the simulation under different resource allocations.

In this chapter, the author describes new techniques to scale on a heterogenous computing cluster the algorithm introduced in chapter 3. The analysis of the algorithm and provide scalability performance of the simulation running under different resource allocations on the Barcelona Supercomputing Center's Minotauro Cluster.

The rest of the chapter is organized as follows: Section 4.2 presents an overview of the algorithm used as a case study. Section 4.3 describes the implementation. Section 4.4 shows the results obtained. Finally, Section 4.5 presents the summary of this Chapter.

4.2 Algorithm Overview

Following our previous development and analysis efforts using different programming models and parallelism approaches, the author uses the wandering crowd behavior algorithm introduced in Chapter 3 where design and complexity details of the algorithm are discussed in detail. However, for clarity sake, this is the basic description of the algorithm.

The sequential algorithm is composed of three primary tasks as shown in the following steps:

Step 1: Navigation space is discretized using a grid.

Step 2: Set up the agents' information (position, direction, and speed) randomly according to the navigation space size.

Step 3: Update position for each agent: Calculate collision avoidance within a given radius. Evaluation of the agent's motion direction switches every 45 degrees covering a total of eight directions counterclockwise. The agent moves in the direction with more available cells.

A double level of tiling is applied. The navigation space and agents are divided into zones. Each zone is assigned to a node which in turn is divided into sub-zones (Figure 4.1). Then, the tasks performed in each sub-zone are executed in parallel. Stencil computations are performed on an inter and intra-node basis. Below the description of the algorithm.

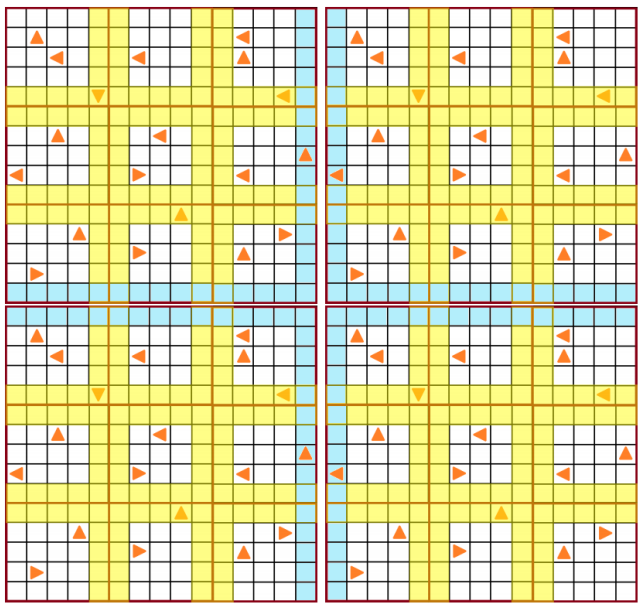


FIGURE 4.1: Dividing the domain in tiles

Buffers to exchange information between zones and sub-zones are defined (agents information and world cells status). Since it is applied a double level of tiling, the system manages local coordinates for zone and sub-zone inside each tile besides the global coordinates. For this reason, it is necessary to compute an offset for each of them to switch between global and local coordinates. It is configured the topology

internode and intra-node defining the neighbors for each zone and sub-zone. The number of zones is variable; the number of zones is matched with the number of nodes, tests with 4, 9 and 16 nodes were done. The system use 16 sub-zones because from the experiments in one node, better results with this configuration were obtained for a large number of agents. The neighbors of each zone are assigned regarding the total number of zones. Then it is initialized an equal number of agents in each sub-zone. The halos to exchange navigation space data with the neighbors (internode and intra-node) are filled *_world_halos* with the info about the occupied cells. Then, iteratively the system exchanges the world_halos with the neighbors, and update the agent's position. The system identifies if an agent goes to a different zone or subzone according to his new position. It fills the *agents_halos* (internode and intra-node) with the agents data. The system exchanges with the neighbors the agents_halos, fills the world_halos and repeat.

Algorithm 3: Crowd simulation algorithm to

```
1  Setup_exchanged_data_buffers();
2  Compute_zone_offset();
3  Compute_subzones_offset();
4  Configure_internode_topology();
5  Configure_intranode_topology();
6  for j ← 0 to subzones do
7  |   Agents_initialization();
8  |   Fill_intranode_world_halos();
9  end
10 Fill_internode_world_halos();
11 for i ← 0 to Iterations do
12 |   Exchange_internode_world_halos();
13 |   for j ← 0 to subzones do
14 |   |   Exchange_intranode_world_halos();
15 |   |   Update_agent_position();
16 |   |   Fill_intranode_agents_halos();
17 |   |   Fill_internode_agents_halos();
18 |   |   Exchange_intranode_agents_halos();
19 |   end
20 |   Exchange_internode_agents_halos();
21 |   for j ← 0 to subzones do
22 |   |   Fill_intranode_world_halos();
23 |   end
24 |   Fill_internode_world_halos();
25 end
```

4.3 Implementation

OmpSs' advantages for code development in heterogeneous architectures were noted in the previous chapter 3. As shown in the algorithm from Section 4.2, a new level of stencil computation and agents' information exchange have been introduced to extend computations to additional nodes. OmpSs flexibility allows us to make MPI

calls within tasks to implement internode stencils. Thus, incremental code development is still possible.

As an example, a code snippet of one function converted to a task is illustrated in Listing 4.1. In this case, **target** directive indicates that the GPU will execute the *updatePosition* task. OmpSs will take care of the memory management by using the directive **copy_deps**. There is an explicit definition of data dependency for variables *agents*, *ids* and *world*, which are input and output of the task. Task execution is done by calling *updatePosition* as a regular language C function.

As noted in Section 4.2, it is applied a second level of tiling, first dividing the domain(agents and the domain search) between the nodes, assigning one tile (called zone) to each node, and inside each node the zone is divided into subzones. Then each subzone is processed as a task, and they are distributed between the computing resources (CPUs and GPUs) for parallel processing. At run-time, this information is exchanged within the nodes by OmpSs and between the nodes by MPI (code lines 31-38 from listing 4.1).

LISTING 4.1: Converting *updatePosition* function into a task to be executed as a CUDA Kernel.

```
1
2  // Kernel declaration as a task
3  #pragma omp target device (cuda)
4  ndrange(2, (int) sqrt(count_agents_total), \\
5  (int)sqrt(count_agents_total), 16, 16) copy_deps
6  #pragma omp task inout(
7  ([agents_total_buffer] agents ) [0; count_agents_total],
8  ([agents_total_buffer] ids ) [0: count_agents_total],
9  [world_cells_block] world )
10 extern "C" global void updatePosition(
11 float4  *agents, *ids, int *subzone, ...);
12
13 // Kernel execution
14 bool runSimulation ( )
15 {
16     ...
17      for ( int i = 0 ; i < subzones; i ++)
18      { ...
19          updatePosition(agents[i], ids[i], subzone[i], ... );
20        ...
21      }
22
23 #pragma omp taskwait
24     ...
25
26      ////////////////////////////////////////
27      //Internode exchange of agents
28      ////////////////////////////////////////
29      MPI_Send(agents_left_out_pos, ...);
30      MPI_Send(agents_left_out_ids, ..);
31      MPI_Send(&count_agents_left_out, ...);
```

```
32
33      MPI_Recv(agents_int_pos, ...);
34      MPI_Recv(agents_int_ids, ...);
35      MPI_Recv(&count_agents_right_in, ...);
36      count_agents_int += count_agents_right_in;
37
38  }
```

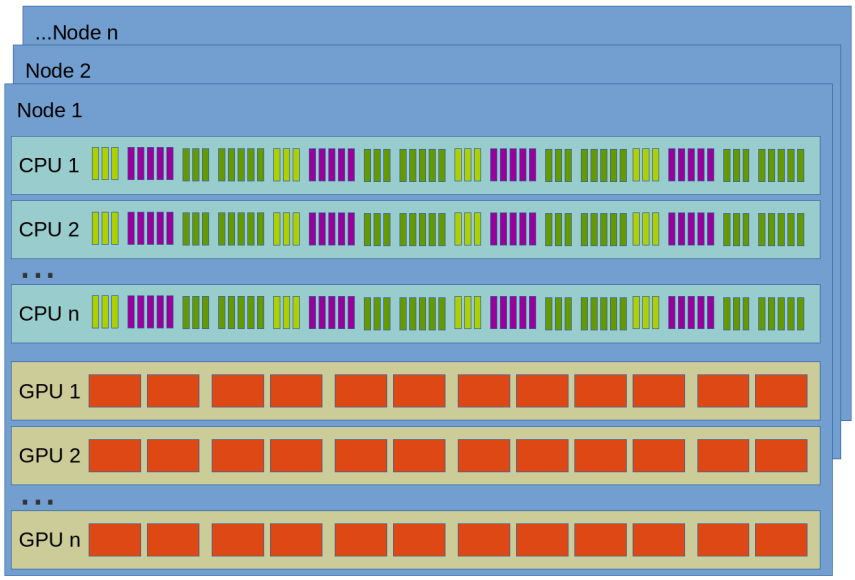


FIGURE 4.2: Tasks processed in parallel in different nodes

Using run-time variables, it is possible to control the number of CPUs and/or GPUs to be used. By default, OmpSs uses all the available computing resources. If more hardware resources are added to one node, for example, more CPUs or GPUs, the simulation will use them without any code changes. Figure 4.2 shows a schema of how tasks are processed in parallel inside each node also it shows the four levels of parallelism, the first one at the node level, the second one at a CPU core level, third one at GPU level and the fourth one at CUDA thread level.

4.4 Results

Tests were done on the Barcelona Supercomputing Center's Minotauro Cluster. Each node has two processors Intel Xeon E5-2630 with 8 cores each processor. With 128 of RAM memory. Two NVIDIA K80 GPU cards. Running Linux operating system. It was used CUDA 7.5, mercurium compiler version 2.1.0, and nanos runtime system version 0.14.1.

Execution time of one-node and multi-node versions were compared, for the multi-node version were used 4, 9 and 16 nodes. It was simulated up to 126 million of agents, in a world of 27852x27852 cells. For the one-node version, the world was partitioned into 16 tiles, i.e., a zone with 16 sub-zones, using two GPUs and 14 CPU cores. In the multi-node version the world was decomposed into 4, 9 and 16 zones, according to the number of nodes, then each zone was partitioned into 16 tiles (sub-zones) for all cases. In each node, both GPUs and 14 CPU cores were used, the remaining two cores control both GPUs. Figure 4.3 shows the results, comparing

both versions.

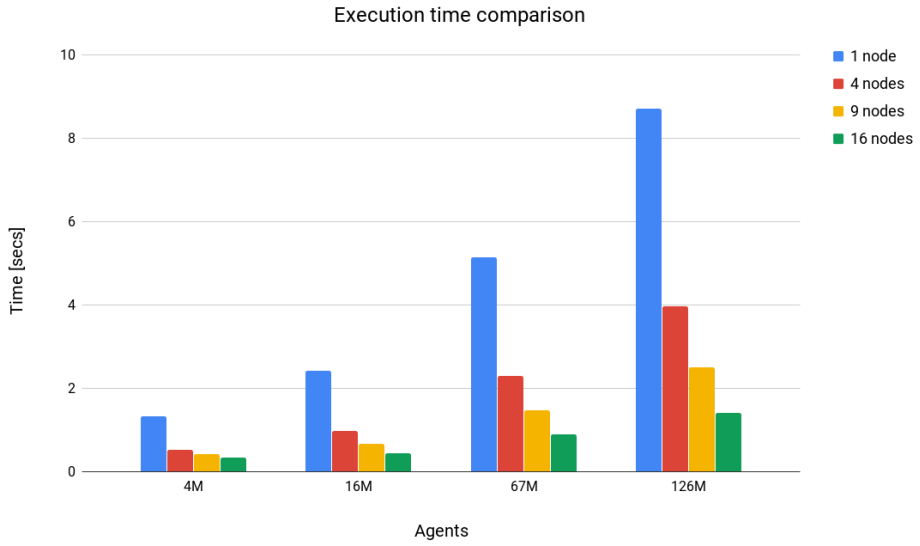


FIGURE 4.3: Execution time one-node vs multi-node

In general, as the number of agents increases, the advantage offered by the use of several nodes concerning the execution time is more evident.

Applying tiling in one node allowed us to simulate a large number of agents, limited just by the host memory size. The bottleneck is the data transfer between the host and the GPUs. Figure 4.4 shows the data transfers in one iteration for both GPUs, for a simulation of 64 million of agents. Although it was overlapped with other tasks performed in the CPU, at one point the data calculated by the GPU is indispensable to continue with the simulation.

For the multi-node version, bottlenecks related to CPU-GPU data transfers are solved by distributing the data between nodes. Nevertheless, another type of bottleneck appears as communication overhead between nodes. It was minimized such bottleneck by compacting data, i.e., when border information exchange occurs, only occupied cells are transferred, and by overlapping computation with communication.

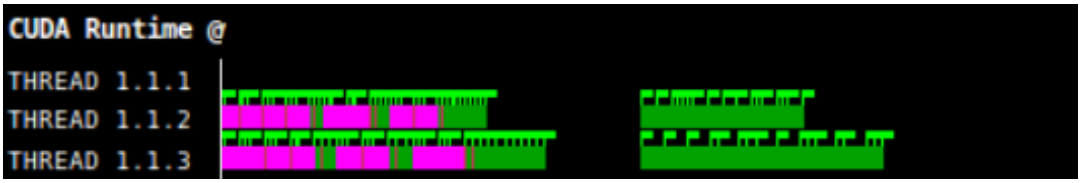


FIGURE 4.4: Data transfer between the host and the GPUs. The pink color represents data transfer from the host to the devices, the green represents transfers from the devices to the host.

Each node of the Minotauro Cluster has a PCI Express 3.0 interface. Summit, the

current cluster at Oak Ridge National Laboratory [84], provide NVLink technology that is 3x faster than PCI Express 3.0. Thus, these bottlenecks could be alleviated by hardware technology improvements. Nevertheless, according to the results the double tiling technique allows to take advantage of all the cluster's computing power.

The achieved speedup for 4, 16, 64 and 126 million agents, using 4, 9 and 16 nodes is shown in Figure 4.5. In general, the results are satisfactory you get a speed-up in all cases. With four nodes, a better value is obtained for the smaller amounts of agents (4M and 16M), the greater the number of nodes the greater the internode communication, therefore, it is necessary to identify the optimal number of nodes to be used based on the number of agents. With 9 nodes the best result is obtained with 16 million. Four million no longer compensate the use of 9 nodes, that is, communications exceed the computing capacity required to process 4 million agents. There is an increase in speed-up for 67 and 127 million agents. With 16 nodes the best result is obtained by simulating the largest number of agents 126 million.

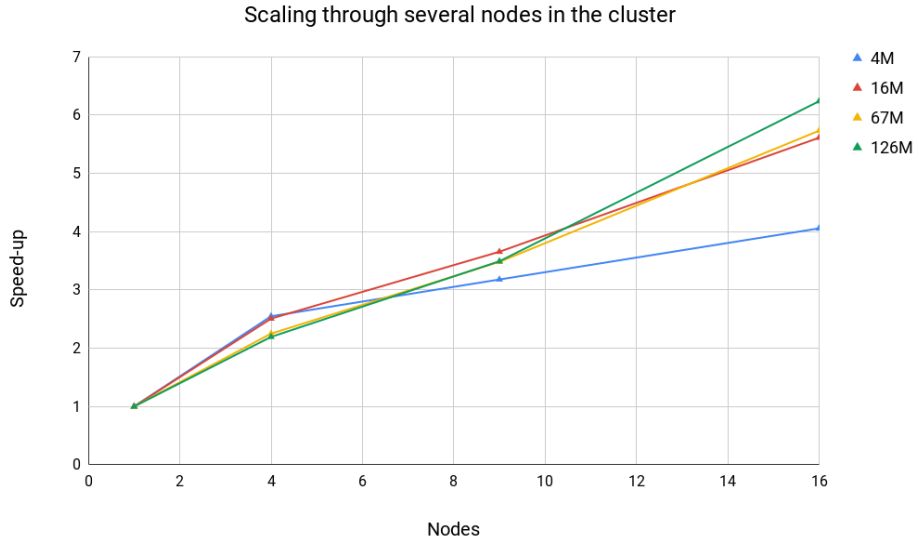


FIGURE 4.5: Speed-up in different number of nodes

4.5 Summary

It was presented the development of a task-based scalable crowd simulation and analyzed how the simulation scales to a different number of nodes. The system can scale up to several hundred million agents by using multiple nodes, using all the computing resources available in the cluster. It is possible to define the optimal number of nodes to be used based on the size of the simulation.

Comparing with our previous work [17], which used MPI and CUDA programming models, the results were improved. The current version allows to take advantage of the computing resources effectively (CPUs and GPUs) by using OmpSs. Besides,

it was identified some bottlenecks and solved them: applying a double tiling technique that allows distributing the work between the nodes, minimizing the transfer of data between nodes and exploiting up to four levels of parallelism.

Chapter 5

GPU Cluster Crowd Visualization

5.1 Introduction

Crowd simulation and visualization at the megacity scale requires significant computational power and memory resources commonly available on High-Performance Computing platforms (HPC). Current HPC clusters have hundreds of thousands and even millions of processors. The community is in a transitioning stage between the petascale and the exascale era.

According to Top500 Supercomputer Sites list [5], the use of heterogeneous architectures combining multi-core processors and accelerators is becoming a common practice; accelerators such as the Intel Xeon Phi and the NVIDIA GPU are commonly used. For our purposes, the GPU has two advantages: accelerates compute, and rendering for in-situ visualization.

The vast amount of data produced by this kind of systems requires further processing to offer insights about the results. Such information usually is simplified before analysis; however, simplifications may hide relevant details. On the other hand, visualization, which transforms these findings into graphical and color representations, can enhance human cognition improving analysis.

In-situ visualization consists of visualizing results on the same cluster where simulation is running [10]. This conveys two advantages: relevant results are stored before simulation finishes, and data movement from processors through different memory hierarchies and storage is reduced or avoided.

In addition, in-situ visualization reduces I/O operations [85], [86] by avoiding full data transfer of results. It also reduces time, allowing the researcher to inspect partial simulation results and perform simulation adaptations as required.

The rest of the chapter is organized as follows: Section 5.2 presents the different configurations to visualize detailed 3D crowds. Section 5.3 describes the development of urban scenarios and its integration with the crowd simulation. Section 5.4 presents the results. Finally, Section 5.5 presents a summary of this chapter.

5.2 Crowd Visualization Engine Modules

The selection of a visualization mechanism depends on the simulation and available HPC infrastructure; our approach for visualization can render detailed 3D crowds.

The crowd is rendered using animated characters with different geometrical and visual appearance.

Figure 5.1 shows the crowd visualization engine's modules designed to visualize detailed 3D crowds. The generation of diversity GoD stage runs as a preprocess. This stage generates and animates 3D characters semi-automatically. In run-time simulation results are stored in an array of ids and agent positions which are used to render unique animated characters. Discrete level of detail (LoD) and view frustum culling (VFC) are implemented in GPU to visualize several hundreds of characters in real time. As a result, the visualization engine coupled with the simulation engine generates frames that are processed or transmitted according to different configurations.

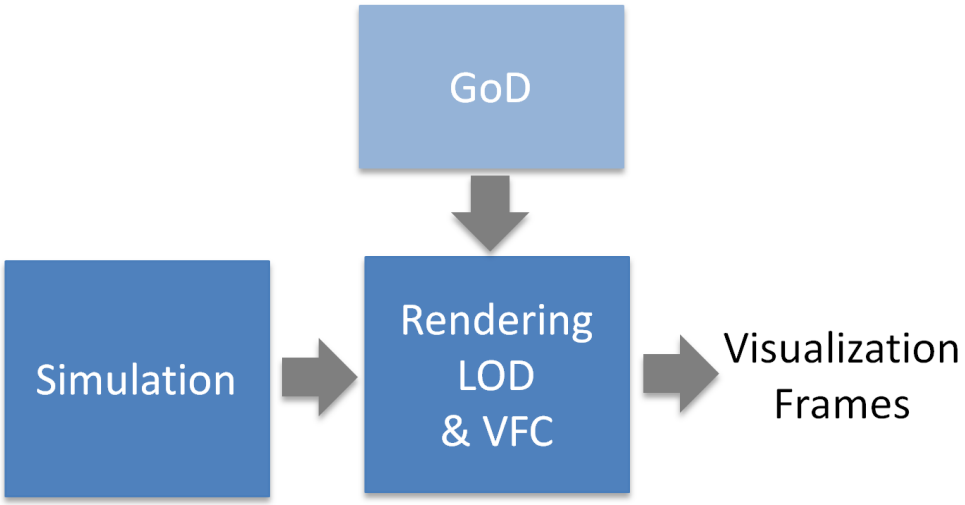


FIGURE 5.1: Crowd engine modules.

Three configurations were developed; they are described in the following paragraphs that are suitable for the particular BSC's infrastructure.

5.2.1 In Situ

The crowd engine was modified to support MPI communications and off-screen rendering (Figure 5.2). MPI communications enable the engine to assign simulation/rendering work to each participating node, share simulation results between nodes, and transmit partial renderings for the final composition.

Off-screen rendering enables each node to capture a visualization frame with color and depth information encoded in an RGBA array where the depth component is stored in the alpha channel. Then this information is downloaded to RAM and transferred to the master node.

Once the master node receives the color plus depth images from each node, it generates the final composite. This image is generated based on sort last depth compositing algorithm [87] implemented in OpenGL Shading Language GLSL. Finally, the composite image is sent to the client through VirtualGL.

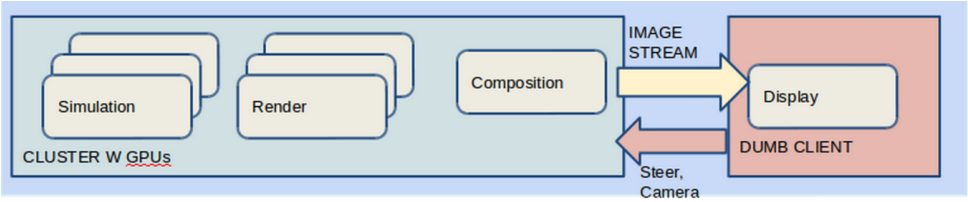


FIGURE 5.2: In-Situ, simulation and visualization occur in the Cluster.

The In-Situ configuration is suitable for massive simulations. All calculations are done in the cluster. The client receives an image with the results. Therefore the client does not require additional rendering resources and it does not restrict the number of characters in the simulation; however, the end user needs to be connected in the same local area network LAN to Minotauro cluster.

5.2.2 Streaming

Accessing simulation results for visualization in any computer is one of the system's goals. It is desirable to enable researchers to access their results for visualization and analysis remotely, and such access may occur during or after simulation. A second goal is to couple the visualization engine to different simulation engines. A case study with Pandora framework [78] was developed.

However, interactive crowd visualization can be affected by bandwidth, latency and image transport technology. BSC's Minotauro cluster has a fast network connection to BSC's LAN and default visualization delivery system uses VirtualGL. This allows rendering crowds at interactive rates. Nevertheless, external connections to Minotauro are restricted resulting in a visualization frame rate drop to 1-2 fps (frames per second).

To achieve these goals, the streaming configuration transmits simulation data instead of rendered frames from a remote resource to the end user's workstation or visualization server (Figure 5.3). The current streaming algorithm does not implement any buffering technique. However, the crowd engine interpolates the received agents' positions between steps to avoid artifacts due to possible data loss.

This configuration is suitable when the bandwidth is lower than the required one for transferring images. However, the size of the simulation is limited by the number of agents the workstation can render.

5.2.3 Web

A particular requirement of the Streaming configuration is the use of a high-end system for visualization. On the other hand, In-situ configuration requires an internal connection between the user and Minotauro cluster. It was designed the Web configuration to fill the gap between streaming and in-situ. This option allows researchers to access visualization results on any device and away from the desktop.

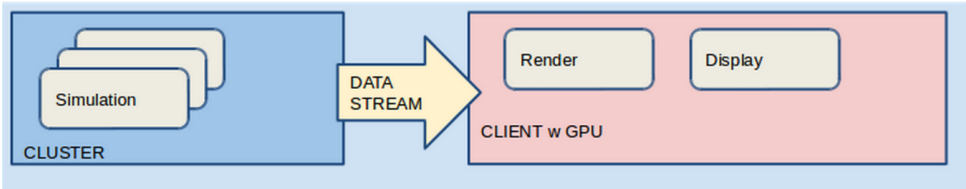


FIGURE 5.3: Streaming mode allows to couple different crowd simulation engines, connect to them remotely and visualize the results in a workstation.

Inspired by cloud gaming technology, this configuration captures the results from visualization and streams them out to a web browser (Figure 5.4). It follows the client/server architecture: the server executes the simulation, visualization, and streaming, and the client displays the visualization frames and captures user interaction events that are sent back to the server.

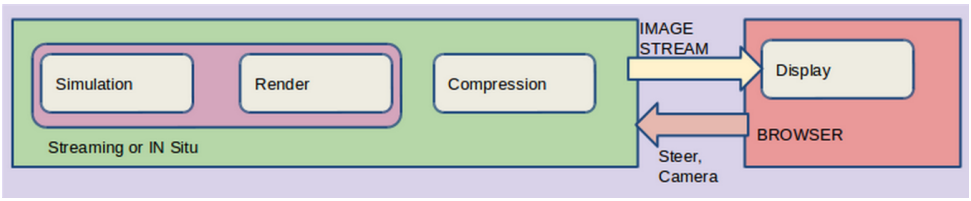


FIGURE 5.4: The Web system architecture allows to display visualization results in web browsers.

5.3 Development and Integration with Urban Scenarios

This kind of systems has an essential use in emergencies. Therefore, urban environments were created as realistic as possible; in this way, users will be ready to face real events. Besides, the use of a virtual interface is being evaluated. It could allow the user to have total immersion in the simulation and interact using devices such as Oculus Rift and Leapmotion.

To simulate realistic urban scenarios the project gathers data from sources such as OpenStreetMap OSM and Natural Hearth from them it is possible to generate 3D geometry of buildings and get data of streets, parks, points of interest, etc. To complement the data of the environment, Open Data were used, these data are published by the public administration of each city, describing aspects such as the city's infrastructure, pollution, population, among others, however, they are not standardized, therefore, each city publishes its data with its format and structure.

The combination of different data sources involves a process of cleaning and extracting data before they can be useful. For this task, tools like Hadoop, Pig, Spark, among others were used.

Lately, the main focus has been the web configuration since it is the most flexible and

accessible. WebSockets are used for communication between the server and the web client. Figure 5.5 shows the system architecture to integrate urban environments.

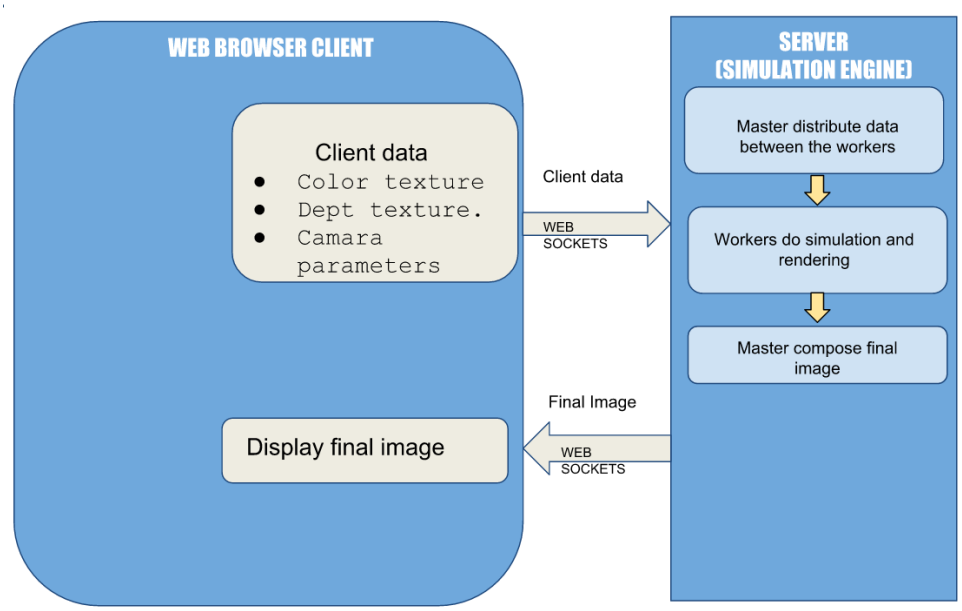


FIGURE 5.5: System architecture to integrate urban environments.

Client pipeline.

The main objective in this stage is to synchronize the camera with the information that is displayed and merge it with the simulation, whenever the user applies any transformation to the camera, (i.e. rotation or translation) the depth buffer (**DBF**) of the scene is extracted, also the scale in pixels per meter of the current zoom level, the $2D$ position relative to the current map, and the RGB of that specific frame. This is used to reconstruct the scene later and composite it with the crowd. The client life cycle can be described as:

1. When the user starts the connection with the client, the first communication will inform the server the current map along with the initial position in latitude and longitude of the center of the camera.
2. The client render will be captured along with its correspondent **DBF**, virtual camera parameters, position, scale, and other necessary data will be encapsulated and sent in a buffer to the simulation server.
3. The client will be waiting and listening for composited rendered frames from the server.
4. Once a frame has arrive at the client, the result will be shown in a canvas inside the *html* client.
5. From the step 2 to 4, the cycle will be repeated infinitely until the client is closed along with the connection.

5.3.1 Server pipeline.

The server side is the one that performs all the data processing; its cycle can be depicted as:

1. The master process starts with the initialization of the simulation and render parameters.
2. The slave's processes are initialized.
3. The master process will spawn a server process with the WebSockets protocol listening for a client.
4. When the client performs a connection with the server, the initial simulation parameters are processed (world map position the most important).
5. The slave process is taken from the idle state to a simulation state.
6. The relative camera position is loaded from the client frame buffer.
7. The camera position is distributed from the master process to the slave process cameras.
8. The simulation is rendered from every slave process camera's point of view to a texture.
9. The rendered scenes textures are sent to the master process, ant the image composition is performed by the master process.
10. The output render is sent to the client by the server process.
11. Steps 6 to will be in an infinite cycle until the client closes the current connection or a prudent time without a response from the client will be expired.

The server uses a map in 2D, consisting of a top-down orthogonal view of the scene to identify obstacles. Figure 5.6 shows the composition image of four partial rendering processes of the crowd characters and the background.

To develop and integrate the urban scene different platforms like CesiumJS, Tangram, Unity3D were tested. Lately, the focus is on Unity3D because it allows development and integration with virtual reality devices like oculus rift, htc vive, and leap motion.

5.4 Results

In-situ configuration was tested in the Minotauro cluster using five nodes. The simulation was decomposed into four tiles, then four nodes simulated and rendered 4096 characters and the remainder performed image composition (Figure 5.7). Figure 5.8 shows a close-up view of the simulation. In this case, frame rates between 15 to 20 fps for a simulation size of 16K characters were obtained.

Two experiments were designed to test the Streaming configuration performance. The first test consisted of streaming data of a wandering behavior previously simulated in Pandora Framework from a remote simulation server to a workstation. The simulation size was 4096 agents, and the workstation contains an Nvidia TITAN GPU. For this case, a consistent frame rate of 60 fps was achieved.

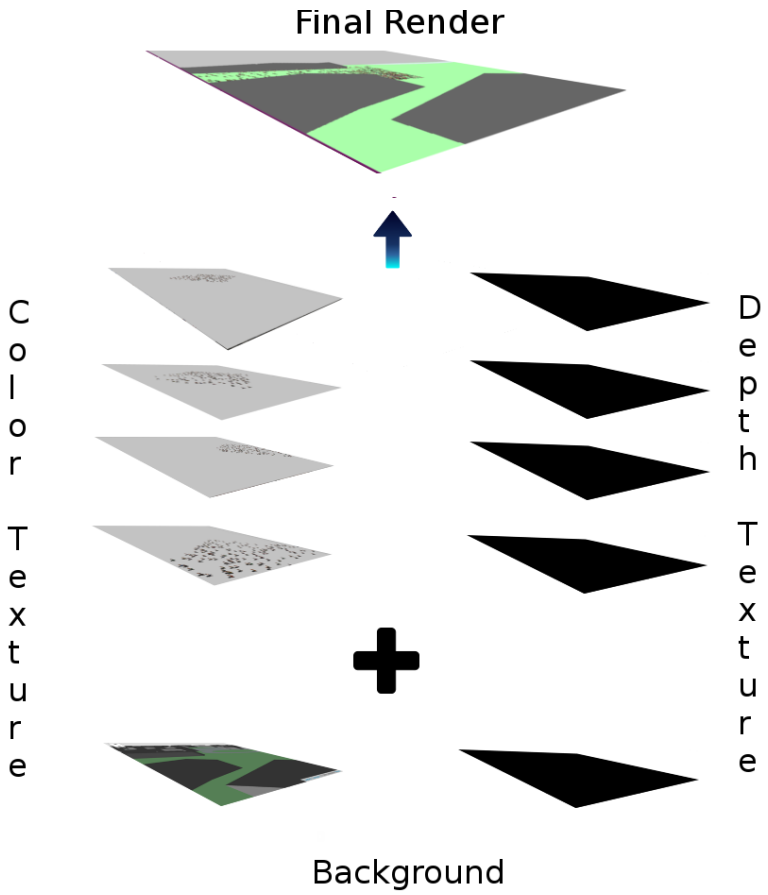


FIGURE 5.6: Composition of four partial rendering processess of the crowd characters and the background.

The second test consisted of executing a simulation in Minotauro cluster and streamed out results every step to the workstation. The simulation size was 4096 agents as in the first case. The system was also able to render 4096 characters at 60 fps.

Two experiments were designed to verify the functionality of the Web configuration. The first one consisted of running the system on the workstation using different web browsers to detect compatibility issues (Figure 5.9). The results showed that Chrome, Firefox, and Opera browsers were able to display the visualization successfully. A simple inspection of the interaction did not reveal any lag between user commands and simulation response.

In the second experiment, two clients were connected to the workstation. The first client was a Windows 7 system, and the second client was an Android-based mobile device. The same visualization was displayed successfully on both devices; however, the lag between user commands and simulation response was noticeable.

Figure 5.10 shows an example of the integration of the crowd simulation with an urban environment.

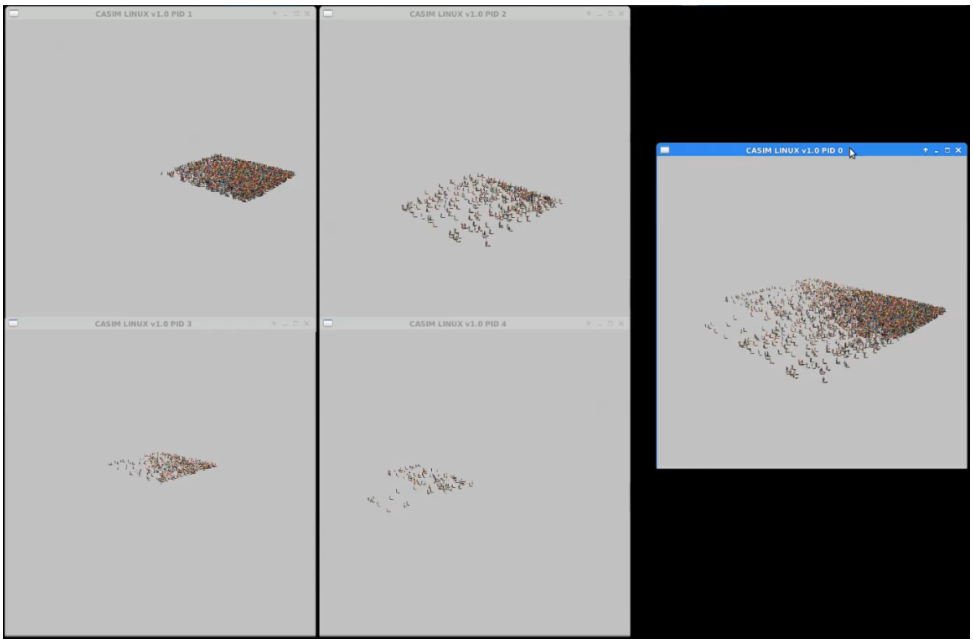


FIGURE 5.7: Far distance view.

5.5 Summary

A configurable crowd engine that supports three configurations was developed: streaming, in-situ, and web. In-situ configuration allows us to take full advantage of the cluster computing resources for large-scale simulation and visualization. The Streaming configuration supports crowd visualization in machines connected outside the BSC's LAN; while the Web configuration allows us to display simulations on any device.

Streaming and Web configuration offer flexibility and may reduce infrastructure costs when advanced visualization systems such as the CAVE or Tiled Display Walls are remotely available. The streaming configuration also avoids the difficulty of coupling already existing simulation tools, and the Web configuration can be used to take advantage of sensors available on mobile devices to improve user interaction. Finally, In-situ configuration offers a testbed to analyze further scalability behavior of interactive simulations that use the GPU for computing and rendering.

Access to different data sources allows the development of realistic urban environments, which seems essential in emergency training systems, as it will enable users to be better prepared to face real events. In this chapter, it was described how to integrate the urban environments with the crowd simulation.

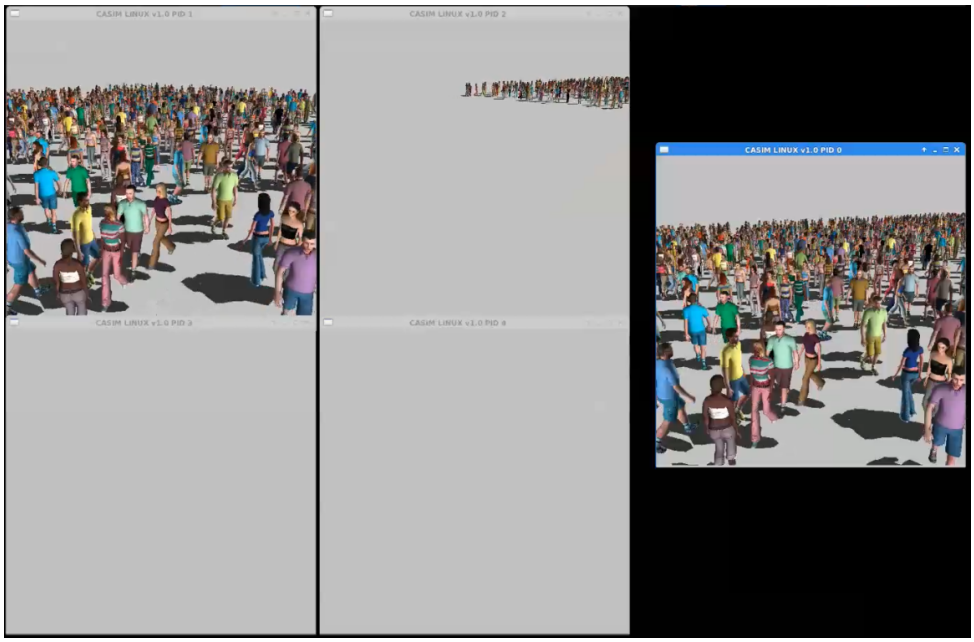


FIGURE 5.8: Close-up view.

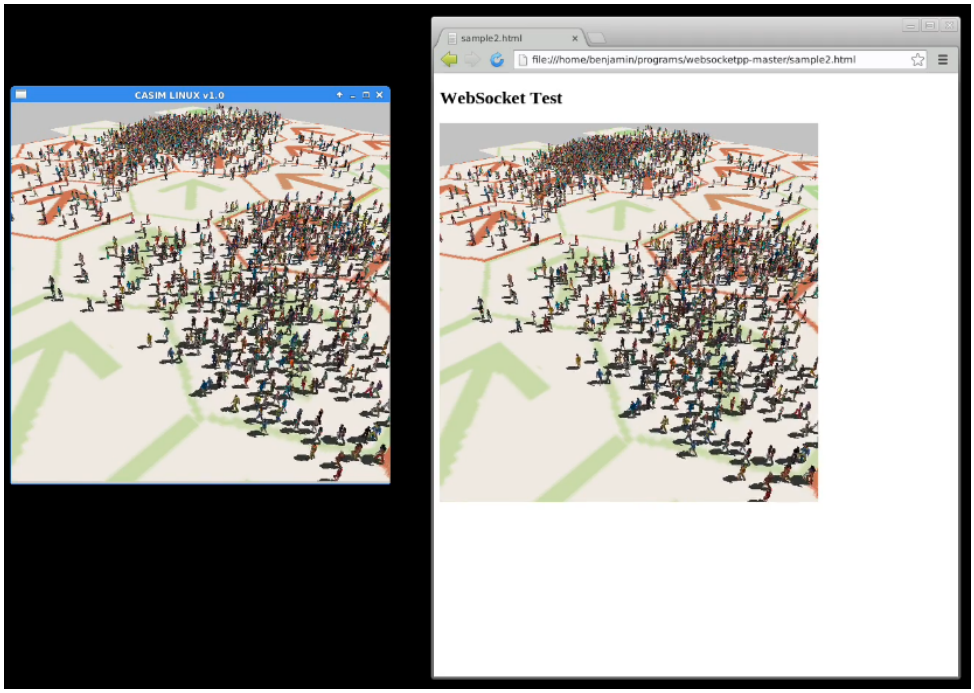


FIGURE 5.9: Web configuration results. Left: OpenGL/glut window. Right: Visualization is displayed in Chrome browser.

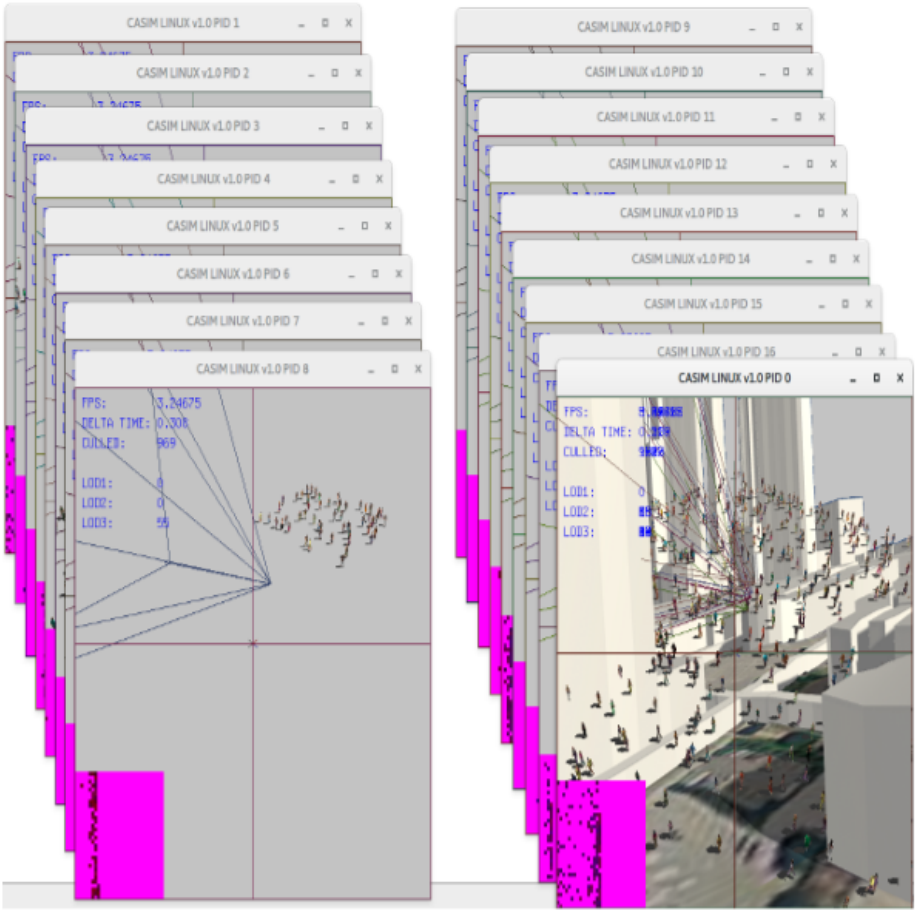


FIGURE 5.10: Integration of the crowd simulation with a urban environment.

Chapter 6

Hierarchical Path-Finding

6.1 Introduction

Path planning for large-scale crowds, autonomous driving in real scenarios or video games [88] is challenging. Due to the dynamism in the scenes and vast search space, it is often necessary to reprocess the path or make an additional process to return to the original one after avoiding a collision, making the process computationally expensive. In this sense, the use of a hierarchical approach offers different advantages: it divides the search space reducing the problem complexity. It can obtain a partial path instead of wait for the complete one, which allows a character to start moving and compute the rest asynchronously. It can reprocess only a part if necessary with different levels of abstraction.

Typically the approach employed is to construct a graph from an occupancy matrix and then to traverse the graph using graph-algorithms such as A* [89], Dijkstra [90] or Breadth-First Search [91]. All of the incumbent methods are very mature. However, they consume relatively high amounts of memory, are slow due to the number of pointer manipulations and are not readily amenable to hardware acceleration for real-time operation. Even enhancements such as visibility graphs do nothing to change the fundamental nature of the graph algorithms and require significant levels of pre-processing which are not feasible to implement in real-time.

In this chapter, a new approach is proposed which operates directly on the grid or bitmap rather than first converting it to a graph making the implementation very hardware friendly. It is domain-independent, and it could be adapted to dynamic scenes. The method makes use of a hierarchical representation which incrementally builds the equivalent of a visibility graph without pre-processing. The hierarchical approach reduces the search space, identifying the areas of interest in high levels of abstraction, allows the tree to be pruned as it is constructed using a heuristic based on density and distance. In this way, the algorithm gives priority to the shortest paths in areas with fewer obstacles.

An analysis of the algorithm is presented in terms of:

- Pre-computation time
- Use of memory
- Speed-up
- and Optimality

The contributions of this chapter are:

- A detailed description of a new hierarchical path-finding algorithm.
- A comparison with the A* as a benchmark for optimality and runtime.
- Results of tests done in maps from video games and urban environments.

The chapter is composed of the following sections: an overview of the related work is presented in section 6.2. Section 6.3 describes the pre-processing stage. Section 6.4 describes in detail the hierarchical path-finding algorithm. The experiments and results are presented in section 6.5. Finally, the summary concludes this chapter in section 6.6.

6.2 Related Work

There is a huge work about path-finding algorithms, A* [89] is one of the most well known algorithms. It is an informed search algorithm. It uses the next evaluation function to determine the order it explores the nodes:

$$f(n) = g(n) + h(n);$$

Where g(n) is the actual cost of an optimal path from start to node n, and h(n) is a heuristic estimation of the cost to get from node n to the goal. The heuristic used to be based on distance, there are different ways to calculate the distance, it could be Manhattan, Euclidean, Octile, etc. In this way, the algorithm gives priority to the nodes that bring it closer to the goal.

Two lists are used to control the processing of the nodes. The closed list contains the nodes already expanded, expanding means to discover its immediate neighboring nodes and evaluate them. And the open list which includes the nodes to be expanded. The algorithm chooses the next node to expand based on the minimal cost of the evaluation function.

From A* many other algorithms are derived like Iterative deepening A* IDA* [92]. IDA* is a depth-limited searching technique, where the limit increases iteratively in a procedural way until the target point is found. Every iteration is a depth-first search that uses the heuristic function to choose the nodes that seem most promising to reach the goal. If any node exceeds the depth-limit for that iteration, the search stops to go more in-depth to the current path and start with another immediate neighbor node of the current node. Otherwise, a new search begins with a higher cost threshold in the next iteration. It doesn't keep track of visited nodes which can cause nodes to be evaluated repeatedly, that could make the process slower.

During the research the author found The Grid-Based Path Planning Competition [93], below we explain some approaches that were part of the competition. This competition provided us with maps and experiments that make up the comparison framework for our algorithm.

Jump Point Search JPS [94] is an optimization of A*. It orders paths such that all diagonal moves are taken first, creating a canonical ordering over all routes. Since obstacles may block the canonical path, the algorithm introduces jump points. A Jump point is a location in the map which allows moving around the obstacle. Therefore the canonical ordering is partially reset. JPS reduces the number of expanded

nodes, accelerating the search.

JPS+ [95] is an optimization of JPS algorithm. To speed up the path searching it introduces an offline pre-process stage to identify jump points apriori. JPS+ pre-calculates the first straight and diagonal jump points in eight possible directions for every traversable node on the grid map. It enhances the performance of the JPS algorithm but also increases memory usages significantly. According to the results reported in [93], it needs up to 3.0Gb of storage and 74 minutes for the pre-processing stage. This is the time and space required for all the maps used in the competition.

N-Subgoal Graphs [96] place subgoals at the convex corners of obstacles and connect them. It is similar to visibility graphs (a visibility graph is a graph of intervisible locations, it means there is a direct connection between their nodes without obstacles) but has fewer edges. The subgoals are used to find the shortest path faster. A hierarchical structure of N-levels of subgoals are created by partitioning the vertices of a simple sub-goal, in this way sub-goals that are not related to a search are excluded, reducing the search space, making the algorithm faster and more efficient. It performs an A* search during the path-planning phase. According to the results reported in [93], it needs up to 293Mb of storage and 2.6 minutes for the pre-processing stage.

Approaches like JPS, Visibility Graphs and alike are not efficient when the search space contains a dense collection of small size obstacles. It would result in a large graph in terms of the number of nodes and edges. Therefore it can not exploit the idea of extended travel distances in a single step reducing the number of expanded nodes.

Single Row Compression SRC [97], it is a pre-processing-based algorithm it uses a Compressed Path Database (CPD), which store the path for any two points in the searching space. The strategy is to query the CPD for partial paths, and again until it gets the goal. This approach balances the computational load over time reducing the lag spike incurred when many agents are moving at the same time. This algorithm is one of the fastest, however, according to [93] the pre-processing stage could need up to 52Gb of storage and 12330 minutes, turning it in one of the most expensive in computing resources and time.

For algorithms alike A*, the computation cost is proportional to the size of the search space. Hence, path-finding on large maps may result in severe performance bottlenecks. When the search space is vast, it is convenient to use a hierarchical method to reduce the complexity dividing the search space. Below are described some of them.

Hierarchical Path-Finding A* - HPA* [98], which is one of the first detailed study of hierarchical pathfinding. It divides the search space into squared clusters, all clusters have the same number of cells, each one with an entrance. A graph connects each cluster on an abstract level. The optimal distances for crossing each cluster are pre-computed and cached. The hierarchy can be extended to N-levels. It searches the upper level, returning an abstract path and go down until it gets the desired refinement of the path. A* is used to look for a path in the abstract level and inside a cluster.

The concept is similar to ours. Both algorithms create a hierarchical structured based on fix size clusters. However, our algorithm does not define an entrance for

each abstract node and create a graph in the abstract level, for this reason, it does not guarantee to find a path but simplify the abstraction process and save space in memory.

Partial–Refinement A* - PRA* [99]. PRA* abstracts the search space looking for cliques and orphans iteratively, with a maximum number of 4-clique. These cliques are reduced to a single node in the parent abstraction. Therefore, it is guaranteed that all nodes in abstraction can reach any other state within a single step, except orphaned nodes. Thus, two nodes are connected, if they ever merge into the same parent. It can start planning at any level of the hierarchy. It uses A* to guide search at any level of abstraction. It minimizes the cost of search by expanding only nodes whose parents are part of the abstract path. It fine-tunes a partial segment at each step, instead of refining the entire abstract path from beginning to end.

SHPA* [100] is an optimization of HPA*. It decomposes the map into many variable-sized fully connected clusters based upon a greedy heuristic, rather than decomposing the map into many same-sized clusters as HPA* does.

The heuristic attempts to merge clusters based on two different properties. It merges clusters that have a large number of abstract nodes relative to their size and avoids merging clusters that exceed a maximum combined size. By building fully connected clusters with similar properties, it achieves better spatial decomposition and improves the search algorithm.

It uses a Euclidean distance heuristic to compute edge weights, and it searches for low-level paths using A*.

DHPA* [100] is an optimization of HPA* designed to be executed in dynamic scenes. It creates clusters in the same way as HPA*, but it run Dijkstra once for each abstract node within a cluster, to compute the optimal path from the low-level node to the abstract node and a path index pointing to a neighbor node within the same cluster, creating a separate cache for each abstract node.

The abstract search utilizes the path length, and the low-level search uses the path index to refine the path. Therefore, unlike HPA* it is not necessary to insert the start and goal nodes into the abstract graph before each search. Additionally, the use of A* online is much more expensive than query the cache info.

When the terrain changes within a cluster, the build algorithm recompute the intra-edges and the inter-edges connected to that cluster. DHPA* is faster than HPA*, but consumes significantly more memory and produces slightly less optimal paths.

Hierarchical Path-Finding for Navigation Meshes - HNA* [81] creates a hierarchical tree from the original map by recursively partitioning a lower level graph with a specific number of nodes, which is defined as a parameter. The objective is to find a balance with edges, looking for a small number of edges to reduce the path-finding cost. The partition is performed until the graph of the highest level cannot be further subdivided. It uses the multilevel k-way partitioning algorithm (MLkP) [101]. It uses A* to perform searches in the hierarchical representation. The algorithm is conceptually similar to HPA* but adapted for a hierarchical navigation mesh.

Quadtrees [102] have been proposed as a way of doing hierarchical map decomposition. This method partitions a map into square blocks with different sizes so that a block contains either only walkable cells or just blocked cells. The problem map is initially partitioned into four blocks. If a block includes both obstacle cells

and walkable cells, then it is further decomposed into four smaller blocks, and so on. An action in this abstracted framework is to travel between the centers of two adjacent blocks. Since the agent always goes to the middle of a box, this method produces sub-optimal solutions [103].

There are many different approaches; each of them provides trade-offs between memory, pre-processing time, and flexibility. The choice depends on the needs and priorities of the problem to solve.

6.3 Pre-processing

In this section, the different tasks performed in the pre-processing stage are explained. For the description, the map brc997d is used. It was taken from the Path-finding Benchmark [104].

6.3.1 Auxiliary Maps

To reduce the search space, a hierarchical structure is created based on the VOLA format [105]. The original map is scaled creating auxiliary maps with different levels of abstraction. The resolution of the map is reduced while maintaining connectivity and density information in the abstraction. Every level is four times smaller than the previous one, e.g., if the size of the original map is 256x256, next level is 64x64, next one 16x16, the map with the smallest resolution is 4x4. Figure 6.1 shows the abstraction for the different levels. The original map is considered the highest level in a hierarchical tree. Dots represent available cells, and a different character represents occupied cells.

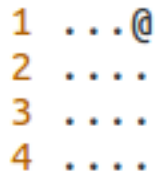
Each cell of a certain level (except the last one) represents a block of 16 cells (4x4) in the next level. The criterion to determine if a block should be considered occupied or free is: if at least one of the 16 cells that comprise it is free, then the whole block is deemed to be free. Otherwise, it would not be possible to define a path that would reach that free cell.

The maps are stored in blocks of 4x4, for level one there is only one block, for level 2 there are 16 blocks, for level 3 there are 256 blocks, for level 4 there are 4096 blocks. This distribution is an essential part of the algorithm because the main idea is to go from one block to the next one in each level, using as guide the path of the previous level.

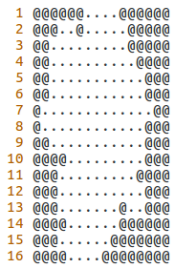
6.3.2 Density

Consider one block available because one of its cells is, this allows us to access these cells when the paths are computed. However, the characteristics of the original map are lost. In level one, it is practically unrecognizable since almost all cells are available.

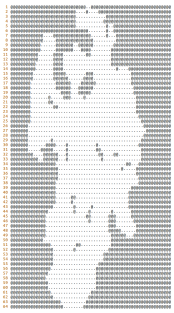
Density is calculated based on the number of available cells in the original map, e.g., In level one we have sixteen cells each one represents 4096 cells of the original map. Therefore its density would be between 0 and 4096. The value is normalized by dividing by the maximum number, in this way the same scale is used in all levels.



(A) Map for level 1



(B) Map for level 2



(C) Map for level 3

(D) Original map - level 4

FIGURE 6.1: Map abstraction

Table 6.1 shows the relation between the cells and the maximum number of available cells for each level.

TABLE 6.1: Density relation between levels

Level	Cells	Maximum number of available cells
1	16	4096
2	256	256
3	4096	16

Thus, the resolution of the original map is conserved regarding available cells. Table 6.2 shows the real values in level 1 for the map in figure 6.1. It is not necessary to calculate density for the last level since every cell only represents itself.

TABLE 6.2: Example of normalize values of density in level one (block 4x4)

0.16	0.29	0.42	0.00
0.40	0.70	0.84	0.13
0.02	0.79	0.76	0.04
0.01	0.81	0.17	0.01

6.3.3 Connectivity

It indicates if two neighboring cells are available. In this case, the neighbors could be at left, up, right or down, for the time being, the path is calculated only in four directions. We use one bit for each direction to store the connectivity of a cell. It is not necessary to compute the connectivity for the original map since we can check

the cell status directly; however, for the rest of the levels, we need to calculate this information.

The connectivity is processed top down. It starts on the last level (the level of the original map). For every block of 4x4 is checked if it has connectivity with its neighbors. By example to verify connectivity between two blocks, their borders are reviewed, if any cell in the edge has connectivity with its neighbor then there is connectivity between the blocks, e.g., if cell 3 of block 1 and cell 0 of block 2 are available, there is connectivity between these two blocks. Table 6.3 shows the neighborhoods of our example with their borders in gray.

TABLE 6.3: Diagram to show connectivity

Block1				**Block2**			
0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15

To store the connectivity, 64 bits are used for each block, four bits to store the connectivity of each cell. Bit 1 to check connectivity at left, bit 2 for up, bit 3 for right and bit 4 for down. Image 6.2 shows the association between bits and directions.

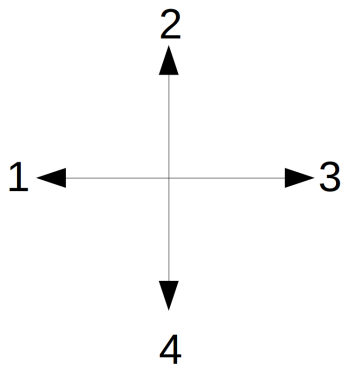


FIGURE 6.2: Use of the bits to store connectivity.

This is a recursive task done top down, from the original map it is possible to know if there is connectivity between the cells in the next level. Table 6.4 helps to show an example of how connectivity is used. Free cells are indicated with * and occupied cells with @. According to the criteria. For this case, there is connectivity between these two blocks, since there are free cells in the borders.

These connectivity values allow us to find a path from one cell in the right border of block one to any cell in block two. However, if the start is in one cell of the left edge of block one there would not be a path, the algorithm exhausts all options trying to find a path.

If the algorithm does not find a path, it disables the connectivity between these two blocks temporarily. Then it would look for a different option one level up. Once the solution is found, the original values of connectivity are restored. It is a permissive

policy. However, it is necessary since there is no knowledge about which will be next start and goal coordinates.

TABLE 6.4: Example of connectivity between two blocks

Block1				Block2			
*	*	@	*	*	*	*	*
*	@	*	*	*	*	*	*
*	@	@	@	*	*	*	*
*	*	@	*	*	*	*	*

The algorithm does not use lists to control which nodes have been already evaluated. Instead, it uses connectivity, which is the way to control which paths are not available for a specific case while the tree is pruned. Connectivity guarantees the same path is not evaluated two times.

6.3.4 Temporal task: Pre-compute Paths

The algorithm applies a Breadth-First Search BFS to look for the best path in a 4x4 blocks. BFS is used because it can operate directly on the grid or bitmap rather than first converting it to a graph making the implementation very hardware friendly. In the next phase of the project, it is planned to implement this function in hardware for applications in robots or drones and calculate the paths in parallel and asynchronous way for applications in crowd simulation or video games. In the meantime, to simulate this result the paths are pre-computed for all levels.

Once the auxiliary maps, density, and connectivity are processed, the paths for all levels are computed, the best ones are stored according to the heuristic. In this way when a search is done, the algorithm queries a Lookup table LUT, instead of calculating it, this gives us faster results in exchange for an initial pre-processing time. Details about pre-processing and execution time can see in section 6.5.

6.4 Hierarchical Path-finding Algorithm

For the time being our algorithm moves in four directions; therefore, Manhattan distance is used for the heuristic. To compute the path density and distance between start and goal are used as heuristic according to this formula:

$$heuristic = density/distance^2$$

By increasing the distance to the square, it gives priority to finding the shortest paths, but still, areas with fewer obstacles are considered. The paths with the highest values for the heuristic are chosen.

A recursive function to process all possible options is used, see listing 6.1.

LISTING 6.1: Path-Finding Algorithm

```
1  //Pre-process stage
2  Load_map();
3  Load_experiment();
4  Identify_abstraction_level();
5  Create_auxiliary_maps();
```

```
6   Setup_connectivity();
7   //Start experiment
8   Compute_coordinates_all_levels();
9   Find_global_path(){
10      For(i = 0; i < level; i ++){
11              Find_path_by_level(i){
12                  ...
13                      if(path not found){
14                          Update_connectivity();
15                          Find_global_path(level-1)
16                      }
17              }
18      }
19  }
```

An example is used to explain how the algorithm works. The start(100,125), and goal(89,247) coordinates are defined. The algorithm calculates the start and goal for all levels converting the coordinates to one dimension, table 6.5 shows the values.

TABLE 6.5: Start and Goal in each level

Level	Start	Goal
1	5	13
2	118	245
3	2009	3926
4	32100	63321

It finds the path in level one which is a block of 4x4 cells. According to our heuristic, the best solution is 5 9 13. Figure 6.6 shows the path on the grid for level one. Level one is used as a guide to finding the solution in level two. Level two is a grid of 16x16 divided into 16 blocks of 4x4.

TABLE 6.6: Grid for level one.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

In level two the algorithm looks for the route to go from block 5 to block 9. It searches for a path in the same row if the direction is left or right, and in the same column if it is up or down. If the cell in the border in the same row or column is not available, it uses look up tables LUT to define the order it checks other cells in the edge.

It starts in cell 118 in block 5, since it is already in the border with block 9, it could be the route between these two blocks. But before, it has to satisfy two conditions; 1) its neighbor of interest should be available (cell 134); 2) there should be a path to go to one of the cells in the border between blocks 9 and 13 (180, 181, 182 or 183). If it is, it appends 118 to the solution and save 134 as a start for the next search if not the algorithm tries with the other cells in the border. Our experiments show us that satisfy condition 2 reduces re-processing, therefore, the time execution.

Now in block 9, the partial start is cell 134. Cell 182 is the first option as a partial goal. It would change the partial goal only if it does not satisfy the two conditions mentioned early. In our example, this was the case. Hence, the new partial goal is cell 181. The next partial start is cell 197, since it is the last block the real final goal which is cell 245 is used. Figure 6.3 shows the path for level two over the grid.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

FIGURE 6.3: Path example in level two.

Dynamic programming principles are applied. Blocks of 4x4 cells are the Optimal Substructure. The conjunction of the partial paths is the solution to the main problem. The algorithm also applies Overlapping Subproblems; when the partial paths are not pre-processed they are saved them as they are computed; in this way, the paths can be reutilized for next searches. When there is pre-processing the algorithm updates the path in the LUT when is reprocessed, in this way it gets better results.

Initially, all cells in the border were evaluated to find the best option; however, relax the search process gives a better performance; there is a trade-off between performance and optimality. Figure 6.4 show one example path for different levels of abstraction in map brc997d.

Sometimes the path found in the first levels does not have a solution in the last levels. For example, if the algorithm does not find a path in level three, the connectivity is updated, and it looks for a different path in level two, it exhausts all options in level two if it does not find a solution it looks for a different path in level one. During the development were identified some cases where the algorithm does not find the solution, the algorithm can not find a path like this 5-6-5-9, it can not return to a block evaluated already, in this case, number 5. The author is exploring alternatives related to the way the algorithm creates the hierarchical structure to solve this issue.

6.5 Experimental Results

Our experiments were performed on a computer with a processor Intel Core i7-7700HQ with 8 cores and 7.7 Gb RAM. The operative system is Ubuntu 16.04.5 LTS (Xenial Xerus) 64-bit, The compiler was g++ 5.4.0. To implement A* it was used the University of Alberta's pathfinding library [106].

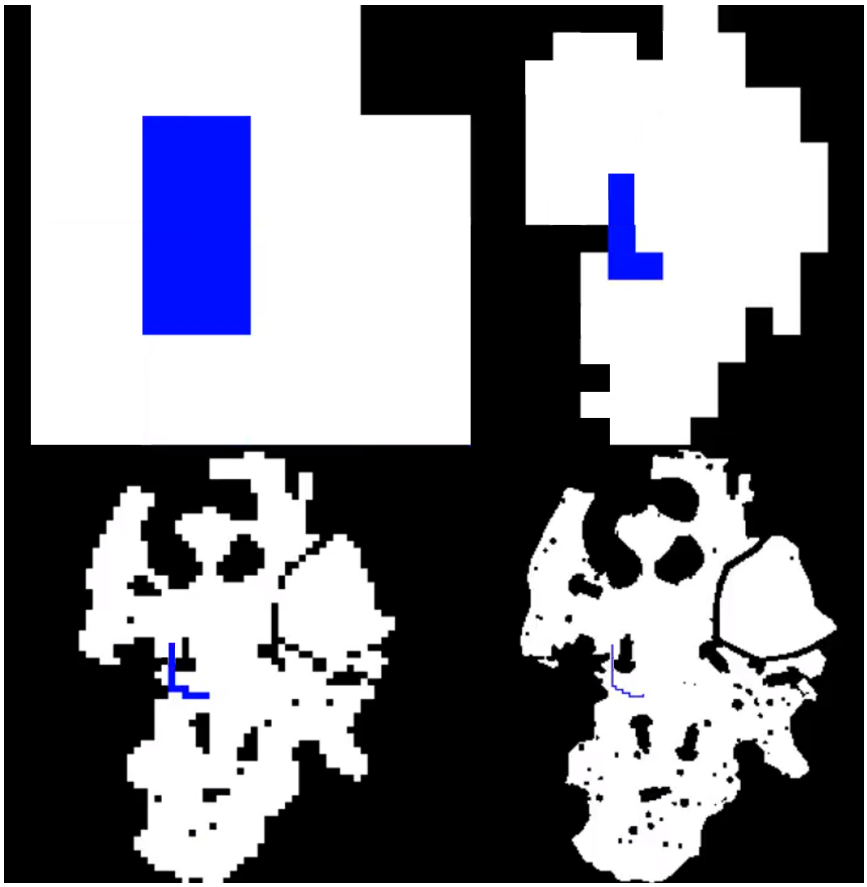


FIGURE 6.4: Path example in different levels in map brc997d.

Maps from video games and urban scenarios were used. Figure 6.5 show all of them. They were taken from the Benchmarks for Grid-Based Pathfinding [104]. A total of 3746 experiments were performed, they were randomly selected from the benchmarks.

6.5.1 Speed-up Analysis

Table 6.7 shows the values for the execution time. The execution time and path length average values of A* and our approach (Hierarchical Path-finding based on Vola) HPV with and without pre-processing paths are presented.

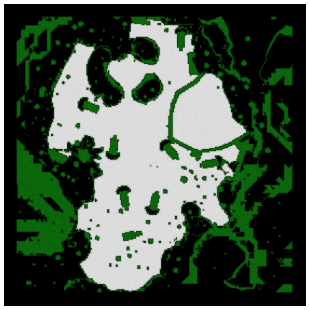
TABLE 6.7: Execution time comparison

Algorithm	Avg Execution Time [ms]	Speed-up	Avg Path Length
A*	6.5	1	225
HPV Pre	1.1	5.9	240
HPV	22.07	0.29	238

If the paths are pre-processed the algorithm gets a speed-up of 5.9, however without pre-processing our algorithm is slower than A*. As was said hardware acceleration would solve it. Otherwise, if there is not available specific hardware for this

(A) Map from the video game: Baldurs Gate II.

(B) Map of Milan.



(C) Map from the video game: Dragon Age: Origins

(D) Map of Shanghai.

(E) Map from the video game: Starcraft.

(F) Map of Sydney.

FIGURE 6.5: Maps used for our experiments.

process, but it is available a multicore processor that is common in commodity hardware nowadays, the pre-processing of paths can be performed in parallel. If a GPU is available this time would be considerably reduced.

Figures 6.6, 6.7, 6.8 shows the execution time versus the path length.

In these charts, it is possible to observe a directly proportional relationship between path length and execution time for A*. In advance it was known that the execution time of A * grows proportionally with the search space, however, in our case, this relationship is not explicitly shown, as we can see in figures 6.7 and 6.8.

An in-depth analysis shows that the execution time of the proposed algorithm increases when a new path is processed. Either because it has not been pre-processed or it is re-processed because the first option did not work. This is an advantage since the execution time of our algorithm does not grow linearly concerning the search

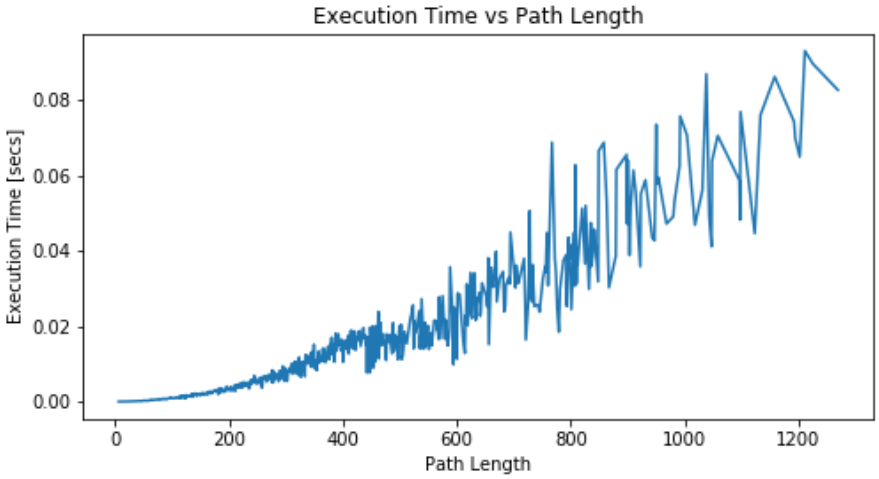


FIGURE 6.6: Time execution vs Path length for A*.

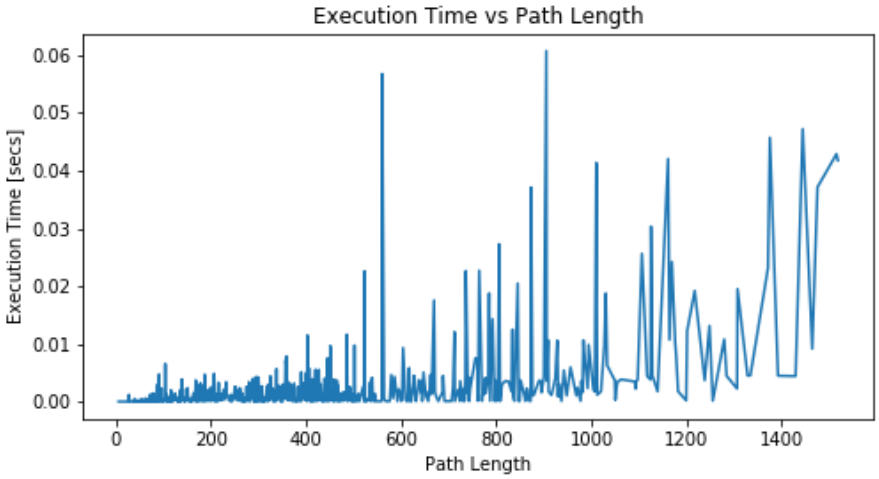


FIGURE 6.7: Time execution vs Path length for HPV with pre-processing paths.

space. And it gives us the possibility of improving it by identifying the most popular and successful routes to reduce the number of reprocessed cases.

The number of times a path is re-processed was calculated in relation to its length. It is possible to see the results in Figures 6.9, 6.10. It is especially clear for HPV pre-processing paths, that does not show a clear relation between execution time and path length, but it shows between re-processing times and path length.

6.5.2 Pre-Processing Time Analysis

The pre-processing stage includes: create the auxiliary maps, compute density values and connectivity. Pre-processing time is reported based on the size of the maps. Also, it was reported the time for pre-processing all paths. Table 6.8 shows the values.

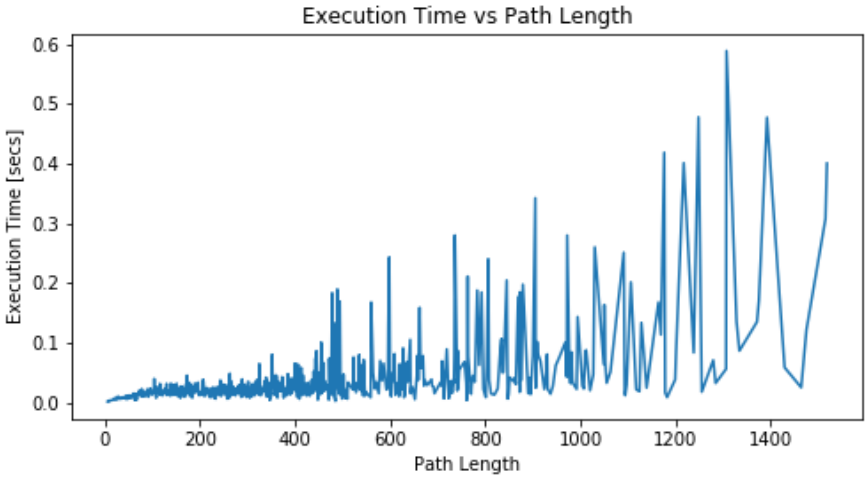


FIGURE 6.8: Time execution vs Path length for HPV without pre-processing paths.

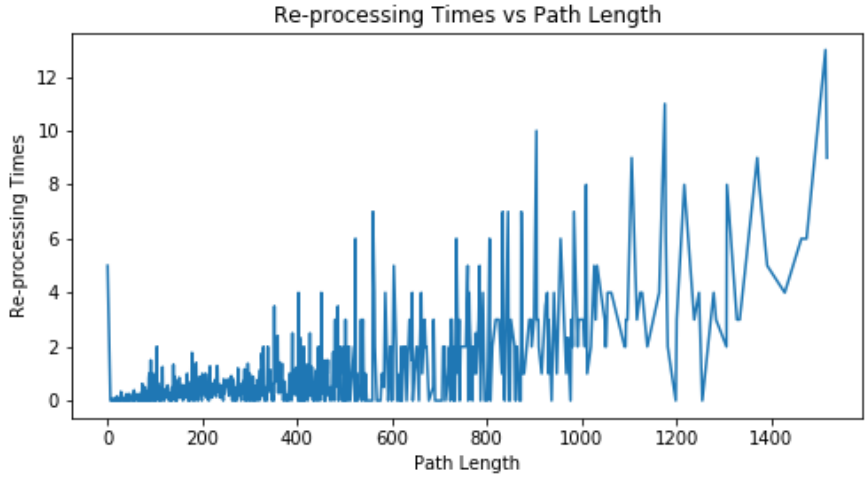


FIGURE 6.9: Re-processing times vs Path length for HPV with pre-processing paths.

6.5.3 Use of Memory Analysis

In this subsection is presented the memory space the algorithm uses for auxiliary structures. Memory space is proportional to the size of the map. A maximum resolution of 1024x1024 was considered because is the highest resolution included in our tests.

One bit is used for each cell of the auxiliary maps. A float number is used for each cell to store the density value. Unsigned integers are used to store connectivity. One unsigned number can store the connectivity of 16 cells. We do not compute connectivity or density for the last level. Table 6.9 shows the size in Kilobytes for each level. The low consume of memory make the algorithm optimal for using it in embedded systems.

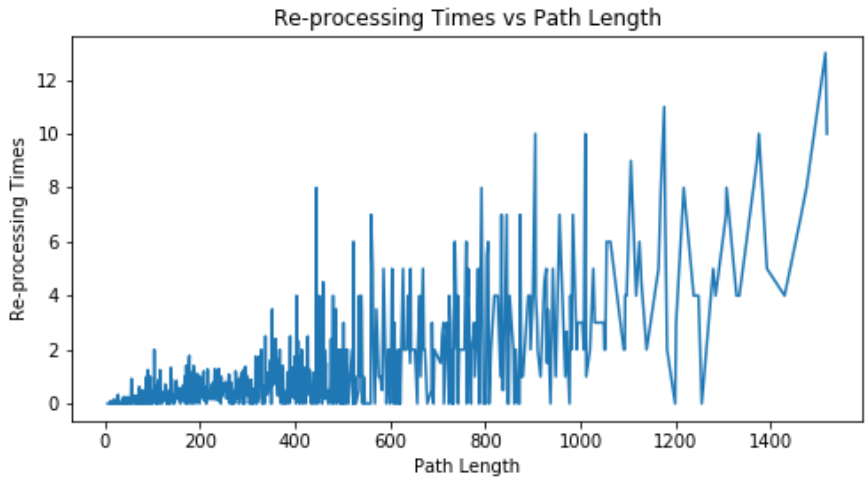


FIGURE 6.10: Re-processing times vs Path length for HPV without pre-processing paths.

TABLE 6.8: Pre-processing time for different map resolution.

Map Size	Time [ms]	Paths [mins]
64x64	0.29	0.17
256x256	0.96	10.27
1024x1024	12.42	97.65

TABLE 6.9: Storage size of auxiliary structures

Level	Resolution	Auxiliary maps [kB]	Density [kB]	Connectivity [kB]
1	4x4	0.002	0.062	0.007
2	16x16	0.032	1	0.125
3	64x64	0.5	16	2
4	256x256	8	256	32
5	1024x1024	131	NA	NA

6.5.4 Optimality

To evaluate optimality, the author compares the path length obtained with the algorithm versus the path length obtained with A*. The average path length values are shown in Table 6.7. The error measures the overhead in percents and is computed with the following formula:

$$e = \frac{hpl - apl}{apl} \times 100$$

Where hpl is the path length obtained with our algorithm, and apl is the path length obtained with A*.

Also suboptimality was calculated which is the parameter used in the competition of path-finding [93] to be able to compare it. To get the value this formula is used:

$$so = \frac{hpl}{apl}$$

TABLE 6.10: Optimality Percentage Error and Suboptimality

Algorithm	Optimality Pct Error	Suboptimality
HPV Pre	6.66	1.06
HPV	5.77	1.05

Suboptimality is within the range published in the competition, which minimum is 1, it means that the path is optimal and the maximum is 2.16, reported by one of the fastest entry, as there is a trade-off between performance and optimality. The acceptable degree of optimality would depend on the application.

6.6 Summary

In this chapter the author presents a new hierarchical path-finding algorithm, this approach offers significant advantages for crowd simulation: it can return a partial path instead of wait for a complete path, which allows a character to start moving, while the rest is computed asynchronously. It can reprocess a partial path or a section of the map with different levels of abstraction enabling better use of resources.

This approach operates directly on the grid or bitmap rather than first converting it to a graph. It makes it suitable for hardware acceleration for real-time operation. It is domain-independent, and it could be adapted to dynamic scenes. It has a low memory consumption which is preferred in embedded systems with constrained resources for applications involving robots or drones.

By implementing part of the algorithm on hardware or apply parallel programming techniques, it is possible to eliminate the pre-processing paths stage. It could increment the speed-up orders of magnitude. Even without these factors, our results show a speed-up near to 6x in comparison with the typical A* approach. Besides the computation cost does not have a proportional growth concerning the search space. However during the development were identified some cases where the algorithm does not find the solution. The author is exploring alternatives to solve this issue.

Chapter 7

Modelling Crowds in Urban Spaces

7.1 Introduction

Humankind is in the era of big cities, which is confronted to emergency situations that require the intervention of qualified personnel to generate an orderly and safe urban experience (e.g., massive exodus, natural disasters, concerts, sports events, protests). In this context, data management and visualization techniques can support real-time observation for understanding the behavior of people and thus help in the development of security strategies (e.g., by supporting on-line and post-mortem analytics for guiding the process of recommendation and decision making in real-time).

The author presents an approach for simulating crowd behavior for supporting control in public spaces. The objective is to visualize and predict the behavior of individuals and groups moving and evolving within real environments. For this purpose, geo-located data produced by mobile devices and other sources of information (e.g., security cameras, drones) are used to predict individual and crowd behavior and detect abnormal situations in the presence of specific events. The challenge of combining all these individual's location with a 3D rendering of the urban environment is also addressed. The data processing and simulation approach are computationally expensive and time-critical, the project rely thus on a hybrid Cloud-HPC architecture to produce an efficient solution.

The Chapter is organized as follows. Section 7.2 gives an overview of related work concerning individuals' location harvesting and crowd simulation techniques. Section 7.3 gives an overview of our approach consisting of two contributions: modeling and locating individuals within crowds and simulating crowd behavior in natural and urban spaces. Section 7.4 gives the general lines of experimentation that were conducted for following and predicting individual and crowd behavior in urban areas. Finally, section 7.5 presents the summary of this chapter.

7.2 Related Work

Studying the crowd is in the heart of research in different computer science disciplines. This chapter combines results from visualization and database domains that have tackled the concept in different perspectives.

7.2.1 Data visualization and Crowd Simulation

It is possible to use a simple visualization of trajectory data to support decision making. However, combining data analytics with the more complex crowd simulation and visualization techniques that have been used in the entertainment industry, where one must create populated environments that seem realistic, is a meaningful way forward.

One of the challenges in crowd simulation is modeling realistic behavior of crowds within an environment (e.g., an army within the battle camp in a shooter game, fans moving to attend a concert or a football match). Rendering, visual variety, character animation, artificial intelligence and motion planning are common problems tackled in visualizing crowd simulations [7].

Crowd simulation is an important research direction in computer games, movies, and virtual reality [107], urban planning, education, and security, among others [108]. Large crowd simulation is in general coupled within the execution of another system, for instance, a video game. Thus, it is often restrained by the limited processing time available. Therefore, the use of GPUs has been proposed for achieving real-time crowd simulations. Realistic simulation of crowd movement can be obtained by observing and mimicking "real crowds".

7.2.2 Data harvesting and analytics for monitoring crowds

The author is interested in techniques that use crowdsourcing (explicit and implicit) for collecting data that contain information about the way people evolve in public and private places. These data collections can be used as input for learning crowd behavior and simulating it in an accurate and realistic manner. The advance of location-acquisition technologies like GPS and Wi-Fi has enabled people to record their location history with a sequence of time-stamped locations, called trajectories.

Some work has been carried out using cellular networks for user tracking, profiting from call delivery that uses transitions between wireless cells as input to a Markov model [109]. Wolf and others [110] used stopping time to mark the starting and ending points of trips. The comMotion system [111] used the loss of GPS signals to detect buildings. When the GPS signal was lost and then later reacquired within a certain radius, comMotion considered this to be indicative of a structure. This approach avoided the false detection of buildings when passing through urban canyons or suffering from hardware issues such as battery loss. GeoLife [112] introduces a social networking service, which aims to understand trajectories, locations, and users, and mine the correlation between users and places in terms of user-generated GPS trajectories. GeoLife offers three key applications scenarios: 1) sharing life experiences based on GPS trajectories; 2) general travel recommendations, e.g., the top interesting locations, travel sequences among locations and travel experts in a given region; and 3) personalized friend and location recommendation.

Existing work in robotics and autonomous vehicles applies automatic learning techniques for making them independent while they evolve in open spaces. They often use collected data for example, for training classifiers to reproduce the behavior of the crowd in a synthetic environment. One of the challenges in data analytics is to

predict by deducing behavior models of the observed subject. A model is a collection of data on some particular aspect of a subject's behavior that, when associated with a limited set of contextual clues, yields predictions on what action the subject will engage in next.

Based on this notion, there is work similar to [113] that use location as a context to infer other data such as the presence of other people. Predestination [114] is an approach that leverages an open-world modeling methodology that considers the likelihood of users visiting previously unobserved locations based on trends in the data and on the background properties of places. Multiple components of the analysis are fused via Bayesian inference to produce a probabilistic map of destinations. The proposed algorithm was trained and tested using a database of GPS driving data gathered from 169 different subjects who drove 7,335 different trips.

The challenge is integrating computationally expensive data analytics with realistic character visualization within realistic environments giving the impression of reality to data analyzers who will be able to make critical decisions.

7.3 Using big data for observing crowds

Figure 7.1 illustrates the general overview of our approach that addresses three main problems: (i) data harvesting, (ii) crowd simulation and analytics and (iii) visualization. Concerning data harvesting, it is possible to use different data sources like social networks, mobile devices data, and public cameras. Right now, the project uses existing temporal geo-located observations concerning individuals' trajectories.

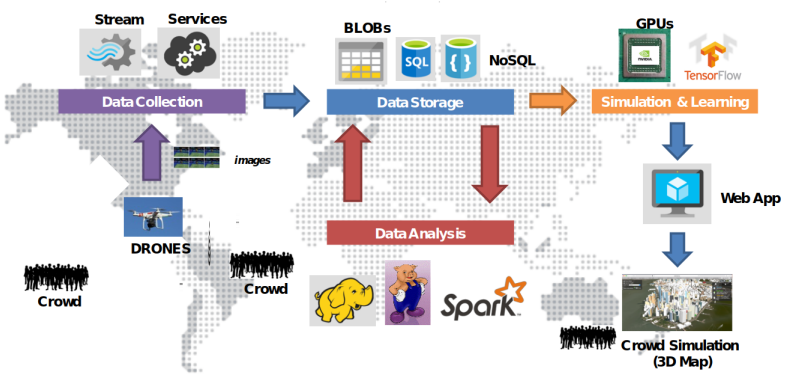


FIGURE 7.1: General overview of the approach.

Building and storing representative data collections about individuals and crowd behavior in urban spaces can be useful for performing offline analytics to discover patterns, relations, and identify those members that might not belong to the group or that might have suspicious behavior. Data and analytics results must be stored in different supports according to the conditions in which they are shared and exploited. The project proposes data storage tools that use data sharing, geographical position, and other context-related strategies to distribute data across different stores, index to various characteristics (time aspects), and efficiently deliver data to data analytics processes.

7.3.1 Identifying and profiling the crowd

Data analytics techniques (temporal and spatial reasoning) are applied for computing trajectories and for identifying crowds. That is, people grouped in a proximity that adopt a specific "behaviour" referring to four well known naïve crowd patterns: (i)casual crowd which is loosely organized and emerges spontaneously, people forming it have very little interaction at first and usually are not familiar with each other; (ii)conventional crowd results from more deliberate planning with norms that are defined and acted upon according to the situation; (iii) expressive crowd forms around an event that has an emotional appeal; (iv) acting crowd members are actively and enthusiastically involved in doing something that is directly related to their goal.

The objective of our analytics study is to identify a triggering "symptom" that can evolve into the constitution of a crowd. For example, someone showing a banner in the middle of a plaza, in front of some monument or government office; people density increasing in some area. In our approach and since for the time being the project do not use images recognition the project address crowd creation identification by measuring people density in specific spatial regions during a time interval. This requires a continuous analysis of the evolution of the status of urban space to measure the population density.

Density measuring is done using different data collections: (i) the continuously harvested observations of the geographical position of individuals (that accept sharing their position) along time; (ii) the images stemming from cameras observing specific "critical" urban areas, like terminals, airports, public places and government offices; (iii) data produced by social networks and applications like Twitter, Facebook, Waze and similar. The occupation density of specific urban regions is measured separately according to the political organization of the space (quarters, areas) in every database. Sliding windows to partition continuous data flow with respect to time intervals are used, and the political division of the urban space is used for filtering, grouping the data and computing density per urban region. This is a straightforward yet somehow costly computation not because of the amount of data but because it should be continuously computed, and both data and density results are stored for performing other analytics processes.

Having different visions of the density of urban spaces given different data collections enables to perform other types of data analytics on the n-tuples region-density and to cluster regions both taking into consideration their density and their geographic position. Accordingly, it is generated a "crowd heat thermometer" showing the crowd dynamic distribution view of the urban space that evolves along time.

Having an insight within the crowd. Not all crowds need to be managed within urban spaces, there are some that happen every day in public transport, and others need particular attention and must be better profiled. The first challenge is to be able to discriminate. Using data collections harvested from social networks, some correlations are computed to identify connected people that were in the same urban place at some time interval, and that might have been part of a crowded event. Not all individuals sharing the same spatial region necessarily participate in a crowd.

Thus, the first operation to solve is given a set of individuals located in the same geographic region at the same time interval, whether an individual located at the same space-time belongs to the crowd. A naïve way of evaluating this predicate belongs-to is knowing whether there is a "social" connection between an individual and at least one of the crowd members. Using this analytics operation, it is possible to draw an urban occupation map and propose some connections among individuals occupying the same urban region in the same period.

The objective of this classification is to identify possible outsiders, that is, individuals that do not belong in the group. Those are the ones in which there is a particular interest because they could be troublemakers for example.

In a more in-depth study, the project could adopt two complementary strategies for defining possible networks hidden in a crowd. First, the contacts graphs of individuals social networks accounts which have been identified in a crowd are retrieved. It is possible to establish relations with other contacts using well-known relations discovery techniques as the one proposed by [115] and filtering strongly related contacts to verify whether they are themselves present in the crowd. The process is computationally expensive because graphs intersections of possibly thousands of individuals are computed. It is incremental in the sense that graphs are stored and they are enriched with new information coming from the observation of other crowds in other moments. These graphs versions are used for profiling and predicting crowd behavior.

Profiling and predicting crowd behavior. Once the crowd has been formally identified and modeled in terms of its participants, it is possible to observe its behavior as if it was an individual: (i) characterize its elements (find the leaders, the followers and eventually the outsiders); and (ii) predict the evolution of its behaviour for example, probability of conflict, space utilisation and risk maps.

Once it was created a first connection graph describing how people possibly participating in the crowd are connected among each other, it is possible to determine which is the degree of influence of each participant of the crowd. This is done by computing the influence of the elements of the crowd towards other elements according to their contacts network, and that can eventually be participants in the crowd. The analysis ends up with groups of outsiders that have to be inspected to understand their presence in the crowd, and their possible role in the event. Again, since the inclusion of participants in the crowds evolves along time, the contacts' network represented by graphs and the influence of participants varies too. The computational cost is significant considering semi-post-mortem computations. Of course, the ultimate objective is to be able to observe this evolution in real time which introduces scalability problems that must be addressed with GPU architectures.

The final task is to predict the behavior of the crowd. Therefore, notions of space occupation and the probability of conflict are used, by searching behavior patterns in the evolution of the crowd status: it emerges with some individuals, increases its size, in achieving the maximum of participants and then it fades. This "life cycle"

happens within a space occupation process that can be controlled by its inherent behavior but that can also be determined by external factors, e.g., police, troublemakers. We define the status of the crowds with a set of attributes including, the approximate amount of participants, the main trajectories of the group, the spatiotemporal region occupied by the group, the possible outgoing directions in which participants can move within the urban space, triggering and a termination event.

Instead of delivering textual or graphical results of these analytics operations the project aim to provide 2D and 3D visualizations that can reproduce the observations and simulate the behavior of the crowd according to real data. The following section explains how.

7.3.2 Simulating and visualizing large crowds in real time

There are several steps in the process of simulating and visualizing large and varied crowds in real time for consumer-level computers and graphic cards (GPUs). Animating varied crowds using a diversity of models and animations (assets) is complex and costly. It is necessary to use expensive models, take a long time to model, and consume too much memory and computing resources. The project proposes more efficient and cheaper methods for simulating, generating, animating and rendering crowds of varied aspect and diversity of behaviors.

In general, the principle consists in mapping human perception of the space stemming from cameras and expressed in geographical coordinates (latitude, longitude), for example, into pixels. For instance, as shown in Figure 7.2, "give me the GPS coordinates of the users evolving in Beijing ordered by time". Once this query has been evaluated by the appropriate data processing infrastructure, Pig Latin [116] was used for this task, results are transformed into the appropriate format. Textures and maps are retrieved to create the 3D space where individuals' movements will be visualized (simulated) according to the observed information. The visualization of a dynamic situation requires computing capacity to be sure that the rendering is realistic. Thus, for scalability, the project relies on a simulation cluster devoted to the execution of this costly task.

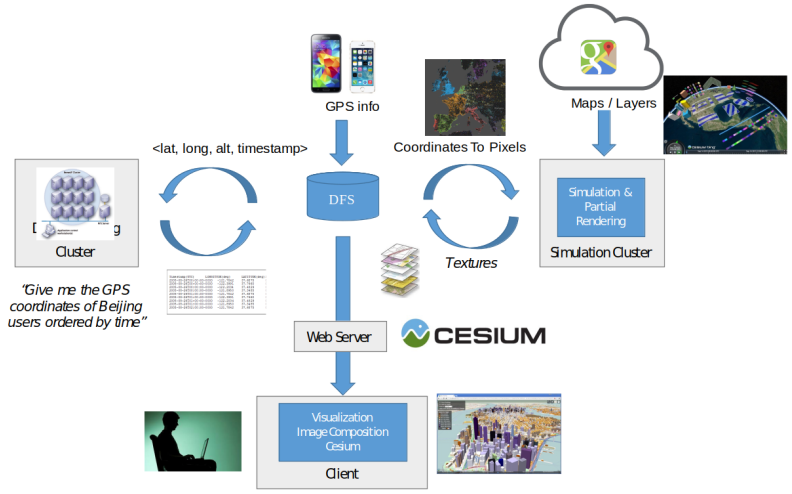


FIGURE 7.2: Visualization process of individuals movement within urban 3D spaces.

Simulation and visualization are based on the work described in [17], modified to integrate Geolife data. The simulation algorithms have been modified such that when given a choice, simulated vehicles and pedestrians will use heatmaps derived from the dataset to follow the most popular routes. Visualization has also been modified to include vehicles as well as pedestrians and to be rendered aligned with and combined with the scenery provided by CesiumJS [117].

The visualization does not only contain the view of the urban space. It also renders individuals moving in it. The level of detail that a data consumer can see about people should depend on her access rights to personal information and on the privacy laws of the geographic space in which observed people are moving. For example, a consumer with few access rights would see people as points or avatars (like video game characters) moving in the street. A consumer with enough access rights should see the actual person walking in a given street.

7.4 Experimentation

The project aims to exploit data collections about the way people move in public places to learn about transit and behavior, later it uses this knowledge to simulate a crowd realistically. For the graphic representation, it combines a 3D view of the urban space and animated characters moving on the scene.

The GeoLife GPS trajectory dataset [118] with data of 182 users, 17,621 trajectories of ca. 1.2 million Km. and 48,000+ hours was used. These data are used to compute spatiotemporal people flows in real crowds to provide data-driven on-line crowd simulation, enhanced with real places geometric data running on GPU and HPC Infrastructure. Since the data set for a given place and time is sparse, the project adds agent-based microsimulation to complement the actual trajectories in the dataset by using all the trajectories in similar moments that are available in the dataset to derive the most probable trajectories for the simulated vehicles or pedestrians.

For the visualization CesiumJS [117] was used, it is an open-source JavaScript library for 3D globes and maps, CesiumJS help us to integrate vehicles and characters in the urban scene.

7.4.1 Computing trajectories

Big Data cleaning and processing tools were used, particularly the language Pig Latin for computing such trajectories. The process is done by defining four declarative expressions as shown in Figure 7.3.

- Step 1: Load the observations clean and prepare the data collection concerning the initial meta-data of the observations. For computing trajectories, three attributes were necessary, the initial and termination times and the transportation modes. The idea is that a trajectory is defined as a set of locations observed at a given moment identified by a time stamp. The PigLatin program presented in the figure implements this process.
- Step 2: Load the GPS logs filtering the latitude, longitude and time stamp according to a predefined schema defined for this purpose.

```
# ------------------------------------------------
# --  Step 1: Load Labels (Trajectories META-DATA)
# ------------------------------------------------

labels = sc.textFile(labelsFilePath)

# Remove file header
labels = labels.filter(
    lambda line: not "Start" in line
)

labels = labels.map(
    lambda line: {
        "start_time": datetime.strptime(line.split("\t")[0], "%Y/%m/%d %H:%M:%S"),
        "end_time":   datetime.strptime(line.split("\t")[1], "%Y/%m/%d %H:%M:%S"),
        "transportation_mode": line.split("\t")[2]
    }
)

# ------------------------------------------------
# --    Step 2: Load GPS logs
# ------------------------------------------------

GPS_logs = sc.textFile(logsPath)

# Remove file header
GPS_logs = GPS_logs.filter(
    lambda line: len(line.split(",")) == 7
)

GPS_logs = GPS_logs.map(
    lambda line: {
        "latitude":  float(line.split(",")[0]),
        "longitude": float(line.split(",")[1]),
        "altitude":  float(line.split(",")[3]),
        "timestamp": datetime.strptime(line.split(",")[5] + " " + line.split(",")[6], "%Y-%m-%d %H:%M:%S")
    }
)

# ------------------------------------------------
# -- Step 3: Find trajectories
# ------------------------------------------------

CR = labels.cartesian(GPS_logs)

CR = CR.filter(
    lambda t: t[0]["start_time"] <= t[1]["timestamp"] and t[0]["end_time"] >= t[1]["timestamp"]
)

GR = CR.groupBy(
    lambda t: (t[0]["start_time"], t[0]["end_time"], t[0]["transportation_mode"])
)

Trayectories = GR.map(
    lambda t: {
        "transportationMode": t[0][2],
        "startTime": t[0][0],
        "endTime":   t[0][1],
        "coordinates": [z[1] for z in t[1]]
    }
)
```

FIGURE 7.3: Computing trejectories.

- Step 3: Finally, the third query estimates the trajectories as sequences of locations where the sequence is determined by the time stamps. With these computed trajectories it is possible to perform other analytics operations and reconstitute the movement of people in the corresponding urban space, in this case, Beijing.

Even if they seem simple due to the expression power of Pig Latin, the execution of these queries can be computationally costly given the volume of data processed. The Pig Latin execution environment was installed in a cluster of 8 machines and executed in parallel. Thus results are obtained in reasonable execution time. The computed trajectories were stored in an integrated data collection on top of which we performed other operations as described in the following lines.

7.4.2 2D tracking of individuals

The objective is to track individuals and crowds in urban spaces. Heat maps are computed to aggregate the trajectories of users during specific time intervals. The result, as illustrated in Figure 7.4 shows those itineraries in red or thick lines. This map shows the most popular trajectories of a person, at the same time could help us to infer habits or predict trajectories in the future, or cluster persons with similar routines. These heat maps concern the Beijing trajectories computed using the Geolife GPS trajectory dataset.

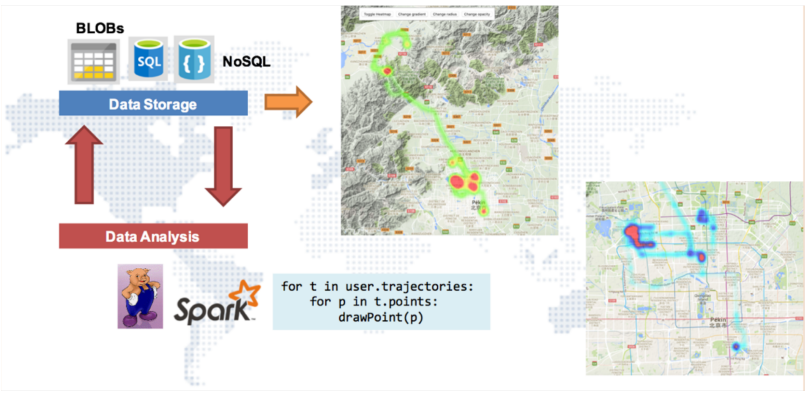


FIGURE 7.4: Heat maps aggregating the trajectories of one person during an interval of time.

Heat maps enable tracking individuals within a specific urban space and see how they move during a particular period of time. The figure shows the simple code for computing heat maps that again serve as new information on which it is possible to perform more analytics. These analytics concern, for example, identifying the most visited regions, observe rush hours in specific areas, and eventually identify crowds. This information could help to improve services and infrastructure in a city. For the time being our experimentation is done in the post-mortem.

7.4.3 3D visualization of individual tracks

The visualization problem is state as follows. It is centered on the user that accesses a 3D virtual space and navigates in it, in our case, for following individuals or the crowd within the simulated urban space. Every interaction of the user with the 3D space triggers a process in which data are retrieved from storage support (memory or disk) to feed the simulation module that will reproduce a realistic behavior of the 3D space. A realistic behavior in our case means that the individual or the crowd will move and behave smoothly as it is seen in "physical reality" and it will react to the environment if some event is produced (due to the interaction of the user). In the case of our experiment for individuals this implies maintaining the position of the individual on the road, respecting the traffic signals (e.g., semaphores, speed, direction), and the organization of the urban space (e.g., buildings, bridges, roads, parks, and other open spaces).

Tests were performed using a workstation with an Intel Core i7-4820K CPU 4-cores at 3.70 GHz, running Linux operating system with 16 GB of RAM and 10MB of cache memory. It includes a GeForce Kepler GTX TITAN Black with 2880 cores and 6 GB of GDDR5 memory.

7.5 Summary

Crowdsourced location data is used to compute spatiotemporal people flows in real crowds. This chapter presented the general approach for simulating crowd behavior and thereby supporting individuals' and crowd behavior in public spaces. The main contribution is combining location-based data collections previously harvested together with online geo-tagged data for visualizing crowds at different levels of

precision and detail according to access control and privacy constraints. The project combines both to provide data-driven on-line crowd simulation, enhanced with realistic urban scenarios. Our data processing and simulation process are computationally expensive and critical; thus, it relies on hybrid cloud-HPC infrastructures for producing an efficient solution.

Chapter 8

Humanitarian Logistics and Cultural Diversity within Crowd Simulation

8.1 Introduction

Human stampedes caused by panic and overcrowding frequently occur because of abnormal events (e.g., a fire or explosion) produced in collective events (e.g., religious pilgrimages, music concerts, and sportive events). These events provoke panic, thus people try to escape pushing each other without realizing that others are being crushed. Since crowds can consist of individuals with diverse physical (i.e., anthropometric) and social characteristics determined by cultural diversity, it is difficult to configure the space in advance and find solutions in real-time to save people and reduce catastrophe.

Crowd simulations for humanitarian logistics provide a means for studying crowd behavior under different conditions. After studying video pedestrian patterns, behavioral scientists have concentrated on the rules that people are accustomed to using in their daily lives and transform them into heuristic formulas. They have focused mainly on four types of danger associated with crowds: (i) trampling and crushing at religious sites; (ii) trampling, crushing and drowning on ships at sea or waterways; (iii) crushing in massive concerts or enclosed spaces such as clubs; (iv) contingency situations of nature, as can be earthquakes, floods, avalanches or landslides, that cause destruction to human-made structures.

In his fruitful work, Helbing has proposed a simplified model of pedestrian interactions: the "social force model." He has addressed crowd dynamics, typically the formation of large-scale spatiotemporal patterns of motion, when many pedestrians interact with each other. Helbing et al. [57] found that high-density flows (up to 10 people per square meter), can result in a phenomenon called crowd turbulence and can trigger the trampling of pedestrians. To prevent such disasters high-density flows must be avoided, particularly the three conditions that can provoke a disaster: (i) low infrastructure capacity, (ii) large numbers of pilgrims, and (iii) merging flows, intersecting flows, or counterflows, especially in reduced spaces. If many pedestrians head toward or pass through a location with low infrastructure capacity in a short time, this location can become a bottleneck with high densities and a high risk of crowd turbulence.

Numerous occurrences of stampedes in the recent past have shown that only coordination between public safety organizations and urban planners cannot solve the problem of managing large crowds. Prediction of pedestrian-flows in flat areas, and buildings with an exceptional architecture or in challenging evacuation situations, have been addressed by simulation models (e.g., queuing models [119], transition matrix models [120], and stochastic models [121]). Also, there are models for pedestrians route choice behavior [32], [122]. Helbing observed that none of these modes adequately considers the self-organizing effects that appear in a crowd.

Addressing a realistic simulation of human stampedes calls for a combination of methods. This research focuses on anthropometric and sociocultural aspects identified in the last stampedes occurred in Mecca, and demonstrate their impact in a catastrophe simulation (e.g., in the holy places of Islam).

This chapter proposes an approach to explore the impact of anthropometry and cultural diversity in the behavior of crowds in panic situations. Human density makes people come into physical contact creating behavior that can be understood as an exchange of real physical forces. Our approach models crowd using microscopic simulation using multi-agent systems, where dynamics are coupled with crowd simulation modified by anthropometric and cultural parameters. Our approach includes techniques for reproducing and simulating the behavior of the crowd to generate models that can help decision-makers to control such situations. The main contribution of our work is to use computational science, data processing and visualization techniques to perform our simulation for eventually supporting critical decision making.

The remainder of the chapter is organized as follows. Section 8.2 introduces our agent-based approach for simulating crowds for addressing human logistics. Section 8.3 describes the experimental validation that was conducted for simulating a case of a stampede and discusses our results. Finally, section 8.4 presents a summary of this chapter.

8.2 A Sociocultural Perspective for Simulating Human Stampedes

Figure 8.1 gives an overview of our approach for simulating stampedes taking into consideration sociocultural and anthropometric perspective.

Our approach develops a multi-agent system to simulate a crowd of multicultural individuals evolving in an urban space. In this context:

- An urban space is a spatiotemporal region. Its configuration models pedestrians' flows of specific densities moving concerning predefined trajectories, through geometric shapes and signals. Space design considers an anthropometric model.
- The anthropometric model describes the physical characteristics of people.
- The organization of the space, the signals, and amenities serving to control the crowd in normal and abnormal situations considering sociocultural aspects.

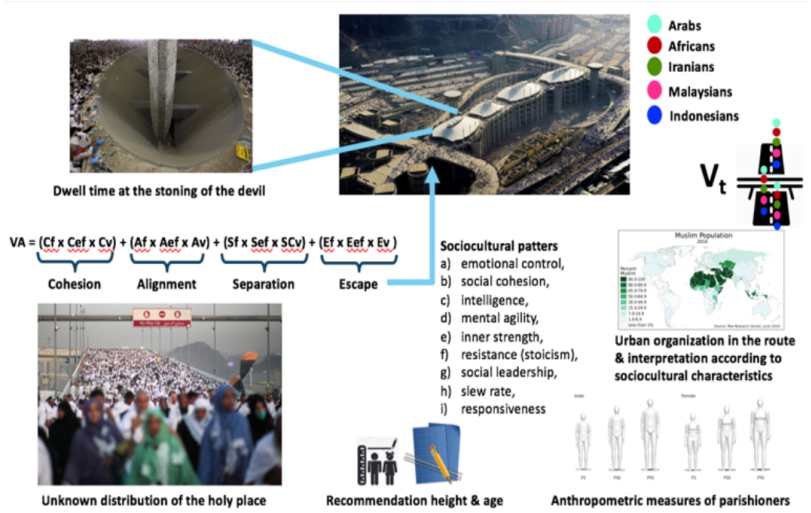


FIGURE 8.1: Simulating and studying stampedes from a sociocultural and anthropometric perspective.

A mathematical model that includes these aspects to predict crowd behavior was developed. It implements rules and constraints representing anthropometric and sociocultural characteristics to make the agents interaction as realistic as possible.

8.2.1 Anthropometric and Sociocultural Model

The variability in the dimensions of the people due to their age, gender and ethnicity creates the need to measure the population with the objective to design public spaces according to their size and phenotypic values considering the standard deviation and percentiles. The phenotypic values and anthropometric measurements are given by existing models that were adopted (see Table 8.1).

TABLE 8.1: Anthropometric measurements considered in our model

Variable	**Unit**	**Variable**	**Unit**
Age	years	Height	cm
Weigth	kg	Body Mass Index (BMI)	(Kg/m_2)
Total Arm Length	cm	Upper Arm Length	cm
Lower Arm Length	cm	Wrist Circunference	cm
Upper Arm Circ. (REST)	cm	Upper Arm Circ. (Contracted)	cm
Chest Circunference	cm	Chest Circunference (Elevated)	cm
Hip Circunference	cm	Thigh Circunference	cm
Calf Circunference	cm	Hand Width	cm
Shoulder Width	mm	Hip Width	mm
Bi-Humerus Diameter	mm	Bi-Femur Diameter	mm
Wrist Diameter	mm	Body Density (BD)	(gm/cm)
Pct Body Fat	%	Total Body Fat (TBF)	kg
Lean Body Mass	kg	Grip Strength (Right Hand)	kg
Grip Strength (Left Hand)	kg		

Another essential aspect to consider particularly for logistics and organization reasons, both in normal and abnormal situations, are sociocultural aspects. For example, in one of the human stampedes in Mecca, two leading forces were identified by analyzing videos. Iranians whose average height often exceed 1.89 and greater muscle mass, and Malians with similar anthropometry but with different languages and sociocultural references. Analysts observed the lack of information in different languages, but also a sociocultural aspect of group formation when walking particularly Iranians and parishioners from Africa.

This behavior is not present in the Saudi Arabian society used to evolve in that environment. It is necessary to see how different groups react to visual signs, since in some cultures visual warnings are not used, and inexperienced travelers, both adults, and children, can get confused and be unable to follow instructions for proper evacuation. Therefore, these characteristics were included in our model. Since, pilgrimages with years of tradition are the focus of this chapter, the project has access to this information: (i) the average distribution of nationalities attending the pilgrimages; (ii) the average distribution of men, women, and children; (iii) the ethnographic and crowd tracking studies, as well as the cultural aspects that can have impact on the behavior of the crowd in the presence of an abnormal event (e.g., a stampede).

These sociocultural aspects are considered in our model: : (i) collective behaviour in public spaces (civility, awareness of traffic signs, respectfulness of traffic signs), (ii) automatic behaviour in dynamic crowds that follow a specific direction (whether people tend to form lines, lineage to the right/left) and (iii) the vital space (the minimum and maximum separation space between two people). For individuals' social behavior in the presence of an abnormal event, the assumptions of existing work discussed in Section 2 are adopted, saying that crushing and pushing depends on the urban facilities and of a kind of irrational behavior of the crowd that depends on the knowledge that people can have of the space. This knowledge can create or not confidence and reduce the degree of panic.

The design of the agent system takes into consideration the anthropometric characteristics and sociocultural knowledge that guides the behavior of the agents when they evolve in the virtual space. Their interaction with the environment and with other agents is guided by the associated sociocultural constraints. Section 8.3 shows how these attributes are interpreted into concrete measurements implemented by agents used for simulating a concrete dynamic crowd.

Figure 8.2 shows the general architecture of the multi-agent system. An agent is a complex autonomic intelligent system that implements an event-reaction cycle guided by the internal knowledge it computes thanks to the data it receives and processes.

The data are produced:

- Continuously: Through the Perception module the agent process the actions around him and react.
- In batch: concerning the general map of the space. These data include the objects in the space, the urban facilities. They are stored in the World model base of the Knowledge module.

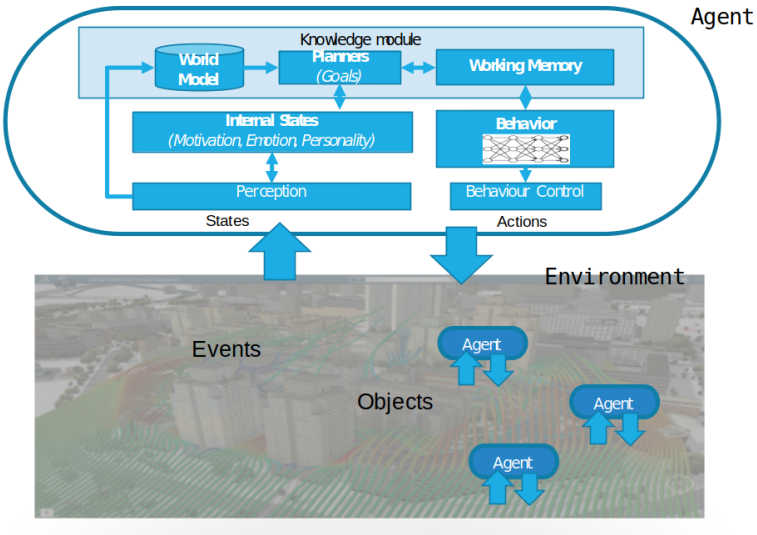


FIGURE 8.2: General architecture.multi-agent system.

The agent processes these data guided by the behavior rules coded in the Behaviour module. The agent plans goals and executes actions that will have an impact on the environment.

The distributed knowledge is used by the multi-agent system that generates a collective behavior. This behavior is implemented in the simulation of a dynamic crowd evolving in a collective event. The agents and environment of our approach have a projection in a virtual 3D world that visualizes their individual and collective behavior.

8.3 Experimentation

Our experiment is related with a religious event in a Muslim world with people from diverse societies, heterogeneous sociocultural characteristics (e.g., different nationalities and have a different understanding of the visual announcements written in Arab) and anthropometry.

Menge [123], a cross-platform, extensible, modular framework for simulating pedestrian movement in a crowd was used. It includes social behaviors models developed by Helbing, Reynolds, these models were presented in Section 2. These models have been recognized to be representative of crowd behavior in urban spaces, and therefore Menge seemed a well-adapted platform.

Menge's architecture is inspired by an implicit decomposition of the problem of simulating crowds into four components: religious masses, mass concerts, shipwreck, and floods. Different combinations of subproblem solutions yield crowd simulators with likewise varying properties. Menge creates abstractions for those subproblems and provides a plug-in architecture so that a novel simulator can be dynamically configured by connecting built-in and bespoke implementations of solutions to the various subproblems.

8.3.1 Use Case: the Hajj

The Hajj, the annual Muslim pilgrimage to Makkah in Saudi Arabia, is one of the largest pedestrian events in the world. It takes place from the 8th to the 13th day of Dhu al-Hijjah, the 12th and last month of the Islamic (lunar) calendar. Each year, up to four million pilgrims approach the holy sites in the region of Makkah to perform their religious duty for many days. In 1950, fewer than 100,000 pilgrims performed the Hajj; in 2005, the number of pilgrims performing it exceeded two million for the first time. Until 2006, several crowd-related disasters led to thousands of casualties. Thus, the Ministry of Municipal and Rural Affairs of the Kingdom of Saudi Arabia (MOMRA) launched projects to prevent future crowd-related accidents.

8.3.2 Experiment Design

To simplify the simulation problem, in the Design of the Experiment (DOE) it was considered that the crowd attending the Hajj is composed of groups belonging to four kinds of Muslim societies: Iranians, Muslim Africans, Muslims Asians no Arabs and Arabs. To obtain an accurate model and social behavior of the crowd our system maintains data about representative individuals in each group and data related to the cultural aspects. The latter is done to distribute in an optimal way for each zone of the mosque associated with a group a specific social behavior.

The main experiment consisted of evaluating individuals from a group of parishioners and performing a simulation on 200 replicas for 100 instances of individuals. This allowed the generation of the best selection of every social group and its possible location in the representation of the DOE (see figures 8.3 and 8.4). The position of each group is obtained by comparing the different cultural and social similarities of each group and evaluating the proposed multiple matching models.

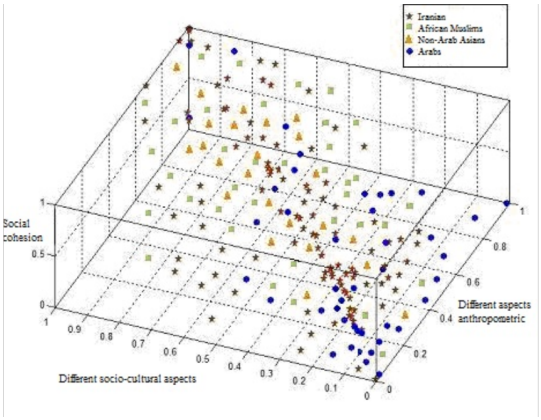


FIGURE 8.3: Spatial representation of sociocultural behaviour categories of the four types of participants in the religious Hajj event.

A weight vector is defined for tuning the fitness function used for simulating people in the crowd:

$$Wi = [0.6, 0.7, 0.8, 0.5, 0.6, 0.4, 0.9, 0.5, 0.4].$$

It represents the importance of the attributes associated with a good parishioner (see table 3): a) emotional control, b) social cohesion, c) intelligence, d) mental agility,

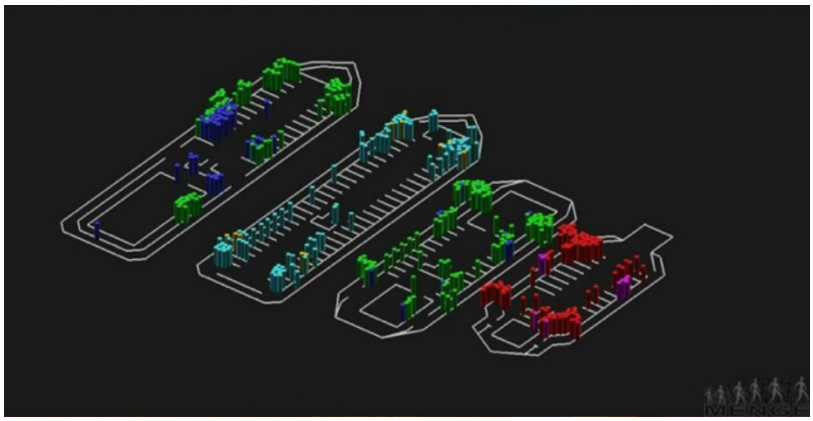


FIGURE 8.4: Distribution of different groups in the Masjid al-Haram mosque during the Hajj.

e) inner strength, f) resistance (stoicism), g) social leadership, h) slew rate, i) responsiveness. For defining these values, the social behavior models proposed by Bogardo [124] were used. They have been used in problems with similar characteristics.

Each attribute is represented by a discrete value between zero and five, where zero represents the null value and five the highest value of the attribute. Besides each attribute is pondered by preference/importance specified by the application context. For instance, for a given analysis, it might be interesting to observe the behavior of the crowd focussing on responsiveness.

The DOE selects the color of each group based on the similarity of two racial groups according to these attributes. A similarity is determined as follows. Given two racial groups the following fitness function determines their similarity:

$$\sum_{i}^{n} abs[corr(W_i Attribute_i^A, W_i Attribute_i^B)]$$

- W_i = vector of weights that represent the importance ponder of attribute i.
- A, B = represent the compared groups
- $Attribute_i$ = vector of the attribute values from each group.

While the correlation value decreases, the similarity between racial groups is stronger.

A threshold of 0.05 is established, if the correlation value is less than the threshold, it assumes that there is no relationship between the groups.

The design of the experiment consists in an orthogonal array test, with the interactions between the variables: emotional control, ability to fight, intelligence, agility, force, resistance, social leadership, and speed. These variables are featuring each possible response to individual behavior.

Table 8.2 shows some possible scenarios resulting from the combination of the values of the attributes and specific sociocultural to represent a problem in a religious place (meaningful representation to participate in the Hajj).

TABLE 8.2: Orthogonal matrix for characterizing the Hajj population.

a	b	c	d	e	f	g	h	i
0	1	2	2	3	3	4	5	1
0	1	2	2	3	4	5	5	1
1	1	3	2	4	4	2	1	2
1	1	3	2	5	3	2	1	2

Each column represents a feature (i.e., a. emotional control, b. social cohesion, c. intelligence, d. mental agility, e. inner strength, f. resistance (stoicism), g. social leadership, h. slew rate, i. responsiveness). Each row represents a racial group considered in our experiment (recall that four racial groups are examined as shown in Figure 8.3).

8.3.3 Simulation Results

The results allow us to analyze the effect of the variables in the color selection of all possible combinations of values reported in Table 8.2. This is the color code correspondance: Iranians in green, Arabs in Cyan, Blue and Pink respectively Indonesians and Malaysians, Africans in red.

Figure 8.3 shows the distribution of agents generated for each group considering their social cohesion, anthropometric and sociocultural characteristics (axis of the 3D space represented in the Figure). It is possible to obtain the speed of walking by measuring the time required to reach a percentage (e.g., 40%) of the desired velocity, starting from rest. This speed is typically in the range of 5 km per hour, but it depends on the pedestrian's current state of fitness and stress, fatigue, drunkenness, inability, group binding, footwear, weight, height, desire to reach a goal, climate, signals, noise, etc. In this case, the following equation was used:

$$v(ti) = \frac{E * K(ti) * \frac{\sum_{j=1}^{n} j\lambda}{n}}{C} \cong 1$$

where:

- E: age
- K: represents the degree of care required depending on agent characteristics. It is given by K = a + b^2 where a denotes expected behavior of the agent given by the degree of disorientation based on the weather; and b denotes the actual behavior of the agent, i.e., its degree of fatigue
- j: denotes an attribute of an agent (i.e., models an agent behavior)
- i: agent
- t: time
- n: number of social attributes per agent
- λ: inability to find an exit
- C: number of relatives.

Figure 8.5 shows that a representative population (agents) was generated. It will be used to simulate their behavior during the Hajj event using the characteristics of the space in which they evolve. Also it is possible to estimate the speed at which they change in the area.

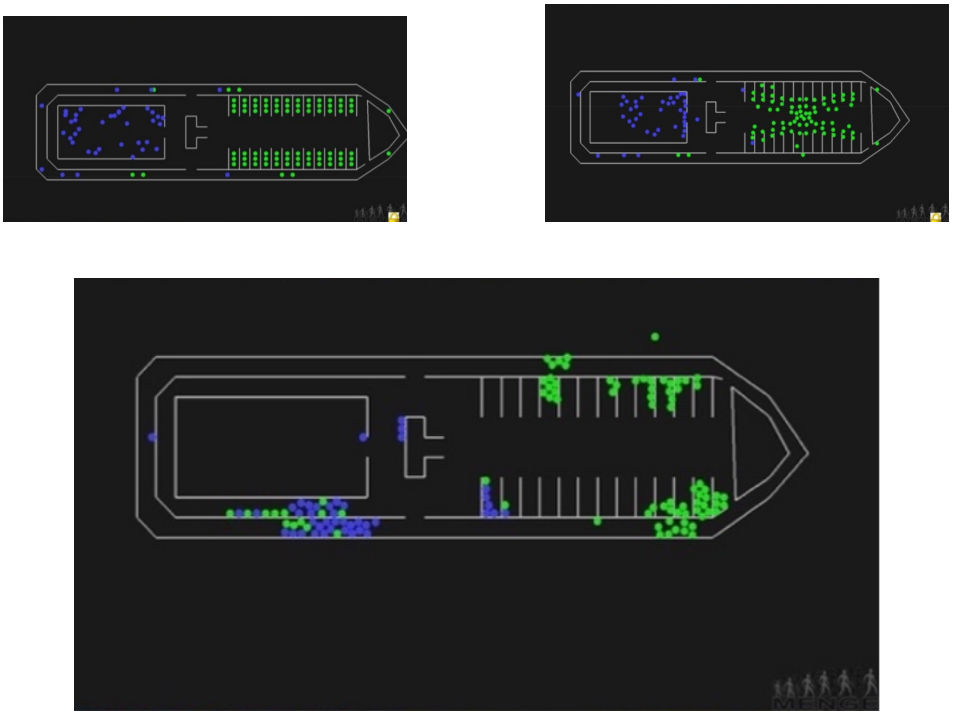


FIGURE 8.5: Evolution of pilgrimage simulation.

Since, the behavior of pilgrims during the Hajj circuit is simulated, they perform in the Masjid al-Haram mosque. It is represented a schematic yet realistic distribution plan of the mosque. To understand the distribution, it is necessary to first explain the way the ritual of the pilgrims is organized in the mosque. The ritual of Tawaf includes: (1) Walking seven times counter-clockwise around the Kaaba. During this circuit, a pilgrim "stops" in front of the Black Stone or at least reduces her pace to point it. A pilgrim performs three rounds in a hurried pace and four in a leisure pace. Tawaf is now also performed on the first floor and roof of the mosque because of the large crowds. (2) Then, a pilgrim does two Rakaat prayers at the Place of Abraham a site near the Kaaba inside the mosque. Because of large crowds during the days of Hajj, they may instead pray anywhere in the mosque. (3) After prayer, pilgrims drink water from the Zamzam well, made available in coolers throughout the Mosque. (4) Finally, a pilgrim runs or walks seven times between the hills of Safa and Marwah, located near the Kaaba.

The place is entirely enclosed by the mosque and can be accessed via air-conditioned tunnels. Figure 8.4 shows the plan of the mosque (different levels) and zones in which the ritual is performed. The size of the boxes represents the density in each group (the number of individuals per square meter). In our simulation, this is represented by the height of the boxes. Note that in our DOE a high density represents the situation in which six individuals share one square meter. Depending on the phase of the ritual the groups are performing, some zones of the mosque are more crowded than others.

As shown in Figure 8.4 it was possible to simulate the groups participating in the Hajj in a realistic manner concerning: (i) the number of individuals in the crowd, (ii) the distribution of their anthropometric (coded in the colour of the box according to the colour code explained above) and sociocultural characteristics used to compute density which is visible in the size of the boxes in 8.4. Note that depending on the group they belong to, pilgrims pray in different areas of the mosque. For example, Arabs in cyan are grouped in the second level; Iranians in the first and third levels; and Indonesians, Malaysians, and Africans mostly share the fourth level.

Then, it was important to model the dynamic aspect of the pilgrimage based on the predefined ritual performed through the spaces and tunnels of the mosque. In our current DOE, it was simulated a simplified version of the itinerary for the time being. The simulation exhibits the "risky" places (e.g., tight tunnels and entrances). Figure 8.5 shows the simulation results of the crowd trying to cross the different zones of the mosque. It shows results of the first level which is occupied by Iranians (green points) and Indonesians (blue points). In the top of Figure 8.5 Iranians are in a kind of initial state and Indonesians are already moving in the surrounding tunnel and occupying the main "patio."

In a second moment, in figure 8.5, it is possible to observe the Iranian crowd (green points) starts to organize itself to gain one of the two exits and reach the tunnel. According to the direction they can only use the right-side exit. The middle and bottom of the Figure 8.5 sequence show how they organize to reach and go through the exit. First, they go to the center of the space then they gather themselves together at some point of the room while others approach the exit and start walking through the tunnel. At the same time, Indonesians (blue points) start moving towards the only exit in the right side of the square. Some gather themselves in the inner tunnel around the "patio" and the others in start going through the main tunnel.

Using this kind of sequences in the simulation helps to understand the collective behavior of different groups. There is particular interest to observe the behavior when they faced "delicate" facilities, for example, tight entrances, or halls, tunnels or other essential points where people tend to stop and the flow stalls. These critical aspects can help to determine the situations and possible logistics problems to face in emergencies and mainly avoid stampedes and crushes in a group that is larger than a million Muslim pilgrims.

Figure 8.6 shows the result of the simulation performed to measure the speed in which pilgrims could run across the tunnels of the mosque in the last part of the itinerary. To give a simple example, Figure 8.6 illustrates the case of Indonesian pilgrims. The objective of this simulation was to identify places where there could be potential crushes or stampedes.

Note that the complex urban distribution of the mosque and the itinerary encouraging people to walk at different paces, and looking for spaces to accomplish all the phases of the ritual, creates potentially risky zones. People doing distinct stages of the ceremony might find themselves going in opposite directions in tight tunnels. Other might want to access specific points without considering the density of the crowd and cause problems. Not respecting (or understanding) instructions, directions and signs can be fatal in such a complex layout. Helbing has emitted recommendations with general urban rules based on the behavior models they came up with. Still, simulations in 3D environments that model both the behavior of each

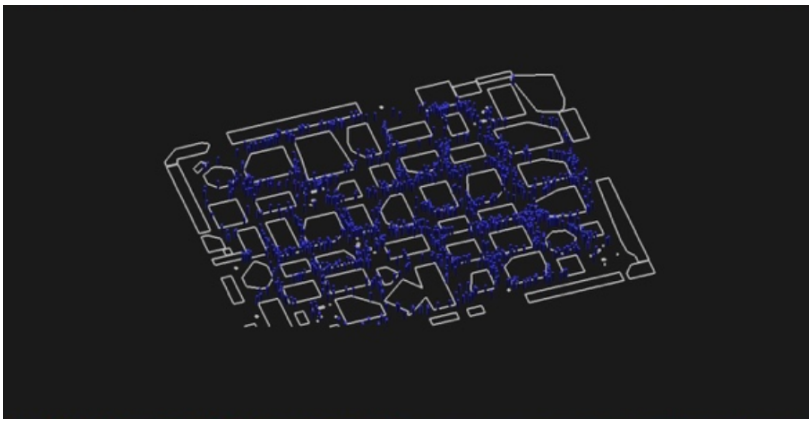


FIGURE 8.6: Detailed view of the distribution of pilgrims in the mosque during the Hajj.

agent including her sociocultural and anthropomorphic features and the dynamic aspect to the behavior are open issues.

8.4 Summary

This chapter uses an agent-based approach for modeling and simulating dynamic human crowds. It includes the use of specific anthropometric and sociocultural aspects that have an impact on individual and group behavior. Different to existing models that focus on the flow dynamics of individuals in particular spaces and therefore provide a partial abstraction of individuals, our approach includes anthropometric and sociocultural aspects because: (i) it aims to address human logistics in which modeling both the urban space and the individuals of the crowds is essential to predict and prevent catastrophes; (ii) it uses virtual worlds to visualize the simulations of dynamic crowds in Public Spaces. The challenge is to generate realistic reactions of the individuals interacting among each other and evolving in the space, but also of the social groups identified within the dynamic crowd and the dynamic crowd considered as an indivisible entity.

The relevance of this research is that it derives a behavioral model of the dynamics of the crowds from empirical observation so this simulation will help to prevent crisis scenarios. Using and supporting simulations and visualization by efficient and ad-hoc data management makes it possible to address real-time dynamic crowd observation and organization.

Decision-makers can consider the state of the dynamic crowd, the way it occupies space and the way individuals form or leave groups. With new technology, it might be possible to configure and reconfigure urban areas dynamically to prevent disasters and to help people to behave safely in collective events. These issues are part of our future work.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

9.1.1 Research Questions Answered

At the beginning of the project, the possibility to implement large scale crowd simulation and visualization on heterogeneous computer architectures was raised. Through applying the right methodology in terms of parallel algorithms development and using innovative parallel programming models such as OmpSs and CUDA and MPI, the author provides efficient implementations that proved the efficiency of his approach and ultimately combined the validity of his hypothesis. In this sense the following objectives were achieved:

- Within the project was developed a new approach to scale crowd simulation and visualization on heterogeneous computing cluster using a task-based technique.
- It abstracts the difficulties that imply the use of heterogeneous architectures like memory management, scheduling, communications, and synchronization — facilitating development, maintenance, and scalability.
- It is capable of scaling on multiple nodes
- It makes efficient use of resources by implementing up to four levels of parallelism

Another issue to explore was the use of real data to develop new models of behavior or improve current ones, also, to verify if these data are useful to give realism to the scenes. Within the project, a case study is presented for a crowd simulation in urban scenarios. Geolocalized data produced by mobile devices are used to predict individual and public behavior and detect abnormal situations in the presence of specific events. The challenge of combining the location of all these individuals with a 3D representation of the urban environment is also addressed. In this sense, the following objectives were achieved:

- New behavior models were developed based on data analysis.
- it was created the infrastructure to be able to consult various data sources such as social networks, government agencies or transport companies such as Uber.
- Realistic urban scenes were developed based on crowdsourced data.

9.1.2 General Conclusions

Large-scale crowd simulation and visualization are essential topics in areas as different as urbanism, training, entertainment among others. This kind of systems requires a vast computational power and memory resources commonly available in High Performance Computing HPC platforms.

Heterogeneous clusters provide various levels of parallelism that need to be exploited successfully to harness their computational power. At node-level, Pthreads, OpenMP directives, among other libraries, provide access to CPU level parallelism and OpenACC, CUDA or OpenCL libraries do the same for accelerators. Once node-level parallelism is achieved, additional scalability is accomplished by allocating computation in several nodes, using the message passing interface (MPI).

Programmers need to combine these programming models in their code and fully optimize it for each level of parallelism which turns into a complex task. These two processes also may suggest programming heterogeneous systems is an ad hoc activity. To exploit to the maximum the capacity of the current HPC infrastructure and be prepared for the exascale era. The project develops a new approach to scale crowd simulation and visualization on heterogeneous computing cluster using a task-based technique.

Its main characteristic is hardware agnostic. OmpSs abstracts the difficulties that the use of accelerators implies, making flexible and efficient use of resources — facilitating the development of the application. In this way, researchers from different areas can use this kind of infrastructure without being experts in computing science. The author shows its implementation in one node and how it scales to a heterogeneous computing cluster.

Visualization transforms results in graphical and color representations can improve human cognition in data or results' analysis. The selection of a visualization mechanism depends on the simulation and available HPC infrastructure.

With the goal of flexibility and take advantage of computing resources as best as possible, three different configurations to connect the simulation with the visualization engine were designed: in-situ, streaming, and web. An analysis of the trade-offs of each one was presented. The In-situ configuration allows performing large-scale simulations. Besides, it offers a testbed to analyze further scalability behavior of interactive simulations that use the GPU for computing and rendering.

Streaming and Web configuration offer flexibility and may reduce infrastructure costs when advanced visualization systems such as the CAVE or Tiled Display Walls are remotely available. The streaming configuration also allows to couple with different simulation engines. The Web configuration can be used to take advantage of sensors available on mobile devices to improve user interaction.

Path planning for large-scale crowds is a computationally expensive process,

due to the dynamism in the scenes and vast search space. Our hierarchical path-finding approach offers significant advantages: it can return a partial path instead of wait for a complete one which allows a character to start moving while computing the rest asynchronously. If part of the route changes which is common in dynamic scenes or video games in which the player is chasing an object or character in motion. It can reprocess only a part of the route with different levels of abstraction.

It has a low memory consumption, it is domain-independent, and it is suitable to improve applying parallelism in a task-based approach. It was presented an analysis of the algorithm in terms of pre-computation time, use of memory, speed-up, and optimality. Our results show the algorithm speed-up near to 6x in comparison with the typical A* approach.

A case study for a crowd simulation in urban scenarios was presented. Geolocated data produced by mobile devices and other sources of information (e.g., security cameras, drones) to predict individual and crowd behavior and detect abnormal situations in the presence of specific events were used. Also, it was addressed the challenge of combining all these individual's location with a 3D rendering of the urban environment. Our data processing and simulation approach are computationally expensive and time-critical; the project relies thus on a hybrid Cloud-HPC architecture to produce an efficient solution.

During the development of our model of crowds in urban spaces, it was developed the infrastructure to be able to consult different data sources such as social networks, government agencies or transport companies such as Uber. Every time there is more data available and better computation resources which allow performing analysis of greater depth, this lays the foundations to improve the current simulation models.

This kind of systems gives decision-makers the opportunity to take measures in real time to avoid catastrophes or take preventive measures identifying which is the public infrastructure that should be improved.

Through our case study in chapter 8, it is possible to observe how anthropometric and sociocultural aspects have an impact on individual and group behavior. Address human logistics in which modeling both the urban space and the individuals of the crowds is essential to predict and prevent catastrophes.

Using and supporting simulations and visualization makes it possible to address real-time dynamic crowd observation and organization. Decision makers can consider the state of the crowd, the way it occupies space and the way individuals form or leave groups. With new technology, it might be possible to configure and reconfigure urban areas dynamically to prevent disasters and to help people to behave safely in public events.

9.2 Future Work

The author plan to add task-based in-situ analysis and visualization capabilities to the system, i.e., it is expected to take advantage of OmpSs' automatic scheduling, memory management, communication and synchronization for crowd visualization and analysis.

A task-based approach may have advantages over current parallel rendering techniques. For example, visualization can be easily coupled or decoupled from the simulation just by adding or removing rendering tasks. This also applies to crowd analysis. It will help in decomposing communication between the simulation, visualization, and analysis into tasks while their execution may take advantage of OmpSs' synchronization and automatic scheduling. Also, it will allow us to deploy the visualization system into immersive or large-scale tiled displays connected to GPU clusters. Finally, OmpSs can be seen as a bridge connecting data from simulation to visualization and analysis while maintaining the rendering algorithms intact.

This interaction is expected to be similar to the OmpSs-CUDA programs, where OmpSs is aware of the CUDA's context, memory, and GPU availability. Nevertheless, the full interaction between OmpSs-OpenGL is not supported yet. Additional challenges such as OpenGL thread safety, OpenGL context creation and registration in OmpSs, OmpSs multi GPU compatibility for graphics or updating of the OmpSs scheduling system to support graphics load balancing have to be solved first.

For our hierarchical path-finding algorithm it is used a recursive function to look for the best path in a 4x4 grid. For applications in crowd simulations or video games, it will be computed the path asynchronously and in parallel with a task-based approach. Further work includes the implementation of diagonal movement to improve the solution and compare with other algorithms. For applications in embedded systems, it will be implemented in hardware the recursive function, and this will accelerate the process by orders of magnitude.

Appendix A

Publications

Journals

- Benjamin Hernandez, Hugo Perez, Isaac Rudomin, Sergio Ruiz, Oriam DeGyves, Leonel Toledo. Simulating and Visualizing Real-Time Crowds on GPU Clusters. Computación y Sistemas, 2014.

- Hugo Pérez, Benjamín Hernández, Isaac Rudomin, Eduard Ayguade. Scaling Crowd Simulations in a GPU Accelerated Cluster. High Performance Computer Applications: 6th International Conference, Mexico City, Mexico, Revised Selected Papers. Springer 2016.

- Israel Tabarez-Paz, Isaac Rudomin, Hugo Perez. Support Vector Machine and Spiking Neural Networks for Data Driven Prediction of Crowd Character Movement. Proceedings of the Artificial Life Conference. The MIT Press 2016.

- Carlos Alberto Ochoa Zezzatti, Isaac Rudomin, Genoveva Vargas Solar, Javier A. Espinosa-Oviedo, Hugo Pérez, José-Luis Zechinelli Martini. Humanitarian Logistics and Cultural Diversity within Crowd Simulation. Computacion y Sistemas, 2017.

- Rudomín, Isaac; Vargas-Solar, Genoveva; Espinosa-Oviedo, Javier A.; Pérez, Hugo; Zechinelli-Martini, José-Luis; Modelling Crowds in Urban Spaces. Computación y Sistemas 2017.

Book Chapters

- Hugo Perez, Benjamin Hernandez, Isaac Rudomin, Eduard Ayguade. Task-based Crowd Simulation for Heterogeneous Architectures. Q. F. Hassan (Ed.). Innovative Research and Applications in Next-Generation High Performance Computing. IGI Global 2016.

Conferences

- Hugo Perez, Benjamin Hernandez, Isaac Rudomin. In Situ Crowd Simulation and Visualization. International Supercomputing Conference in Mexico ISUM. Mexico 2014.

- Isaac Rudomin, Benjamin Hernandez, Hugo Perez. Generation, Simulation and Rendering of Large Varied Animated Crowds. International Supercomputing Conference in Mexico ISUM. Mexico 2014.

- Hugo Perez, Benjamin Hernandez, Isaac Rudomin. Task-based Simulation and Visualization of Crowds for Heterogeneous Cluster. International Supercomputing Conference in Mexico ISUM. Mexico 2015.

- Hugo Perez, Benjamin Hernandez, Isaac Rudomin. OmpSuperscalar: Task-Parallel Simulation and Visualization of Crowds with Several CPUs and GPUs. GPU Technology Conference GTC. USA 2015.

- Genoveva Vargas-Solar, Javier A. Espinosa-Oviedo, Hugo Perez, Isaac Rudomín, José Luis Zechinelli-Martini. Modelling Crowds in Urban Spaces: From Big Data to Smart Secure Regions. International Supercomputing Conference in Mexico ISUM. Mexico 2016.

- Hugo Perez, Isaac Rudomin, Eduard Ayguade. In Situ Crowd Simulation and Visualization BSC International Doctoral Symposium. Spain 2016

- Hugo Perez, Isaac Rudomin, Eduard Ayguade. Crowd Simulation and Visualization BSC International Doctoral Symposium. Spain 2017

- Isaac Rudomin, Hugo Perez, Leonel Toledo, Jorge Hernandez. Urban Scale Crowd Data Analysis, Simulation and Visualization GPU Technology Conference – NVIDIA. USA 2017

Posters

- Israel Tabarez-Paz, Isaac Rudomin, Hugo Perez. Support Vector Machine and Spiking Neural Networks for Data Driven Prediction of Crowd Character Movement. Proceedings of the Artificial Life Conference. MIT Press 2016.

- Alberto Ochoa-Zezzati, Isaac Rudomín, Hugo Perez, Javier A. Espinosa-Oviedo, Genoveva Vargas-Solar. Simulation of Human Stampedes for Improving Decision Takin in Humanitarian Logistics. Jornadas de Cooperación Conacyt-Catalunya. Spain 2016.

- Hugo Perez, Isaac Rudomin, Eduard Ayguade. Crowd Simulation and Visualization. Advanced Computer Architecture and Compilation for High-Perfomance and Embedded Systems. Italy 2016.

Bibliography

[1] H. Ritchie and M. Roser, "Urbanization", *Our World in Data*, 2019. [Online]. Available: https://ourworldindata.org/urbanization.

[2] N. Pelechano, J. M. Allbeck, and N. I. Badler, *Virtual crowds : Methods, simulation, and control*, English. [San Rafael, Calif.] : Morgan Claypool Publishers, 2008, Cover title, ISBN: 1598296418 (pbk.)

[3] W. Li, M. Jiang, Y. Chen, and M. C. Lin, "Estimating urban traffic states using iterative refinement and wardrop equilibria", *IET Intelligent Transport Systems*, 2018.

[4] D. Wolinski, "Crowd simulation: Realism assessment of interaction models", PhD thesis, INRIA-IRISA Rennes Bretagne Atlantique, équipe Mimétic, 2012.

[5] *Top500*, http://top500.org, 2015.

[6] K. Moreland, "Oh, $#*@! exascale! the effect of emerging architectures on scientific discovery", in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, IEEE, 2012, pp. 224–231.

[7] D. Thalmann, H. Grillon, J. Maim, and B. Yersin, "Challenges in crowd simulation", in *2009 International Conference on CyberWorlds*, 2009, pp. 1–12. DOI: 10.1109/CW.2009.23.

[8] W. Gropp and M. Snir, "Programming for exascale computers", *Computing in Science and Engg.*, vol. 15, no. 6, pp. 27–35, Nov. 2013, ISSN: 1521-9615. DOI: 10.1109/MCSE.2013.96. [Online]. Available: https://doi.org/10.1109/MCSE.2013.96.

[9] A. Kageyama and T. Yamada, "An approach to exascale visualization: Interactive viewing of in-situ visualization", *Computer Physics Communications*, vol. 185, no. 1, pp. 79 –85, 2014, ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2013.08.017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010465513002804.

[10] J. P. Ahrens, "Visualization and data analysis at the exascale", *Technical Report LLNL-TR-474731*, Jan. 2011. DOI: 10.2172/1011053.

[11] S. Singh, M. Kapadia, P. Faloutsos, and G. Reinman, "An open framework for developing, evaluating, and sharing steering algorithms", in *Motion in Games*, Springer, 2009, pp. 158–169.

[12] *Open Paths*, https://openpaths.cc, 2017.

[13] *FOILing NYC Taxi Trip Data*, http://chriswhong.com/open-data, 2017.

[14] *Uber Movement*, https://movement.uber.com, 2017.

[15] H. Perez, B. Hernandez, I. Rudomin, and E. Ayguade, "Task-based crowd simulation for heterogeneous architectures", in *Innovative Research and Applications in Next-Generation High Performance Computing*, Q. F. Hassan, Ed., 1st ed., IGI Global, Jul. 2016, ch. 8, ISBN: 9781522502876.

[16] H. Pérez, B. Hernández, I. Rudomín, and E. Ayguadé, "Scaling crowd simulations in a gpu accelerated cluster", in *High Performance Computer Applications: 6th International Conference, ISUM 2015, Mexico City, Mexico*, I. Gitler and J. Klapp, Eds. Cham: Springer International Publishing, 2016, pp. 461–472, ISBN: 978-3-319-32243-8. DOI: 10.1007/978-3-319-32243-8_32. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-32243-8_32.

[17] B. Hernandez, H. Perez, I. Rudomin, S. Ruiz, O. DeGyves, and L. Toledo, "Simulating and visualizing real-time crowds on gpu clusters", *Computación y Sistemas*, vol. 18, no. 4, 2014.

[18] G. Vargas-Solar, J. Espinosa-Oviedo, H. Perez, I. Rudomín, and J. L. Zechinelli-Martini, "Modelling crowds in urban spaces: From big data to smart secure regions", in *International Supercomputing Conference in Mexico ISUM. Mexico 2016.*, 2016.

[19] A. A. Ochoa, I. Rudomin, G. Vargas-Solar, J. A. Espinosa-Oviedo, H. Pérez, and J.-L. A. Zechinelli-Martini, "Humanitarian logistics and cultural diversity within crowd simulation", *COMPUTACION Y SISTEMAS*, vol. 21, no. 1, pp. 7–21, 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01517161.

[20] I. Rudomin, E. Millán, and B. Hernńdez, "Fragment shaders for agent animation using finite state machines", *Simulation Modelling Practice and Theory*, vol. 13, no. 8, pp. 741 –751, 2005, Programmable Graphics Hardware, ISSN: 1569-190X. DOI: http://dx.doi.org/10.1016/j.simpat.2005.08.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X05000894.

[21] P. Richmond, S. Coakley, and D. M. Romano, "A high performance agent based modelling framework on graphics card hardware with cuda", in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, ser. AAMAS '09, Budapest, Hungary: International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 1125–1126, ISBN: 978-0-9817381-7-8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1558109.1558172.

[22] 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, ISSN: 1460-7425. [Online]. Available: http://jasss.soc.surrey.ac.uk/11/4/10.html.

[23] D. G.O.T.L.R.I.R. S. Isaac Rudomin Benjamín Hernández, "Gpu generation of large varied animated crowds", *Revista Computación y Sistemas*, vol. 17, no. 3, 2013.

[24] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas, "Ompss: A proposal for programming heterogeneous multi-core architectures", *Parallel Processing Letters*, vol. 21, no. 02, pp. 173–193, 2011. DOI: 10.1142/S0129626411000151. eprint: https://doi.org/10.1142/S0129626411000151. [Online]. Available: https://doi.org/10.1142/S0129626411000151.

[25] J. Balart, A. Duran, M. Gonzàlez, X. Martorell, E. Ayguadé, and J. Labarta, "Nanos mercurium: A research compiler for openmp", in *European Workshop on OpenMP (EWOMP'04). Pp*, 2004, pp. 103–109.

[26] *Nanos++*, https://pm.bsc.es/projects/nanox, 2015.

[27] G. Ozen, E. Ayguadé, and J. Labarta, "On the roles of the programmer, the compiler and the runtime system when programming accelerators in openmp", in *Using and Improving OpenMP for Devices, Tasks, and More*, L. DeRose, B. R. de Supinski, S. L. Olivier, B. M. Chapman, and M. S. Müller, Eds., Cham: Springer International Publishing, 2014, pp. 215–229, ISBN: 978-3-319-11454-5.

[28] B. D. Hankin and R. A. Wright, "Passenger flow in subways", *OR*, vol. 9, no. 2, pp. 81–88, 1958, ISSN: 14732858. [Online]. Available: http://www.jstor.org/stable/3006732.

[29] S. J. Older, "Movement of pedestrians on footways in shopping streets", *Traffic Engineering and Control*, vol. 10, pp. 160–163, 1968.

[30] F. J. J., "Designing for pedestrians: A level-of-service concept", *Transportation Research Board Highways Research Record*, pp. 1–15, 1971.

[31] J. Berclaz, F. Fleuret, and P. Fua, "Robust people tracking with global trajectory optimization", in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1, 2006, pp. 744–750. DOI: 10.1109/CVPR.2006.258.

[32] A. Borgers and H. J. P. Timmermans, "City centre entry points, store location patterns and pedestrian route choice behaviour: A microlevel simulation model", *Socio-Economic Planning Sciences*, vol. 20, no. 1, pp. 25–31, 1986. [Online]. Available: https://EconPapers.repec.org/RePEc:eee:soceps:v:20:y:1986:i:1:p:25-31.

[33] V. Blue and J. Adler, "Emergent fundamental pedestrian flows from cellular automata microsimulation", *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1644, pp. 29–36, 1998. DOI: 10.3141/1644-04.

[34] "Highway capacity manual", in *Special Report, 209*. Transportation Research Board, 1994.

[35] J. P. Keating, "The myth of panic", *Fire journal*, vol. 76, no. 3, pp. 57–61, 1982.

[36] D. Elliott and D. Smith, "Football stadia disasters in the united kingdom: Learning from tragedy?", *Industrial & Environmental Crisis Quarterly*, vol. 7, no. 3, pp. 205–229, 1993. DOI: 10.1177/108602669300700304.

[37] D. J. Parker and J. Handmer, "Disaster at hillsborough stadium: A comparative analysis", in *Hazard management and emergency planning: Perspectives on Britain*. Wiley Online Library, 1992, ch. 10.

[38] D. Caner, "Human behaviour in fires", *Fire and Materials*, vol. 18, 267–268, 1994.

[39] T. Zhao and R. Nevatia, "Bayesian human segmentation in crowded situations", in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 2, 2003, pp. II–459. DOI: 10.1109/CVPR.2003.1211503.

[40] M. Rodriguez, S. Ali, and T. Kanade, "Tracking in unstructured crowded scenes", *2009 IEEE 12th International Conference on Computer Vision*, pp. 1389–1396, 2009.

[41] G. J. Brostow and R. Cipolla, "Unsupervised bayesian detection of independent motion in crowds", in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1, 2006, pp. 594–601. DOI: 10.1109/CVPR.2006.320.

[42] S. Ali and M. Shah, "A lagrangian particle dynamics approach for crowd flow segmentation and stability analysis", in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–6. DOI: 10.1109/CVPR.2007.382977.

[43] B. N, H. H, and W. M, "Tracking individuals in surveillance video of a high-density crowd", vol. 8399, 2012, pp. 8399 –8399 –8. DOI: 10.1117/12.979489. [Online]. Available: https://doi.org/10.1117/12.979489.

[44] H. Jiang, S. Fels, and J. J. Little, "A linear programming approach for multiple object tracking", in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383180.

[45] C. Huang, B. Wu, and R. Nevatia, "Robust object tracking by hierarchical association of detection responses", in *Computer Vision – ECCV 2008*, D. Forsyth, P. Torr, and A. Zisserman, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 788–801.

[46] B. Leibe, K. Schindler, and L. V. Gool, "Coupled detection and trajectory estimation for multi-object tracking", in *2007 IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–8. DOI: 10.1109/ICCV.2007.4408936.

[47] L. Zhang, Y. Li, and R. Nevatia, "Global data association for multi-object tracking using network flows", in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587584.

[48] T. Zhao and R. Nevatia, "Tracking multiple humans in crowded environment", in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 2, 2004, pp. II–II. DOI: 10.1109/CVPR.2004.1315192.

[49] B. Wu and R. Nevatia, "Detection and segmentation of multiple, partially occluded objects by grouping, merging, assigning part detection responses", *International Journal of Computer Vision*, vol. 82, no. 2, pp. 185–204, 2009, ISSN: 1573-1405. DOI: 10.1007/s11263-008-0194-9. [Online]. Available: https://doi.org/10.1007/s11263-008-0194-9.

[50] Y. Li, C. Huang, and R. Nevatia, "Learning to associate: Hybridboosted multi-target tracker for crowded scene", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 2953–2960. DOI: 10.1109/CVPR.2009.5206735.

[51] D. Sugimura, K. M. Kitani, T. Okabe, Y. Sato, and A. Sugimoto, "Using individuality to track individuals: Clustering individual trajectories in crowds using local appearance and frequency trait", in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 1467–1474. DOI: 10.1109/ICCV.2009.5459286.

[52] H. Tsutsui, J. Miura, and Y. Shirai, "Optical flow-based person tracking by multiple cameras", in *Conference Documentation International Conference on Multisensor Fusion and Integration for Intelligent Systems. MFI 2001 (Cat. No.01TH8590)*, 2001, pp. 91–96. DOI: 10.1109/MFI.2001.1013514.

[53] T. Yamane, Y. Shirai, and J. Miura, "Person tracking by integrating optical flow and uniform brightness regions", in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, 1998, 3267–3272 vol.4. DOI: 10.1109/ROBOT.1998.680942.

[54] L. Kratz and K. Nishino, "Tracking with local spatio-temporal motion patterns in extremely crowded scenes", in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 693–700. DOI: 10.1109/CVPR.2010.5540149.

[55] L. Henderson, "On the fluid mechanics of human crowd motion", *Transportation Res.*, vol. 8, pp. 509–515, Dec. 1974. DOI: 10.1016/0041-1647(74)90027-6.

[56] R. L. Hughes, "A continuum theory for the flow of pedestrians", *Transportation Research Part B: Methodological*, vol. 36, pp. 507–535, Jul. 2002. DOI: 10.1016/S0191-2615(01)00015-7.

[57] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics", *Physical review E*, vol. 51, no. 5, p. 4282, 1995.

[58] D. Helbing, "A mathematical model for the behavior of pedestrians", *Behavioral Science*, vol. 36, no. 4, pp. 298–310, 1991. DOI: 10.1002/bs.3830360405. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/bs.3830360405. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/bs.3830360405.

[59] P. Gipps and B. Marksjö, "A micro-simulation model for pedestrian flows", *Mathematics and Computers in Simulation (MATCOM)*, vol. 27, no. 2, pp. 95–105, 1985. DOI: 10.1016/0378-4754(85)9003. [Online]. Available: https://ideas.repec.org/a/eee/matcom/v27y1985i2p95-105.html.

[60] 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, pp. 507–525, Jun. 2001. DOI: 10.1016/S0378-4371(01)00141-8.

[61] S. Gopal and T. Smith, "Human way-finding in an urban environment: A performance analysis of a computational process model", *Environment and Planning A*, vol. 22, pp. 169–191, Jan. 1990. DOI: 10.1068/a220169.

[62] C. W. Reynolds, "Evolution of corridor following behavior in a noisy world", 1994.

[63] C. Reynolds, "Flocks, herds and schools: A distributed behavioral model", *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 25–34, Aug. 1987, ISSN: 0097-8930. DOI: 10.1145/37402.37406. [Online]. Available: http://doi.acm.org/10.1145/37402.37406.

[64] S. Paris, J. Pettré, and S. Donikian, "Pedestrian reactive navigation for crowd simulation: A predictive approach", in *Computer Graphics Forum*, Wiley Online Library, vol. 26, 2007, pp. 665–674.

[65] J. van den Berg, S. Patil, J. Sewall, D. Manocha, and M. Lin, "Interactive navigation of multiple agents in crowded environments", in *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, 2008, pp. 139–147.

[66] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance", in *Robotics research*, Springer, 2011, pp. 3–19.

[67] 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, 123:1–123:9, Jul. 2010, ISSN: 0730-0301. DOI: 10.1145/1778765.1778860. [Online]. Available: http://doi.acm.org/10.1145/1778765.1778860.

[68] 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, ISSN: 0027-8424. DOI: 10.1073/pnas.1016507108. eprint: https://www.pnas.org/content/108/17/6884.full.pdf. [Online]. Available: https://www.pnas.org/content/108/17/6884.

[69] D. Helbing, P. Molnár, I. J. Farkas, and K. Bolay, "Self-organizing pedestrian movement", *Environment and Planning B: Planning and Design*, vol. 28, no. 3, pp. 361–383, 2001. DOI: 10.1068/b2697.

[70] N E. Miller, "Experimental studies of conflict", *Personality and the Behavioural Disorders*, vol. 1, Jan. 1944.

[71] K. Lewin, "Field theory of social science: Selected theoretical papers", *The ANNALS of the American Academy of Political and Social Science*, vol. 276, no. 1, pp. 146–147, 1951. DOI: 10.1177/000271625127600135. eprint: https://doi.org/10.1177/000271625127600135. [Online]. Available: https://doi.org/10.1177/000271625127600135.

[72] Yang, Gonzalez-Banos, and Guibas, "Counting people in crowds with a real-time network of simple image sensors", in *Proceedings Ninth IEEE International Conference on Computer Vision*, 2003, 122–129 vol.1. DOI: 10.1109/ICCV.2003.1238325.

[73] G. K. Still, *Crowd dynamics*, 2000.

[74] K. M. Thompson, *Scale, spectacle and movement: Massive software and digital special effects in the lord of the rings.* 2006.

[75] *Golaem crowd*, http://www.golaem.com, 2015.

[76] K. Nagel and M. Rickert, "Parallel implementation of the transims microsimulation", *Parallel Computing*, vol. 27, no. 12, pp. 1611 –1639, 2001, Applications of parallel computing in transportation, ISSN: 0167-8191. DOI: https://doi.org/10.1016/S0167-8191(01)00106-5.

[77] M. Kiran, L. S. Chin, P. Richmond, M. Holcombe, D. Worth, and C. Greenough, *Flame: Simulating large populations of agents on parallel hardware architectures*.

[78] X. Rubio-campillo, "Pandora: A versatile agent-based modelling platform for social simulation", in *Proceedings of SIMUL 2014, The Sixth International Conference on Advances in System Simulation*, 2014.

[79] T. Yu, M. Dou, and M. Zhu, "A data parallel approach to modelling and simulation of large crowd", *Cluster Computing*, vol. 18, no. 3, pp. 1307–1316, 2015, ISSN: 1573-7543. DOI: 10.1007/s10586-015-0451-y. [Online]. Available: https://doi.org/10.1007/s10586-015-0451-y.

[80] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems", *Proceedings of the National Academy of Sciences*, vol. 99, no. suppl 3, pp. 7280–7287, 2002, ISSN: 0027-8424. DOI: 10.1073/pnas.082080899. eprint: https://www.pnas.org/content/99/suppl_3/7280.full.pdf. [Online]. Available: https://www.pnas.org/content/99/suppl_3/7280.

[81] N. Pelechano, J. M. Allbeck, M. Kapadia, and N. I. Badler, *Simulating heterogeneous crowds with interactive behaviors*. Natick, MA, USA: A. K. Peters, Ltd., 2016, ISBN: 1498730361, 9781498730365.

[82] *Extrae*, http://www.bsc.es/computer-sciences/extrae, 2015.

[83] *Paraver*, http://www.bsc.es/computer-sciences/performance-tools/paraver, 2015.

[84] NVIDIA, "Summit and Sierra Supercomputers: An Inside Look at the U.S. Department of Energy's New Pre-Exascale Systems ", NVIDIA, Tech. Rep., Nov. 2014.

[85] K. Ma, "In situ visualization at extreme scale: Challenges and opportunities", *IEEE Computer Graphics and Applications*, vol. 29, pp. 14–19, Nov. 2009, ISSN: 0272-1716. DOI: 10.1109/MCG.2009.120.

[86] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K. Ma, "In situ visualization for large-scale combustion simulations", *IEEE Computer Graphics and Applications*, vol. 30, no. 3, pp. 45–57, 2010, ISSN: 0272-1716. DOI: 10.1109/MCG.2010.55.

[87] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs, "A sorting classification of parallel rendering", *IEEE Comput. Graph. Appl.*, vol. 14, no. 4, pp. 23–32, Jul. 1994, ISSN: 0272-1716. DOI: 10.1109/38.291528. [Online]. Available: https://doi.org/10.1109/38.291528.

[88] C. Foudil, N. Djedi, C. Sanza, and Y. Duthen, "Path finding and collision avoidance in crowd simulation", *CIT*, vol. 17, pp. 217–228, Jan. 2009. DOI: 10.2498/cit.1000873.

[89] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968, ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

[90] E. W. Dijkstra, "A note on two problems in connexion with graphs", *Numer. Math. 1*, pp. 269–271, 1959.

[91] S. Even, *Graph algorithms*, 2nd. New York, NY, USA: Cambridge University Press, 2011, ISBN: 0521736536, 9780521736534.

[92] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search", *Artif. Intell.*, vol. 27, pp. 97–109, 1985.

[93] N. R. Sturtevant, J. M. Traish, J. R. Tulip, T. Uras, S. Koenig, B. Strasser, A. Botea, D. Harabor, and S. Rabin, "The grid-based path planning competition: 2014 entries and results", in *SOCS*, 2015.

[94] D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps", in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, ser. AAAI'11, San Francisco, California: AAAI Press, 2011, pp. 1114–1119. [Online]. Available: http://dl.acm.org/citation.cfm?id=2900423.2900600.

[95] D. Harabor and Grastien, "Improving jump point search", in *International Conference on Automated Planning and Scheduling*, 2014.

[96] T. Uras and S. Koenig, "Identifying hierarchies for fast optimal search", 2014.

[97] B. Strasser, D. Harabor, and A. Botea, "Fast first-move queries through run-length encoding", in *SOCS*, 2014.

[98] A. Botea, M. Müller, and J. Schaeffer, "Near optimal hierarchical pathfinding", *Journal of Game Development*, vol. 1, pp. 7–28, 2004.

[99] N. Sturtevant and M. Buro, "Partial pathfinding using map abstraction and refinement", in *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*, ser. AAAI'05, Pittsburgh, Pennsylvania: AAAI Press, 2005, pp. 1392–1397, ISBN: 1-57735-236-x. [Online]. Available: http://dl.acm.org/citation.cfm?id=1619499.1619557.

[100] A. Kring, A. J. Champandard, and N. Samarin, "Dhpa* and shpa*: Efficient hierarchical pathfinding in dynamic and static game worlds", in *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE'10, Stanford, California, USA: AAAI Press, 2010, pp. 39–44. [Online]. Available: http://dl.acm.org/citation.cfm?id=3014666.3014675.

[101] G. Karypis and V. Kumar, "Multilevelk-way partitioning scheme for irregular graphs", *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 96 –129, 1998, ISSN: 0743-7315. DOI: https://doi.org/10.1006/jpdc.1997.1404. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731597914040.

[102] H. Samet, "An overview of quadtrees, octrees, and related hierarchical data structures", in *Theoretical Foundations of Computer Graphics and CAD*, R. A. Earnshaw, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 51–68, ISBN: 978-3-642-83539-1.

[103] J. Hirt, D. Gauggel, J. Hensler, M. Blaich, and O. Bittel, "Using quadtrees for realtime pathfinding in indoor environments", May 2010. DOI: 10.1007/978-3-642-27272-1_6.

[104] N. Sturtevant, "Benchmarks for grid-based pathfinding", *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144 – 148, 2012. [Online]. Available: http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf.

[105] J. Byrne, S. Caulfield, L. Buckley, X. Xu, D. Pena, G. Baugh, and D. Moloney, "Applications of the vola format for 3d data knowledge discovery", *International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery. China*, vol. 13, 2017.

[106] *HOG Path-Finding Library*, https://webdocs.cs.ualberta.ca/ nathanst/hog.html, 2019.

[107] G. Rong, Y. Liu, W. Wang, X. Yin, D. Gu, and X. Guo, "Gpu-assisted computation of centroidal voronoi tessellation", *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 3, pp. 345–356, 2011, ISSN: 1077-2626. DOI: 10.1109/TVCG.2010.53.

[108] O. De Gyves, L. Toledo, and I. Rudomín, "Comportamientos en simulación de multitudes: Revisión del estado del arte", in *Research in Computer Science Special Issue: Avances en Inteligencia Artificial*, vol. 62, 2013.

[109] A. Bhattacharya and S. K. Das, "Lezi-update: An information-theoretic approach to track mobile users in pcs networks", in *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, ser. MobiCom '99, Seattle, Washington, USA: ACM, 1999, pp. 1–12, ISBN: 1-58113-142-9. DOI: 10.1145/313451.313457. [Online]. Available: http://doi.acm.org/10.1145/313451.313457.

[110] J. Wolf, R. Guensler, and W. Bachman, "Elimination of the travel diary: Experiment to derive trip purpose from global positioning system travel data", *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1768, pp. 125–134, 2001. DOI: 10.3141/1768-15. eprint: https://doi.org/10.3141/1768-15. [Online]. Available: https://doi.org/10.3141/1768-15.

[111] N. Marmasse and C. Schmandt, "Location-aware information delivery with commotion", in *Proceedings of the 2Nd International Symposium on Handheld and Ubiquitous Computing*, ser. HUC '00, Bristol, UK: Springer-Verlag, 2000, pp. 157–171, ISBN: 3-540-41093-7. [Online]. Available: http://dl.acm.org/citation.cfm?id=647986.741313.

[112] Y. Zheng, X. Xie, and W.-Y. Ma, "Geolife: A collaborative social networking service among user, location and trajectory", *IEEE Data(base) Engineering Bulletin*, 2010. [Online]. Available: https://www.microsoft.com/en-us/research/publication/geolife-a-collaborative-social-networking-service-among-user-location-and-trajectory/.

[113] D. Ashbrook and T. Starner, "Using gps to learn significant locations and predict movement across multiple users", *Personal and Ubiquitous Computing*, vol. 7, no. 5, pp. 275–286, 2003, ISSN: 1617-4917. DOI: 10.1007/s00779-003-0240-0. [Online]. Available: https://doi.org/10.1007/s00779-003-0240-0.

[114] J. Krumm and E. Horvitz, "Predestination: Inferring destinations from partial trajectories", in *Eighth International Conference on Ubiquitous Computing (UbiComp 2006)*, Springer, 2006. [Online]. Available: https://www.microsoft.com/en-us/research/publication/predestination-inferring-destinations-from-partial-trajectories/.

[115] A.-N.S.T.-G.A.O.-F.L.G.-S.I.B. J. Garcia-Gasulla Dario and J. Vázquez-Salceda, "Social network data analysis for event detection", in *21st European Conference on Artificial Intelligence (ECAI'14)*, 2014.

[116] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: A not-so-foreign language for data processing", in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08, Vancouver, Canada: ACM, 2008, pp. 1099–1110, ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376726. [Online]. Available: http://doi.acm.org/10.1145/1376616.1376726.

[117] *CesiumJS*, https://cesiumjs.org/, 2019.

[118] Y. Zheng, H. Fu, X. Xie, W.-Y. Ma, and Q. Li, *Geolife gps trajectory dataset - user guide*, 2011. [Online]. Available: https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/.

[119] S. J. Yuhaski and J. M. Smith, "Modeling circulation systems in buildings using state dependent queueing models", *Queueing Systems*, vol. 4, no. 4, pp. 319–338, 1989, ISSN: 1572-9443. DOI: 10.1007/BF01159471. [Online]. Available: https://doi.org/10.1007/BF01159471.

[120] D Garbrecht, "Describing pedestrian and car trips by transition matrices", *Traffic Quarterly*, vol. 27, pp. 89–109, 1973.

[121] O. M. Ashford N. and P. D. McGinity, "Stochastic modelling of passenger and baggage flows through an airport terminal", *Traffic Engineering & Control*, vol. 17, no. 5, pp. 207–210, 1976.

[122] D. Helbing, "A stochastic behavioral model and a 'microscopic' foundation of evolutionary game theory", *Theory and Decision*, vol. 40, no. 2, pp. 149–179, 1996, ISSN: 1573-7187. DOI: 10.1007/BF00133171. [Online]. Available: https://doi.org/10.1007/BF00133171.

[123] S. Curtis, A. Best, and D. Manocha, "Menge: A modular framework for simulating crowd movement", *Collective Dynamics*, vol. 1, no. 0, pp. 1–40, 2016, ISSN: 2366-8539.

[124] E. S. Bogardus, "A social distance scale", in *Sociology and Social Research*. 1933, pp. 265–271.