UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RENATO SILVEIRA

Configurable Flows

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Prof. Dr. Luciana Porcher Nedel
Coadvisor: Prof. Dr. Edson Prestes

Porto Alegre
April 2015

CIP – CATALOGING-IN-PUBLICATION

Silveira, Renato

Configurable Flows / Renato Silveira. – Porto Alegre: PPGC da UFRGS, 2015.

115 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2015. Advisor: Luciana Porcher Nedel; Coadvisor: Edson Prestes.

1. Path-planning. 2. Agent navigation. 3. Pedestrian simulation. 4. Potential field. 5. Boundary value problem. I. Nedel, Luciana Porcher. II. Prestes, Edson. III. Título.

“Science is the belief in the ignorance of experts.”

— RICHARD FEYNMAN

ACKNOWLEDGMENT

I would like to express my gratitude to every person that contributed to this thesis.

First of all, I would like to thank my advisors Luciana Nedel and Edson Prestes. Their brilliant observations have helped shape this work.

Next, I would like to thank all my colleagues at the Computer Graphics lab. Their helpful suggestions and discussions really helped me. Thank all my friends who made the present work possible.

A word of thanks goes to Prof. Dr. Antonio Tavares da Costa Junior, who teach me all the mathematics background I know today. Thank Prof. Dr. Benamy Turkienicz for all the motivation to complete the thesis.

I would like to thank all the PPGC/UFRGS members for helping me solve the problems experienced on this journey. Thank CAPES and CNPq for financial support.

Lastly, I would like to thank my family for all the support. This thesis is dedicated to them, especially to the memory of my father José Vítor Silveira.

ABSTRACT

In this work, we propose a new solution to agent navigation based upon boundary value problems (BVP), called Configurable Flows, to control steering behaviors of characters in dynamic environments. We use a potential field formalism that allows synthetic actors to move negotiating space, avoiding collisions, and attaining goals while producing very individual paths. The individuality of each character can be set by changing its inner field parameters leading to a broad range of possible behaviors without jeopardizing its performance.

BVP Path Planners generate potential fields through a differential equation whose gradient descent represents navigation routes from any point of the environment to a goal position. Resulting paths are smooth and free from local minima. In spite of these advantages, these kind of planners consumes a lot of time to produce a solution. Our approach combines a BVP Path Planner with the Full Multigrid Method, which solves elliptic partial differential equations using a hierarchical strategy. The proposed planner enables real-time performance in large environments. Results show that our proposal spends less than 1% of the time needed to compute a solution using the original BVP planners in several environments.

We refine our Path Planner by introducing a new form of the core equation that permits to easily cope with terrain inhomogeneities. This is accomplished by locally changing the concavity/convexity of the potential, and then creating regions with higher or lower navigation preferences. As the potential field requires several steps to converge, this approach can be expensive computationally. To overcome this problem, we integrate this novel core equation to the hierarchical planner, emerging a wide variety of applications. We believe our proposal can contribute to several areas of research including agent navigation, pathfinding for games, crowd simulation and robotics. Our publications reinforce the relevance of the proposed method.

Keywords: Path-planning. agent navigation. pedestrian simulation. potential field. boundary value problem.

Fluxos Configuráveis

RESUMO

A animação de agentes autônomos em aplicações que exigem uma resposta em tempo real ainda é um desafio, se o problema necessitar que o agente alcance um local com precisão em um mundo virtual (planejamento de caminho), mover-se realisticamente de acordo com sua personalidade, intenções e humor. É muito difícil produzir um comportamento natural através de alguma estratégia que foque no controle global dos agentes. Por outro lado, levar em conta a individualidade de cada agente pode ser uma tarefa custosa.

A proposta dessa tese é uma nova solução para o problema da navegação de agentes, baseada em problemas de valores de contorno (BVP), chamada "Configurable Flows", para controlar a navegação de agentes em ambientes dinâmicos. Nós usamos um formalismo baseado em campos potenciais para possibilitar que agentes virtuais se movam negociando o espaço, evitando colisões, e alcançando seus objetivos, através de caminhos diferenciados. A individualidade de cada agente pode ser definida alterando parâmetros internos dos agentes.

Planejadores BVPs produzem campos potenciais através de equações diferenciais das quais o gradiente descendente representa rotas navegacionais de qualquer lugar do ambiente para algum destino. Os caminhos resultantes são suaves e livres de mínimos locais. Apesar dessas vantagens, estes tipos de planejadores consomem muito tempo de execução para produzir uma solução. A nossa abordagem combina nosso planejador BVP com o método *Full Multigrid*, o qual resolve equações diferenciais parciais elípticas usando uma estratégia hierárquica. Os resultados mostram que a nossa proposta gasta menos que 1% do tempo necessário para computar a solução usando os planejadores BVPs tradicionais.

Nós refinamos o planejador introduzindo uma nova forma para o núcleo da equação que permite facilmente lidar com terrenos não-homogêneos. Isto é obtido através de mudanças locais na concavidade/convexidade do potencial, criando regiões com altas ou baixas preferências de navegação. Nós integramos esta nova equação ao planejador hierárquico, surgindo uma ampla variedade de aplicações. Nossa proposta contribui para diversas áreas incluindo a navegação de agentes, *pathfinding* em jogos, simulação de multidões, e a navegação de robôs. Nossas publicações reforçam a relevância e robustez do método proposto.

Palavras-chave: Planejamento de caminho, navegação de agentes, simulação de pedestres, campos potenciais, problema de valores de contorno.

LIST OF ABBREVIATIONS AND ACRONYMS

BVP *Boundary Value Problem*

CF *Configurable Flows*

FMG *Full Multigrid method*

GPU *Graphics Processing Unit*

GS *Gauss-Seidel Method*

HCF *Hierarchical Configurable Flows*

PDE *Partial Differential Equation*

PP *Path Planner*

PRM *Probabilistic Roadmap*

RRT *Rapidly Exploring Random Trees*

RTS *Real-Time Strategy*

SOR *Successive Over-Relaxation method*

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The use of synthetic actors acting as autonomous agents in interactive applications such as games and virtual reality experiences is becoming more and more common (KAPADIA et al., 2013a). Autonomous agents, also called *non-player characters*, are characters with the ability of playing a role in the environment with life-like and improvisational behavior. Some suitable skills for these characters (often simulating human beings) include a realistic appearance, the ability to produce natural movements, and the aptitude to reason and act in an unforeseeable way (NINOMIYA et al., 2014).

The simulation of virtual humanoids moving in a synthetic world involves the semantic representation of the environment, the definition of the agent initial position and its goal (target position). By setting these parameters, an algorithm can be used to find a trajectory to be followed from the initial position to the goal position. The process of detailing this trajectory into discrete motion is known as *path-planning*. This will be the central topic of this thesis.

In real world, if we consider different persons (all in the same initial position) looking to achieve the same target position, each path followed will be unique. Even for the same task, the strategy used for each person to reach his/her goal depends on his/her physical constitution, personality, mood, reasoning, urgency, and so on. From this point of view, a good algorithm to move characters across virtual environments should generate expressive, natural and unexpected steering behaviors. The search for realistic behaviors requires high-performance for real-time graphics applications which compel developers to look for more efficient and fewer expensive methods that produce yet good and almost natural movements, making this a challenging problem.

Many researchers are working on methods to improve the quality of the steering behavior of synthetic agents with a low computational cost (WOLINSKI et al., 2014; GARCIA; KAPADIA; BADLER, 2014; KAPADIA et al., 2013a; KAPADIA et al., 2013b), It is very difficult to produce natural behavior by using a strategy that focuses on the global control of characters. Additionally, taking into account the individuality of each character may be a costly task. As a consequence, most of the approaches proposed in computer graphics literature do not take into account the individual behavior of each agent, compromising the simulation quality.

Despite *humanoid*, *autonomous agent*, and *behavior* being terms used in many different contexts, in this work we limit their usage in order to match our goals. For the sake of simplicity, we consider *humanoids* as a kind of *autonomous agent* with behaviors (driven by stimulus), represented by a computational model, and capable of producing physical manifestations in a

virtual world. The term *behavior* will be used mainly as a synonym of *animation* or *steering behavior* and intends to refer to improvisational and personalized action of a *humanoid*.

1.1 Motivation

Recent advances in computer hardware, especially on GPUs, allow the creation of synthetic actors visually indistinguishable from real actors. They also allow the performance improvement in the realistic and intelligent behaviors, making virtual agents more convincing. This permits the use of more elaborate and robust techniques that customize the path found based on the individuality of the agents' behavior. These behaviors can be represented using a set of constraints or rules that ensures the agent will avoid certain areas, minimize risks, avoid collisions, reduce the cost (usually distance) of the path, and so on.

A wide variety of path-planners has been developed mixing ideas from many fields such as networking, graph theory, search and combinatorial optimization, algorithms and data-structures. Unfortunately, none of the approaches completely solves the problem, because each situation can have different restrictions and assumptions. As an example, Robotics needs paths to a finite-dimensional object that avoid obstacle collisions by some margin (due to sensors imprecision). Computer games need paths to be found very fast, using a restricted quantity of memory.

A great performance is fundamental in both situations, mainly in computer games, since a scene, defined by agents' positioning in the environment, must be rendered with a frequency of about 30 to 100 frames per second to be visually convincing. The application has about 0.02 seconds per frame in order to update the system's states, to handle input and output data, to do graphics processing, to perform physics computation, IA, path planning, and so on. This implies a time less than 1 milliseconds for planning the agents' path.

Considering those issues, the proposed planner provides an efficient and configurable solution to navigation of virtual actors while producing individual and quality paths. The planner is formulated as a Boundary Value Problem (BVP) and is able to handle different situations using this same formalism. Parameters can be configured to sculpt the potential field, producing navigation routes with preferences while keeping the best property of this kind of planner, that is the absence of local minima. The planner's efficiency is obtained through a hierarchical approach.

1.2 Objectives

Our proposal is to create an efficient and robust technique to control the navigation of agents in a virtual world that generates individual and natural behaviors for humanoids. For this, we improve some aspects of a path planner develop by our group (DAPPER et al., 2006). The planner is based on two levels: the global planner and the collision avoidance module. The global path planner is responsible for finding a path through the environment. The collision avoidance module is responsible for controlling the behavior of the agent for avoiding moving obstacles. Despite some applications requiring a blend of the two previous concepts (global planner and collision avoidance), the distinction has many advantages including simplicity, a modular implementation, and easy integration with existing collision avoidance techniques.

This planner had some limitations which made impossible its use in real-time applications. Also, a wide varieties of validations are needed to be done. Our goal was to contribute punctually in these validations and create a new planner, more efficient, to be used in real-time applications. This new planner is our main contribution.

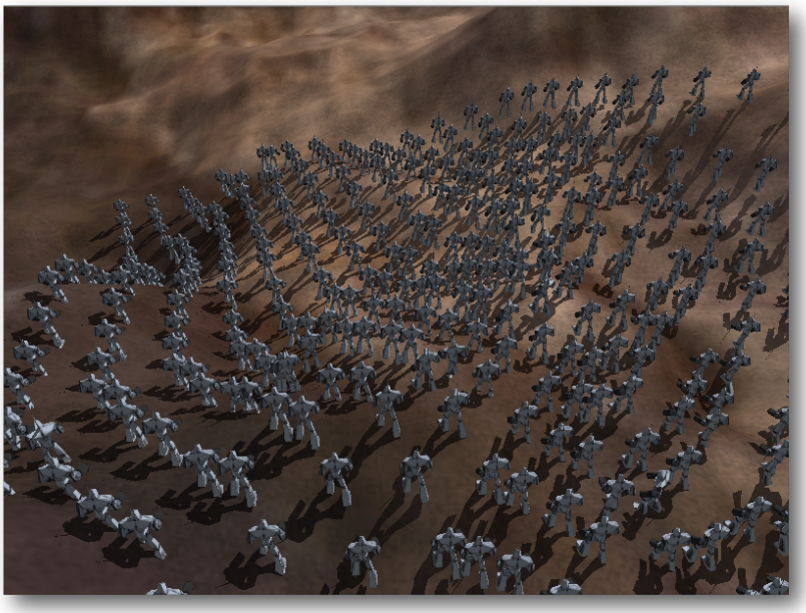


Figure 1.1 – Multiple agents simulation.

Source: Created by the author.

Efficiency is achieved by reducing the complexity of the problem by means of a hierarchical approach where the environment is represented in several levels of discretization. Paths on the low-resolution level are computed very fast and is used to accelerate subsequent levels where

paths are refined and smooth. In each level of the hierarchy, a potential field is computed, whose gradient descent represents navigational routes. The resulting trajectories are smooth and free of local minima. The advantage of the proposed technique compared with existing agent navigation techniques is that the same principle is used to achieve a wide variety of features required for agent's navigation and simulation. These features include high-quality paths (smooth and natural paths), collision avoidance from static and moving obstacles, distinctive behavior for each agent and real-time performance.

Figure 1.1 illustrates a screenshot of multiple agents simulation using our method. Basically, the proposed technique should be able to deal with:

- **Multiple agents:** a large number of agents moving around in the virtual environment must be handled and those must avoid each other.
- **Agents' individuality:** different paths should be produced based on agent's distinctive behavior, which represents individual characteristics of each agent like its own personality, intentions and mood. These characteristcs should be dynamically changed, impacting the generated path.
- **Real time:** agents' motion must be computed in real time. The planner should be able to provide a path in real time while the simulation is being executed.
- **Dynamics:** virtual environments should contain not only static obstacles, but also dynamic obstacles. They must be handled in real time.
- **Coherence:** a group of agents in a virtual environment may share the same goal. The agents must stay together and follow the same route to the goal, while keeping their individual paths, avoiding obstacles.
- **Inhomogeneous Terrain:** The potential field should be adjusted to generate regions with higher or lower preference which helps the designer to deal with terrain reasoning (STERREN, 2001) and tactical planning at the path planning step. The same principle used to produce the potential field is used in order to achieve these features.

1.3 Applications

In this section, we discuss some potential applications for our technique in the games industry, the film industry, robotics and simulation industry.

1.3.1 Games

Real-time path planning in games is a challenging and still unsolved problem (NIEUWENHUISEN; KAMPHUIS; OVERMARS, 2007), since good paths and real-time response are needed. There are several workaround done by the game industry to adjust path planners to the game needs. One of the most well-known game styles is Real-Time Strategy (RTS). In such games, the user commands groups of agents, usually over a planar surface. Figure 1.2 shows a screenshot of the *Starcraft II* (ENTERTAINMENT, 2010) game, where a group of *zergs* are attacking the *terran's* base. Details of the navigation technique are not shared by the company, but several bugs and unnatural paths are criticized by players. The most common issues of path-planners reported by gamers include the quality of motion, occurrence of collisions, repeated motions, reaction on dynamic changes and unnatural behavior of groups of characters.

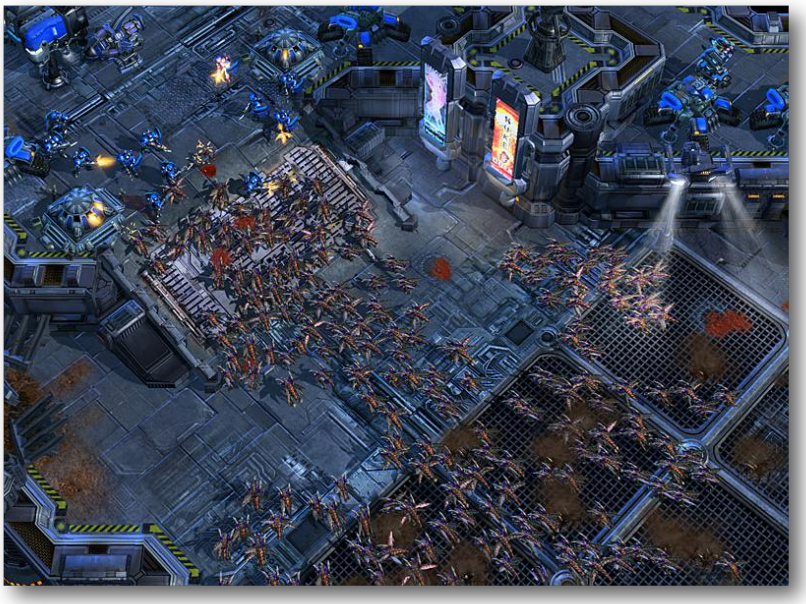


Figure 1.2 – *Starcraft II*. Multi-agent navigation.

Source: <http://www.365waystopay.com/play-starcraft-2/>. Last access: February 2015.

In addition to criticism of bugs and unnatural paths another comment is the fact that game path-planners provide the shortest path which is quite stereotyped and it is not exhibited by humans during navigation. Generally, natural paths are close to the shortest path, but not exactly the shortest path. Another example of games that highlight agent navigation and simulation are the games *Dead Rising* (CAPCOM, 2006) and *Assassin's Creed* (UBISOFT, 2007). In *Dead Rising*, the main character should escape from a crowd of zombies. In *Assassin's Creed*, the

main character is an assassin who masquerades in the crowd. In both games, a crowd simulation technique is used to populate the environment. Figure 1.3 illustrates the crowd in *Dead Rising* game.

Figure 1.3 – Dead Rising. This game uses crowd simulation to populate the environment.

Source: <http://newenglandgamer.com/dead-rising-film-works/>. Last access: February 2015.

1.3.2 Movies

Another application that uses large numbers of agents is cinema. In this context, virtual actors are used for populating a scene since they are cheaper than using real actors. For instance, in the trilogy *Lord of the Rings* (CINEMA, 2003), illustrated in Figure 1.4, epic wars were populated by virtual actors.

For a greater realism, the virtual actors need to move and act as a crowd. To simulate this situation, human behavior and interaction were analyzed to replicate the collective behavior of crowds. Dealing with a large amount of virtual characters, conventional animation techniques, such as path sketching, are not sufficient to represent and handle the whole crowd. This results in the need for specific techniques to control crowd trajectories, as well as the individual trajectories of each agent in the crowd.

Unlike games, in the movie industry the real-time performance is not the bottleneck. There is a greater exigency in the realism of the behavior of virtual actors. The path planner has to provide high-quality paths to achieve it.

Figure 1.4 – Battle of Helms Deep in the movie *Lord of the Rings*. Most of the people are virtual.

Source: <http://lotr.wikia.com/wiki/Battle_of_the_Pelennor_Fields>. Last access: February 2015.

1.3.3 Robotics

One of the main goals of robotics is to create robots that can act autonomously in the environment. This goal is very complex and requires advances in several domains, as localization, mapping, and planning. At the autonomous navigation domain, the path planner should produce a fast answer, particularly in problems that involve risk to human life, like rescue operations during natural disasters, for instance. In these cases, robots are expected to act efficiently and spending a minimum time in path planning. Figure 1.5 illustrates the robot navigation. The feasibility of real-time path planning is dependent on the accuracy of the map acquired by the robot sensors. As there are inaccuracies in the robot's sensors, one requirement is that the path planner must provide a path with clearance from obstacles.

1.3.4 Pedestrian Simulation

Modern airports, shopping centers, bridges and train stations are complex buildings with a high frequency of pedestrians, as illustrated in Figure 1.6. People must be able to move inside these buildings without disturbances and with comfort, and it is crucial that in emergency cases people can be evacuated safely and in time. Pedestrian jams caused by unknown bottlenecks are disastrous and few seconds can be determinant over life and death. To evaluate a problem-free walkability and a secure emergency evacuation in complex buildings the pedestrian simulation is necessary. The agent-based pedestrian simulation allows the flexible modeling of pedestrian behavior based on the architectural topology of the building or environment in question. Several

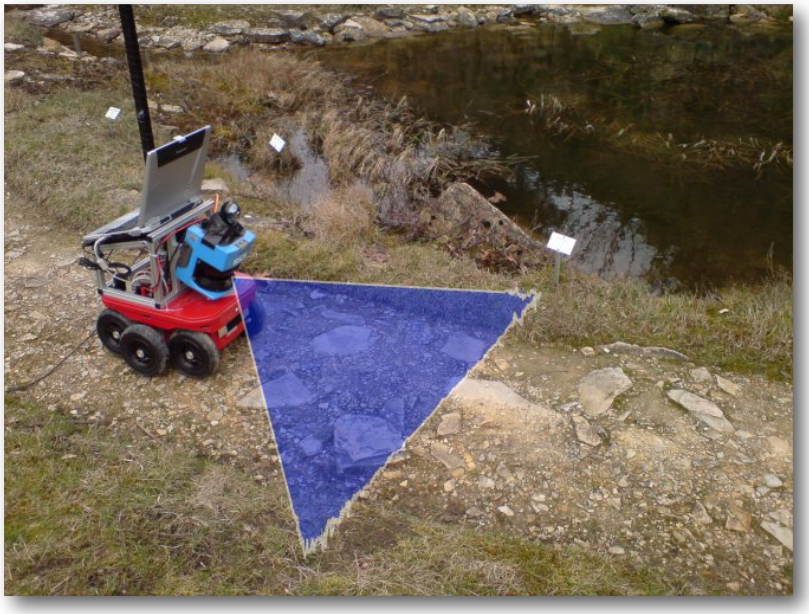


Figure 1.5 – Robot navigation. A quality path with clearance from obstacles is required.

Source: (BARTEL et al., 2007). Page 1.

areas benefit from this simulation, such as: design, safety and egress audit of public spaces and buildings (train stations, airports, hospitals, public places, etc.), control and improvement of pedestrian flows in urban planning and architecture, walkability studies, implementation of human movement in traffic scenarios and marketing research tool.

1.4 Organization

The rest of the text is organized as follow:

- **Chapter 2** presents the existing approaches to handle the path-planning problem. They are classified as Grid Based algorithms, Roadmaps algorithms including Visibility Graphs, Probabilistic Roadmaps, Rapidly Exploring Random Tree, Waypoints and Potential Fields. Finally, Collision Avoidance techniques are presented to handle local collisions.
- **Chapter 3** shows the fundamental concepts used to develop the formalism of Configurable Flows. Also, it shows the strategy to use the Configurable Flows as global and local planner. This chapter introduces a way to sculpt the potential field, changing its concavity/convexity and configuring the flows, which allows path-planning in terrains with different traversal characteristics.
- **Chapter 4** presents the Hierarchical Configurable Flows Path-Planner, which is an effi-

Figure 1.6 – Pedestrian in a daily situation. Simulate pedestrian is important to predict and prevent accidents.

Source: <http://www.kennislink.nl/publicaties/waarom-we-altijd-te-weinig-tijd-hebben>. Last access: February 2015.

cient version of Configurable Flows suitable for being used in real-time applications.

- **Chapter 5** presents quantitative and qualitative results obtained using both planners.
- **Chapter 6** presents our conclusions and future works.
- **Chapter 7** shows contributions obtained during the Ph.D..
- **Chapter 8** provides supplemental materials to complement the theory of presented methods.

2 RELATED WORK

In the literature, there are many different terms for the problem addressed in this proposal, such as motion planning, path planning, navigation, route planning and pathfinding. Motion Planning, also known as “the Piano Mover’s Problem” (SCHWARTZ; SHARIR, 1983) is a fundamental problem in robotics (LAVALLE, 2006) . The classical motion planning problem can be described as the following: given a three-dimensional rigid body and a known set of obstacles, the task is to find a collision-free path from a start configuration to a goal configuration.

Most of the time, this problem becomes to find a collision-free trajectory that will move a robot from its initial position to its final position. This type of motion planning problem is commonly called Path Planning. Path planning, navigation, and pathfinding are terms used in many different contexts. In this proposal, we will use these terms with the same meaning. Figure 2.1 illustrates the path planning problem.

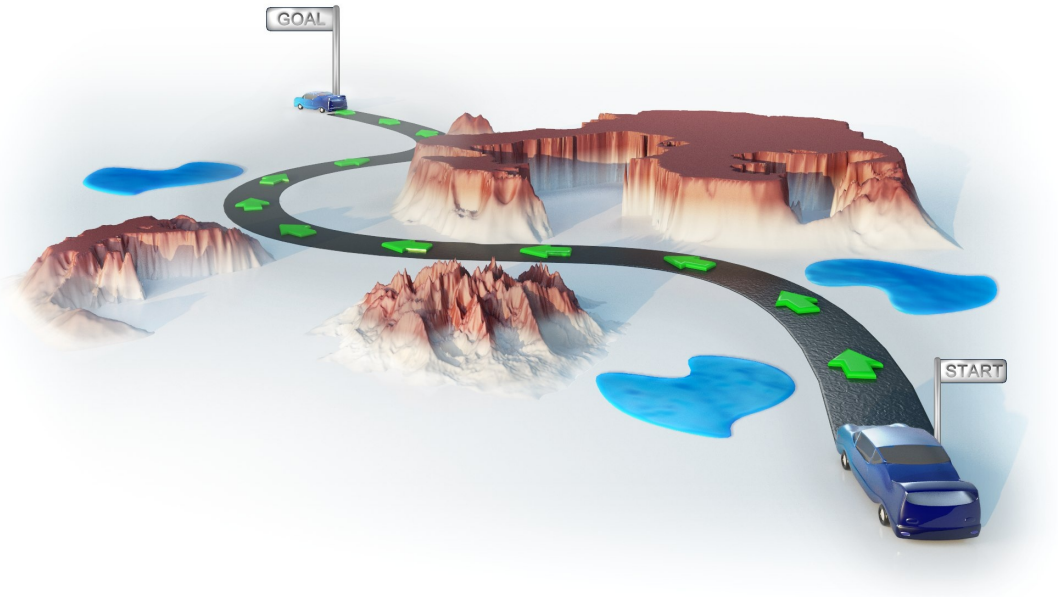


Figure 2.1 – Path planning problem: find a path that takes an agent from its initial position to its goal position without colliding with obstacles.

Source: <http://marius.sucan.ro/blog/2008/new-works/>. Last access: February 2015.

The basic principles to generate a path between two known positions in a bi-dimensional world are relatively simple and well-studied in the robotics field. However, to find the path is not enough when we want to endow artificial characters with natural and realistic movements similar to humans. Most of the approaches proposed in computer graphics literature use two-step strategies. In the first step – path planning step – a valid path is defined, and in the second

one – the collision avoidance step – this path is adjusted to fit a realistic movement. These steps are often addressed as separate problems that need to be interfaced with a fully functional navigation system.

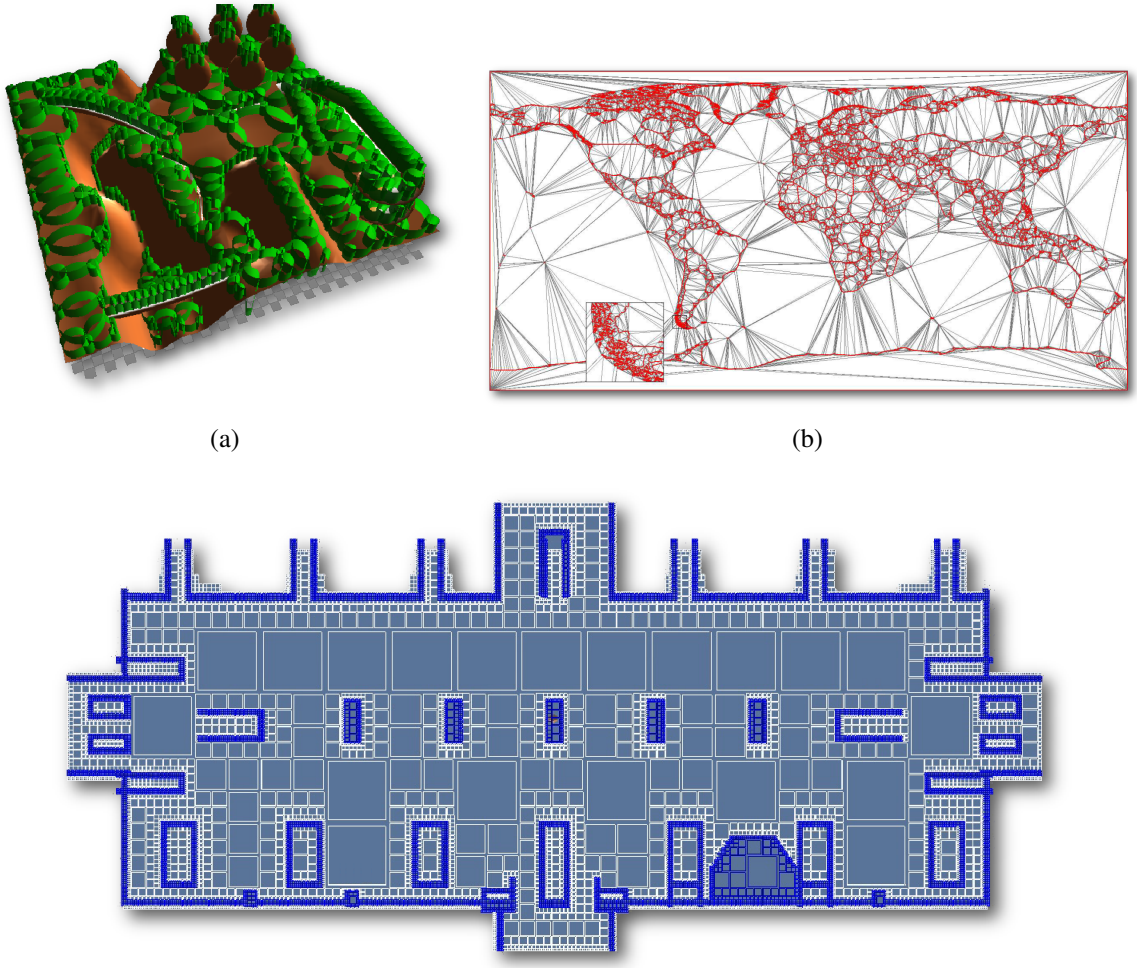
A path planning algorithm usually have a representation of the environment (with a graph or a discretization) and then searches for a path between the initial and the goal positions for the agent. There is a large number of methods for solving the basic path planning problem. Not all of them solve the problem in its full generality. Despite many external differences, the methods are based on few different general approaches which are: cell decomposition or grid based, roadmaps, and potential fields.

2.1 Grid Based Algorithms

Grid-based or Cell Decomposition methods consist of decomposing environment into cells. Two general methods can be distinguished: the approximate cell decomposition and the exact cell decomposition. The approximate cell decomposition consists of using predefined cell shapes (uniform grids, quadtrees, circles) (BANDI; THALMANN, 1998; SHILLER; YAMANE; NAKAMURA, 2001; PETTRé; LAUMOND; THALMANN, 2005; SHAO; TERZOPOULOS, 2005), whose union is the environment discretization. The exact cell decomposition consists of computing cells such that their union is exactly the environment (constrained Delaunay triangulation, convex polygons, trapezoidal) (KALLMANN; BIERI; THALMANN, 2003; LAMARCHE; DONIKIAN, 2004). Figure 2.2 illustrates these decompositions.

The representation complexity of the latter method is directly dependent on the input geometry complexity. If the environment is small and fits well into a grid, this technique is useful. However, when the environment becomes complicated and large, grid based methods take a large amount of computation time. With the environment discretization, a search algorithm is used to obtain a sequence of cells which the agent must traverse in order to go from the initial position to the goal position. The path created by grid search tends to look unnatural, being a rough approximation for the smooth path that a human would follow. To produce smooth paths using grid searches, the paths need to be smoothed in the post-processing phase resulting in expensive queries.

Search algorithms are responsible for finding paths, whereas the search space is the discretization of the environment. We can think of the search space as a map, while the search algorithm is the set of rules that allows to find a path on the map. There are several search algorithms used for navigation, such as the Best-first search (PEARL, 1984), Breadth-first search (PEARL,



(a) (b)

(c)

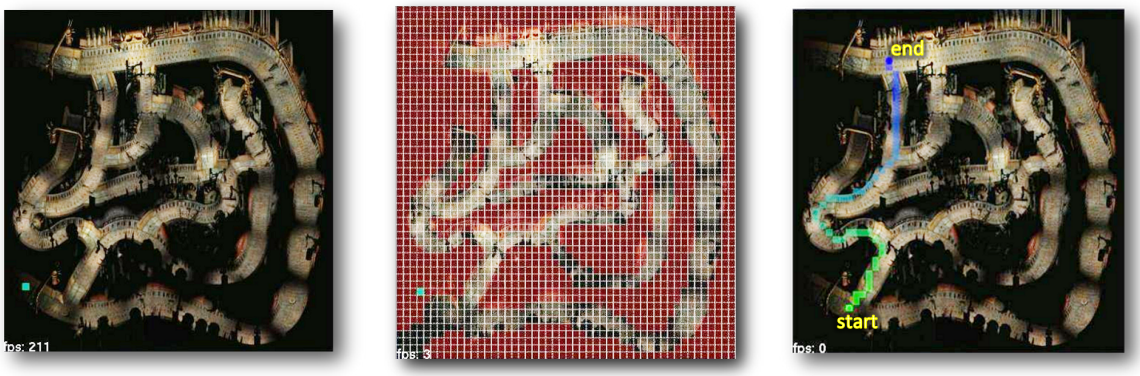
Figure 2.2 – Cell decomposition consisting of circles (a), a Delaunay triangulation (b) and a Quadtree (c).

Source: (a) (PETTRé; LAUMOND; THALMANN, 2005), page 6, (b) (KALLMANN; BIERI; THALMANN, 2003), page 18 , (c) (SHAO; TERZOPOULOS, 2005), page 22.

1984), Depth-first search (PEARL, 1984), D* (STENTZ; MELLON, 1993), Dijkstra (DIJKSTRA, 1959), and so on. Their use depends on the amount of available memory and the way that the search space should be available.

One of the most used search algorithms was proposed by Hart et al. (HART; RAPHAEL, 1968), called A*. A* is an improved version of the Dijkstra (DIJKSTRA, 1959) work. Dijkstra introduced a heuristic graph search algorithm that, given a source and a destination node, finds the shortest path connecting the two nodes. Initially, the algorithm finds the distance between the source and all nodes of the graph. Next, it determines the path with minimal cost, following the graph nodes from the source to the destination. A* works in the same way, but it uses a

heuristic to reduce the search space. Figure 2.3 illustrates the path produced by the A* algorithm. First, the environment (Figure 2.3 (a)) is discretized (Figure 2.3 (b)), and then, a path is found connecting the green cell to the blue cell ((Figure 2.3 (c)).



(a) (b) (c)

Figure 2.3 – Path produced by the A* algorithm: the environment (a), its discretization (b) and the path found connecting the green cell to the blue one (c).

Source: Created by the author.

Several variations of the original A* algorithm have been proposed. The recent work of Bleiweiss (BLEIWEISS, 2008), Kapadia et al. (KAPADIA et al., 2013b) and Garcia et al. (GARCIA; KAPADIA; BADLER, 2014) presented a technique to implement graph search algorithms in the graphics hardware. Others focus on inputting behavior into the search algorithm, besides finding the shortest path. For instance, Hara et al. (HARABOR; BOTEA, 2008a) recent work that plan a path for agents with different sizes in heterogeneous environments and Huang et al. (HUANG MUBBASIR KAPADIA; KALLMANN, 2014) that plan a path for coherent and persistent groups in arbitrarily complex environments, producing splitting and merging behaviors. Also, the works of Ninomiya et al. (KAPADIA et al., 2013c; NINOMIYA et al., 2014) that present a path planner that satisfies multiple spatial constraints imposed on the path such as staying behind a building, walking along walls, or avoiding the line of sight of patrolling agents.

Considering grid based discretizations, Kang et al. (KANG; KIM; KIM, 2010) use an adaptive Delaunay discretization and a map of regions of interest to plan a path of agents enabling user interaction. Wang et al. (WANG; DANG; PAN, 2010) uses a time-variable grid discretization to find paths in unknown environments. Torchelsen et al. (TORCHELSEN et al., 2010) uses the triangular discretization that represents the 3D surface mesh to find a path on arbitrary surfaces. Finally, Kallmann et al. (KALLMANN, 2010) uses a new type of navigation meshes to have the environment discretization to find quality shortest-paths.

2.2 Roadmap

Roadmap is a topological graph which represents an abstraction of the environment where each vertex is a position and each edge is a path that connects two positions (LAVALLE, 2006). This graph must contain a sufficient amount of paths to make any path planning query easily solvable. Basically, roadmap techniques can be described in three steps. First, a graph that contains the connectivity of the environment is built. Then, the agent initial and goal positions are added as vertices in this graph. Finally, a path connecting the initial and goal positions is searched, using a search algorithm like A* mentioned in Section 2.1. Different approaches can be used to create roadmaps, the most used are Visibility Graph, Probabilistic Roadmap, Rapidly Exploring Random Trees and Waypoints. Generally, they differ in the way the graph is built.

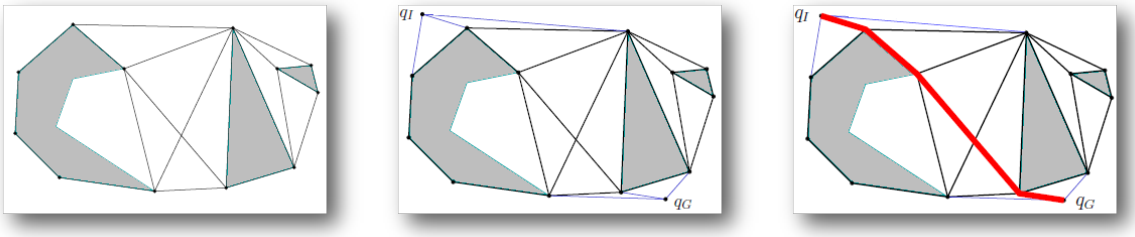
Most of the paths produced by roadmap methods are piecewise linear, have many redundant trajectories, and little clearance to the obstacles, resulting in unnatural paths. While techniques exist for smoothing the paths (GERAERTS; OVERMARS, 2004; KAMPHUIS et al., 2004; KARAMOUZAS; OVERMARS, 2008) they are too slow to be applied in the query phase in real-time applications. There are several works using roadmaps for agent navigation, such as Lamarche's work (LAMARCHE, 2009) that analyzes unstructured 3D triangular meshes to construct the roadmap and the work of Roland et al. (GERAERTS; OVERMARS, 2007) that creates a special kind of roadmap, called Corridor Maps, that generates a corridor through a path queried in the roadmap and uses a potential field inside this corridor to steer the agent's navigation.

2.2.1 Visibility Graphs

The Visibility Graph (LOZANO-PéREZ; WESLEY, 1979) connects together vertices of the environment's geometry if and only if they see each other. Then, the agent initial and goal positions are added as vertices in this graph and a path is searched. Figure 2.4 exemplifies this method. Grey polygons are obstacles and the graph is created connecting two vertices that are visible to each other (Figure 2.4 (a)). To query for a path, a start vertex (q_I) and a goal vertex (q_G) is inserted in the graph (Figure 2.4 (b)) and a path is found connecting those two vertices, represented by the red line, in Figure 2.4 (c).

This approach ensures finding the shortest path between two configurations, but it can be memory consuming in open environments with sparse obstacles. Also, the generated paths are very close to obstacles, which implies on unnatural movement of agents. An alternative is to

compute the generalized Voronoi diagram (Ó'DÚNLAING; YAP, 1985) inside the environment, which generates paths that maximize clearance with obstacles.



(a) (b) (c)

Figure 2.4 – Visibility Graph. Connect vertices that see each other (a), add the initial position (q_I) and final position (q_G) to the graph (b) and search for a path (red line) (c).

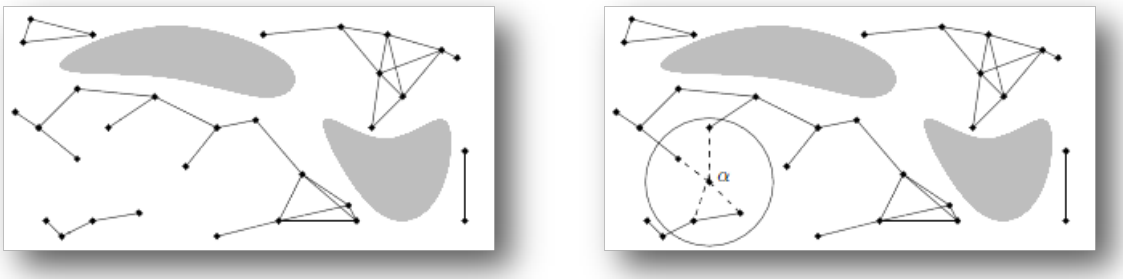
Source: Created by the author.

2.2.2 Probabilistic Roadmap Method

The Probabilistic Roadmap Method (PRM) consists of generating vertices by randomly sampling positions in the environment (Figure 2.5 (a)) and linking those positions if a collision-free path connects them (Figure 2.5 (b)). Due to the random nature of the PRM method, the paths generated have low quality and can take long detours. Such method has been used to plan paths inside populated 2D virtual environments (BAYAZIT; LIEN; AMATO, 2002; BERG et al., 2008) or huge 3D environments (SALOMON et al., 2003) and has demonstrated its capacity to deal with relatively high dimensional problems (KAVRAKI et al., 1996).

2.2.3 Rapidly Exploring Random Trees

Another way of generating alternative paths is through the Rapidly Exploring Random Trees (LAVALLE, 1998) (RRTs). An RRT is a tree where the root is the initial position and the some leaves converge to the goal. This method is particularly suitable for solving single-query path planning problems in high dimensional spaces. Due to their random nature, RRTs seem ideal for obtaining different paths given a specific path query. Although the construction method is simple, it is not an easy task to find a method that acts in this tree and yields a desirable behavior. Figure 2.6 illustrates this tree in a square region in the plane. The center of the square region is chosen as the initial position (Figure 2.6 (a)), and the tree expands in a few

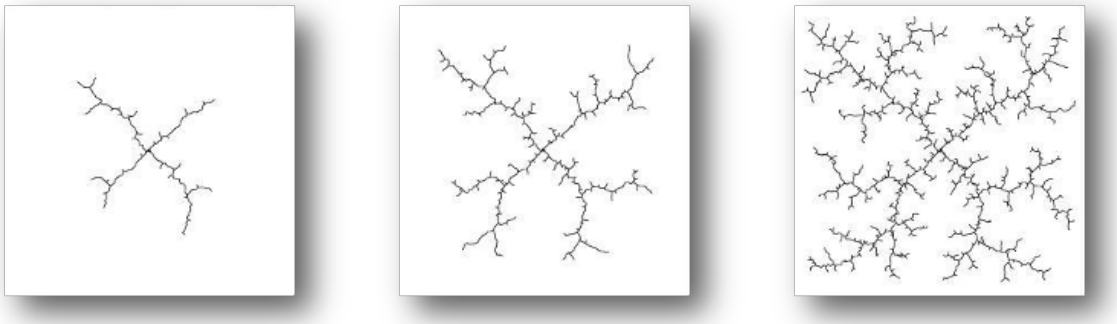


(a) (b)

Figure 2.5 – PRM. The graph representing the environment is built through randomly sampled vertices (a). The vertex α is sampled and linked to the nearest vertices if a collision-free path connects them (b).

Source: Created by the author.

directions (Figure 2.6 (b)) to explore the four corners of the square (Figure 2.6 (c)).



(a) (b) (c)

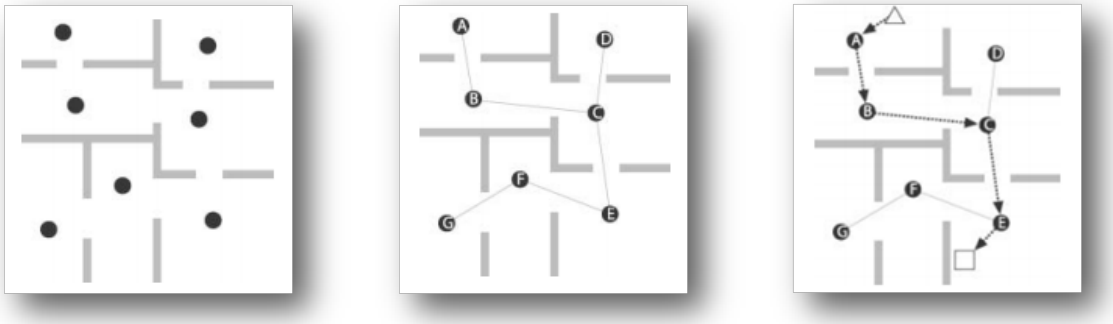
Figure 2.6 – Rapidly Exploring Random Trees. The tree's growth after some iteration are showed in (a),(b) and (c).

Source: <http://msl.cs.uiuc.edu/rrt/about.html>. Last access: February 2015.

2.2.4 Waypoints

The Waypoint method is based on a graph whose vertices and edges are manually placed by the designer to get the most efficient environment representation (Figure 2.7 (a)). Traveling from one waypoint to another, like other roadmap methods, is expressed as a graph search problem (Figure 2.7 (b) and (c)). This method has the benefit of representing the environment with the least amount of vertices for the path planner to deal with. Waypoints are useful for

creating efficient obstacle free pathways through static environments but are unable to deal with dynamic environments.



(a) (b) (c)

Figure 2.7 – The designer specifies some points (a) and creates a graph connecting these points (b). The agent initial and goal positions are added to the graph (c) and a path is queried.

Source: Created by the author.

2.3 Potential Fields

In a classical work in the field of robotics, Khati (KHATIB, 1980) proposed an elegant path planning method. He considered that instead of looking for a good path and trying to control the agent's movement around it a good planner should provide a potential field, or a force field (its gradient), that spanned the whole region of maneuver, producing a continuum of alternative paths. The potential field is devised to incorporate obstacles and goals, and should guide the agents at all times indicating the best direction to follow. Figure 2.8 illustrates this method. Its most straightforward implementation is a simple superposition of fictitious forces: obstacles forces that repel the agent to prevent collisions; and target forces that attract the agent. Such superposition is not always successful since for some environment configuration the agent can end up trapped in local minima before reaching the target.

The classical use of potential fields often presents local minima and unnatural paths (KOREN; BORENSTEIN, 1991). Figure 2.9 illustrates this problem. The agent gets stuck in the environment due to the forces that act on the agent, canceling each other. This problem can be partially solved by providing good potential functions or by executing random moves (BARRAQUAND; LANGLOIS; LATOMBE, 1992). The path from the start to the goal can be found by following the direction of the steepest descent of the potential toward the goal. While this

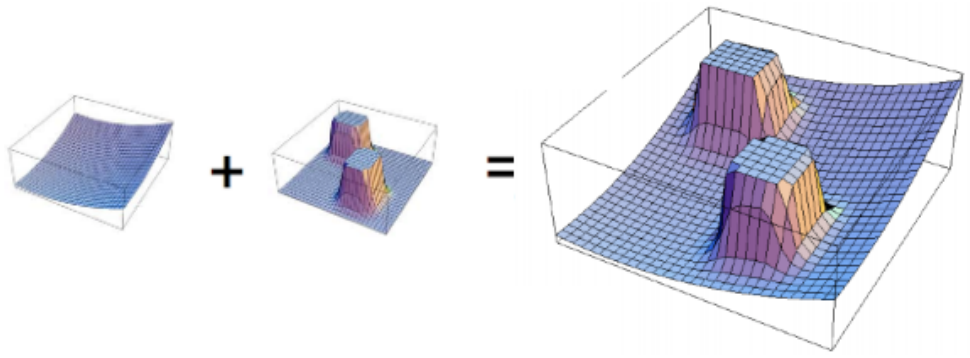


Figure 2.8 – Potential Field. The potential is generated by the superposition of fictitious forces: obstacle forces that repel the agent to prevent collisions; and target forces that attract the agent.

Source: <https://taylorwang.files.wordpress.com/2012/04/potential-field1_robot.jpg>. Last access: February 2015.

method has some flexibility to avoid local hazards (such as small obstacles and other moving objects), it is not useful for path planning in interactive virtual environments as it might take too much time to create the path. Since a change in target or environment requires significant re-computation, these navigation methods are generally confined to systems with non-changing goals and static environments.

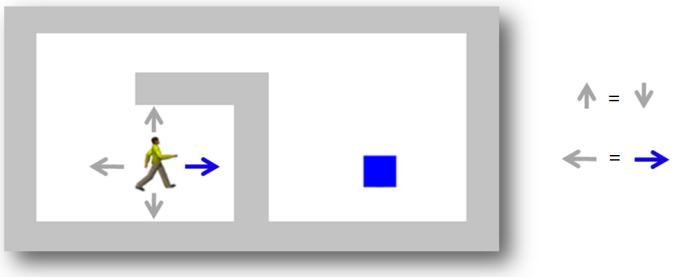


Figure 2.9 – Local Minimum. The agent gets stuck in the environment due to the forces that act on the agent canceling each other.

Source: Created by the author.

2.4 Crowd Simulation

Researches from a broad range of fields, as architecture (SCHELHORN et al., 1999), robotics (MOLNAR; STARKE, 2001), computer graphics (REYNOLDS, 1987; HODGINS; BROGAN, 1994; BOUVIER; GUILLOTEAU, 1996; MUSSE; THALMANN, 1997) and physics

(HELBING; MOLNáR, 1995) have attempted to simulate collective behavior by computer models. These models focus on different aspects of the collective behavior that emerges in a spontaneous way when people act as a crowd. There are several research branches in crowd simulation, such as crowd animation, crowd behavior generation, crowd interaction and crowd rendering (THALMANN; MUSSE, 2007). The recent works that follow the crowd animation branch are relevant to this proposal.

Shao et al. (SHAO; TERZOPOULOS, 2005) emulate the rich behavior complexity of real pedestrians in urban environments. The virtual environment is represented by a hierarchical collection of maps: a topological map, a perception map and a path map. These maps are used in the perceptual, behavioral, and cognitive control steps and are handled by a state machine. This technique has the ability to produce prodigious quantities of intricate animation of pedestrians carrying out various individual and group activities.

Treuille et al. (TREUILLE; COOPER; POPOVIć, 2006) proposed a model based on continuum dynamics. Continuum dynamics is a branch of dynamics that deals with the analysis of the kinematics and the dynamical behavior of materials modeled as a continuous mass rather than as discrete particles. The crowd is handled through a continuum potential field that simultaneously integrates global navigation with moving obstacles such as people. This solves the motion of large crowds without the need for explicit local collision avoidance. This technique can simulate thousands of people at real-time frame rates. Figure 2.10 (a) illustrates this technique.

Pelechano et al.(PELECHANO; ALLBECK; BADLER, 2007) create the HiDAC system for High-Density Autonomous Crowds. This system uses a combination of psychological and geometrical rules with a social and physical forces model to produce a variety of emergent behaviors. These behaviors varying from agent line formation to pushing behavior, i.e., behavior relative to the current situation, personalities of the individuals and perceived social density. This system can handle up to 1800 agents with real-time frame rate.

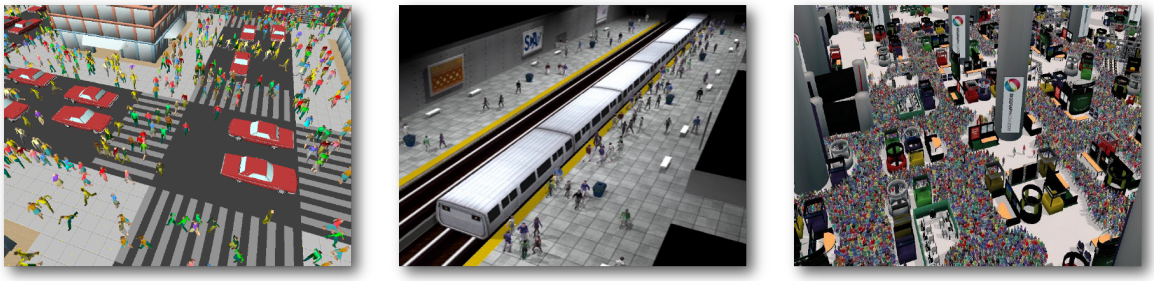
Lerner et al. (LERNER et al., 2007) present an example-based crowd simulation technique that allows autonomous agents to display complex natural behaviors that are often missing in crowd simulations, like subtle variations of the people speed and direction, unpredictable behaviors, like people stop and change direction and walk against the flow. Examples are created from tracked video segments of real pedestrian crowds. During a simulation, autonomous agents search for examples, in the video data, that closely match the situation that they are facing. Trajectories taken by real people in similar situations are copied to the simulated agents, resulting in natural behaviors. Although the system enables agents to exhibit natural behavior,

only the crowd consisting of few people runs in real-time.

Yeh et al. (YEH et al., 2008) model agent interactions through the use of *composite agents*. Each composite agent consists of a basic agent that is associated with one or more proxy agents. This formulation allows an agent to exercise influence over other agents greater than that implied by its physical properties, which allows modelling a variety of emergent behaviors such as aggression, social priority, authority, protection and guidance. Figure 2.10 (b) illustrates this technique.

Van den Berg et al. (BERG et al., 2008) use a PRM to represent a cognitive map of the environment to provide a macroscopic pathfinding, and combines it with fast and localized navigation for each agent. During runtime, each agent travels toward its goal using the precomputed roadmap of the static environment taking into account various constraints (such as speed limit, path clearance, shortest ways, fastest routes), while purposely avoiding collision with nearby obstacles or incoming agents through local rules. This technique can handle thousands of agents using a multi-core implementation.

Narain et al. (NARAIN et al., 2009) modeled the crowd as a continuum fluid described by a density and flow velocity. Local collision avoidance is mapped into a continuous domain to obtain a variational constraint on the crowd flow, which is introduced as the *unilateral incompressibility* constraint. This constraint acts as a large-scale collision avoidance step to accelerate the simulation. The flow varies from freely compressible when the density is low to incompressible when the agents are close to each other. This technique can handle thousands of agents in small grids (40×40 cells) in real time. Figure 2.10 (c) illustrates this technique.



(a) (b) (c)

Figure 2.10 – Crowd simulation. Illustrations from the works of Treuille et al. (a), Yeh et al. (b) and Narain et al. (c)

Source: (a) (TREUILLE; COOPER; POPOVIć, 2006), page 1, (b) (YEH et al., 2008), page 9, (c) (NARAIN et al., 2009), page 1.

Yersin et al. (YERSIN et al., 2009) extend the concept of motion patches (LEE; CHOI; LEE,

2006) to densely populate large environments. They build a population from a set of blocks containing a pre-computed local crowd simulation. Each block is called a crowd patch. They address the problem of computing patches, assembling them to create virtual environments, and controlling their content to answer designers' needs. The main advantage is the handling of large-scale environments and populations by drastically lowering the need for computation resources dedicated to simulation. Figure 2.11 (a) illustrates this technique.

Lerner et al. (LERNER et al., 2009) present a data-driven approach for fitting behaviors to simulated pedestrian crowds. The method annotates agent trajectories, generated by any crowd simulator with action tags. In a preprocessing stage, the stimuli which motivate a person to perform an action are encoded in examples. Using the examples, specific influence functions are encoded into two-dimensional maps which evaluate, for each action, the relative importance of a stimulus within a configuration. At runtime, given an agent stimulus configuration, the importance of each stimulus is determined and compared to the examples. Then, the probability of performing each action is approximated and an action tag is chosen accordingly. This technique is not suitable if real-time performance is required.

Ju et al. (JU et al., 2010) proposed a method that blends existing crowd data to generate a crowd animation. The animation can include an arbitrary number of agents, extends for an arbitrary duration, and yields a natural-looking mixture of the input crowd data. This animation is accomplished by introducing a *morphable crowd* model that allows to encode the formations and individual trajectories in crowd data. Then, its original spatiotemporal behavior can be reconstructed and interpolated at an arbitrary scale. This technique can handle almost one hundred agents in real time.

Sewall et al. (SEWALL et al., 2010) proposed a method for the synthesis and animation of realistic vehicle traffic flows. This technique is based on a continuum model of traffic flow using single-lane continuum flow model to handle multi-lane traffic. This correctly handles lane changes and merges, as well as traffic behaviors due to changes in speed limit. This technique is limited to networks of highway-class roads.

Guy et al. (GUY et al., 2010) use an optimization method to compute a biomechanically energy-efficient, and collision-free trajectory that minimizes the amount of effort for each heterogeneous agent in a large crowd. Also, it is possible to automatically generate many emergent phenomena such as lane formation, crowd compression, edge and wake effects. This technique can interactively simulate large crowds with thousands of agents. Figure 2.11 (b) illustrates this technique.

Bicho et al. (BICHO et al., 2012) propose a method for interactively controlling the move-

(a) (b)

Figure 2.11 – Crowd simulation. Illustrations from the works of Yersin et al. (a) and Guy et al. (b)

Source: (a) (YERSIN et al., 2009), page 1, (b) (GUY et al., 2010), page 11.

ments of crowds where individual agents affect each other by competing for the space where they move. This was motivated by space colonization algorithm, classically used to model leaf venation pattern and the branching architecture of trees. Emergent behaviors like collision avoidance, relationship of crowd density and speed of agents, and the formation of lanes are observed.

Kapadia et al. (KAPADIA et al., 2013a) proposes a real-time planning framework for multi-agent navigation that uses multiple heterogeneous problem domains of differing complexities for navigation in large and dynamic virtual environments. A path planning problem (start and goal configuration) is dynamically decomposed into a set of smaller problem instances across different domains, where an anytime dynamic planner is used to efficiently compute and repair plans for each of these problems. It ensures precise navigation control and little computational overhead.

Dutra et al. (DUTRA et al., 2013) propose the use of precomputed potential fields combined with RVO (BERG; LIN; MANOCHA, 2008) – described in the next section – enabling the change of the agent's goal momentary through local goals. These goals provide different behaviors for a group of agents.The potential field is a modified version of those proposed by Treuille et al. (TREUILLE; COOPER; POPOVIć, 2006) and is used to control the velocity of agents in a group. The RVO is used to handle collision among agents. This technique can handle thousands of agents in interactive rates.

Golas et al. (GOLAS; NARAIN; LIN, 2014) postulated that modeling interpersonal forces is necessary for simulating crowd turbulence. To validate their hypothesis they propose a novel

model for turbulent crowds, being able to simulate stop-and-go waves as well as chaotic behavior symptomatic of crowd turbulence. The model shows good correspondence with quantitative metrics proposed for detecting crowd turbulence and establishes the importance of modeling friction for simulating crowd turbulence.

Musse et al. (MUSSE; CASSOL; JUNG, 2012) propose a new model to quantitatively compare characteristics of two crowds. The local velocity and spatial position is compared using a histogram distances, in 4-D. There are very few works proposing quantitative metrics to compare crowds characteristics. The results of this work correlate with visual inspection.

Berset (BERSETH; KAPADIA; FALOUTSOS, 2013) presented a framework that aims to estimate the complexity of a steering situation given the configuration of the obstacles and the agents involved. The core of the framework is a novel set of complexity-related features and their combination into a single metric, the scenario complexity. Their statistical experiments with three steering algorithms showed a strong negative correlation between our metric and the dynamic performance of the algorithms. This work complements recent research in evaluating crowd techniques and provides a strong foundation for developing a standard suite of challenging benchmarks for testing and comparing crowd simulation algorithms.

Another work to compare and evaluate crowd simulation was proposed by Wolinski et al. (WOLINSKI et al., 2014). They created a framework to evaluate multi-agent crowd simulation based on real-world observations of crowd movements. The idea is to automatically estimate the parameters that enable the simulation algorithms to best fit the given data through a combinatorial optimization problem.

The main similarity between most of the crowd simulation techniques is the use of a global path planner and local collision avoidance technique. Each of these techniques has also been studied separately. Path Planning was mentioned previously and some collision avoidance techniques are commented below.

2.5 Collision Avoidance

Collision avoidance problems have been studied in control theory, traffic simulation, robotics and crowd simulation. Different techniques have been proposed for collision avoidance in group and crowd simulations (MUSSE; THALMANN, 1997; REYNOLDS, 1999; SUGIYAMA; NAKAYAMA; HASEBE, 2001; LAMARCHE; DONIKIAN, 2004; HELBING et al., 2005; FOUDIL; NOUREDDINE, 2006; THALMANN, 2006; SUD et al., 2007; GOLAS; NARAIN; LIN, 2013). These are based on local coordination schemes, velocity models, prioritization

rules, force-based techniques, or adaptive roadmaps. Other techniques (YERSIN et al., 2008) have used LOD techniques to trade-off fidelity for speed. Recently, a common technique used for collision avoidance is the Velocity Obstacles (VO).

VO is the set of all velocities of a robot that will result in a collision with another robot at some moment in time, assuming that the other robot maintains its current velocity (FIORINI; SHILLERT, 1998). If the robot chooses a velocity inside the velocity obstacle then the two robots will eventually collide, if it chooses a velocity outside the velocity obstacle, such a collision is guaranteed not to occur. Figure 2.12 shows the VO of a robot A induced by another robot B.

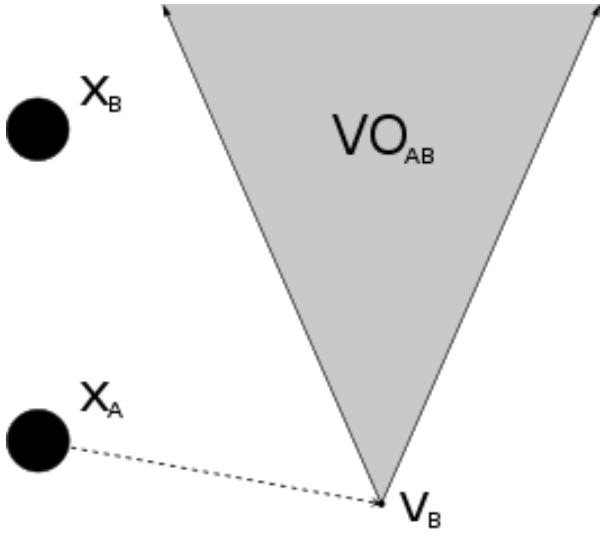


Figure 2.12 – The velocity obstacle VO_{AB} for a robot A, with the position X_A, induced by another robot B, with the position X_B and the velocity V_B.

Source: <http://en.wikipedia.org/wiki/Velocity_obstacle>. Last access: February 2015.

The VO for a robot A induced by a robot B may be formally written as

$$VO_{A|B} = \{\mathbf{v}|\exists t > 0 : (\mathbf{v} - \mathbf{v}_B)t \in D(\mathbf{x}_B - \mathbf{x}_A, r_A + r_B)\}$$

where A has position $\mathbf{x}_A$ and radius r_A, and B has position $\mathbf{x}_B$, radius r_B, and velocity $\mathbf{v}_B$. The notation $D(\mathbf{x}, r)$ represents a disc with center $\mathbf{x}$ and radius r.

The notion of VO was proposed for path planning in dynamic environments and has been extended to deal with uncertainty in sensor data (FIORINI; SHILLERT, 1998; FULGENZI; SPALANZANI; LAUGIER, 2007). Recently, Berg et al. (BERG; LIN; MANOCHA, 2008; BERG et al., 2008) extended the VO formulation for reducing collisions between agents. Berg's technique, however, relies on extensive sampling for computing collision-free velocities which

prevents fast implementations. Other extension (PARIS; PETTRE; DONIKIAN, 2007) have also been proposed. However, this technique provides higher-order path-planning with implementations that are not yet fast enough for very large simulations.

2.6 Discussion

There is a large number of methods for solving the basic path planning problem. Not all of them completely solve the problem. Despite many external differences, the methods are based on a few different general approaches, which are cell decomposition or grid-based, roadmaps, and potential fields.

Grid methods work better in small environments. When the environment becomes large, grid based methods take a large amount of computation time, and generally produce non-smooth paths. Most of the paths produced by roadmap methods are piecewise linear. They have many redundant trajectories, and little clearance to the obstacles, resulting in unnatural paths. Also, a roadmap that works for a particular agent, may not work for an agent with different size since a wider agent cannot fit into a narrow path valid for a thinner agent. Occasionally, potential fields may present the local minima problem.

Crowd simulation generally uses hybrid methods to overcome these problems. Also, they can focus on a macroscopic level, dealing with a crowd as a whole, or on a microscopic level, dealing with agents individual interactions. Our proposed technique makes use of a hybrid idea, using the grid based potential field to control agent navigation. Also, the agent's individuality is taken into account, allowing the use of our technique in a macroscopic or microscopic level.

3 CONFIGURABLE FLOWS

The classical use of potential fields often presents local minima and unnatural paths (KOREN; BORENSTEIN, 1991). A way to generate a potential field that is free from local minima and produces natural paths is through the proposed technique, that use a numerical solution of a convenient partial differential equation with boundary conditions - a boundary value problem (BVP). Boundary conditions are central to the method and indicate which regions in the environment are obstacles and which ones are targets. Hence, the obstacles' configuration influences the path followed by the agents.

3.1 Overview

Our research group has been working with potential fields based on BVP for several years. In 2001, Prestes et al. (PRESTES et al., 2001) proposed use of relaxation methods for calculation of harmonic potentials to robot exploration of unknown environments. The potential field calculated indicated safe paths towards the unexplored regions.

In 2006, Dapper et al. (DAPPER et al., 2006) proposes the use of BVP for virtual agent navigation. The proposed work allows a group of synthetic actors to move negotiating space, avoiding collisions, attaining goals in prescribed sequences while at the same time producing very individual paths. The individuality of each pedestrian could be set by changing its inner field parameters. This led to a broad range of possible agents' behaviors.

We (SILVEIRA et al., 2010a) showed a better understanding of the use of this kind of potential field, presenting some results exploring situations as steering behavior in corridors with collision avoidance and competition for a goal, and searching for objects in unknown environments. Also, a proposal to produce different behaviors for agents by automatically changing the size of the field of view of each agent. This planner was named "Configurable Flows".

Meanwhile, in 2008, we (SILVEIRA; PRESTES; NEDEL, 2008) presented a new approach to managing the movement of groups in dynamic environments using an algorithm that includes a strategy to keep formations during the displacement of the agents' group. Also, they proposed a sketch-based navigation.

Fischer et al. (FISCHER; SILVEIRA; NEDEL, 2009) presented a strategy to implement this planner using CUDA on GPU. With the GPU-based strategy, a speed up of 56 times the previous implementation was achieved, allowing its use in situations with a large number of autonomous characters, which is commonly found in games. We also implemented our planner in an RTS game engine (SILVEIRA et al., 2010b) obtaining better results than the native path planner.

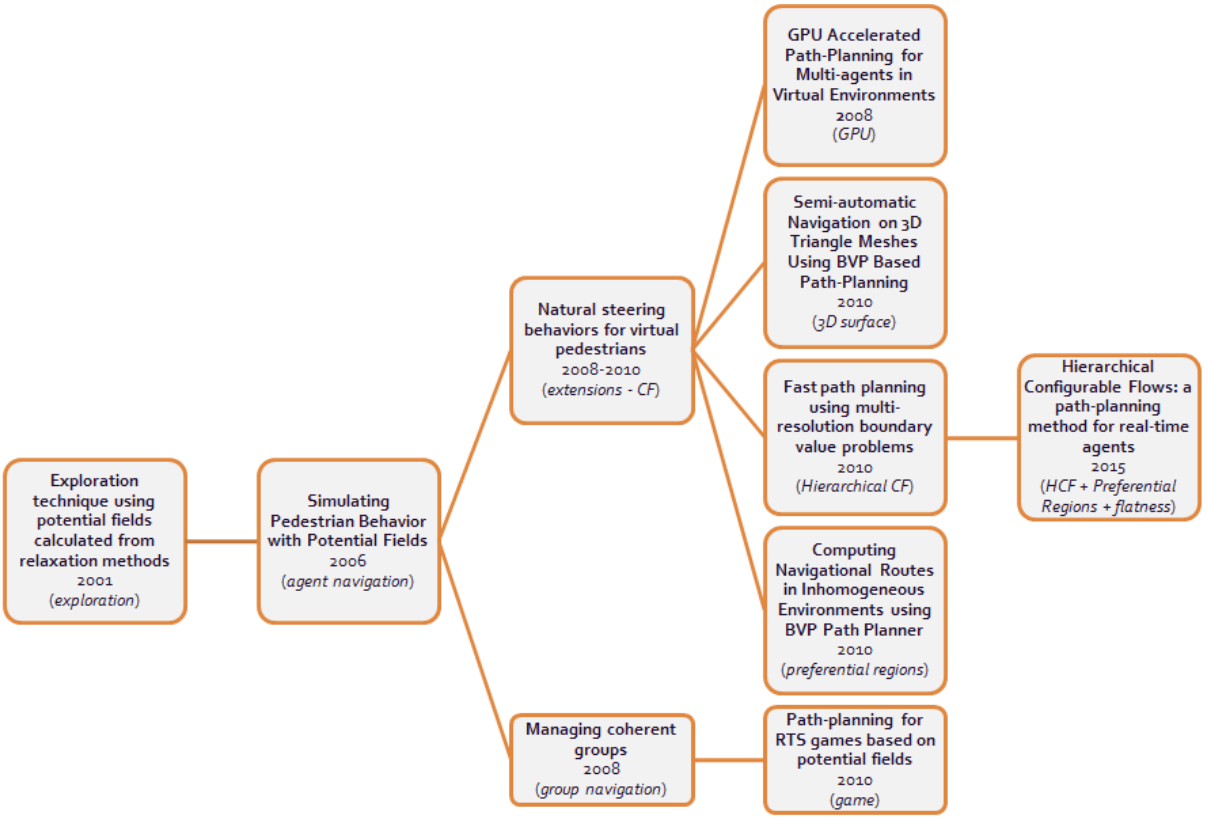


Figure 3.1 – Overview of the work on potential fields based on BVP done by our research group.

Source: Created by the author.

Inspired by these works, Fischer et al. (FISCHER; NEDEL, 2011) presented a technique for path-planning over 3D surfaces. The planner handles complex surfaces of arbitrary genus or curvature, represented by a triangle mesh, without the need of 2D parametrizations. The technique was able to generate paths on arbitrary surfaces with similar quality as those generated by BVP-based methods in planar environments.

In 2010, we proposed a hierarchical path planner, called "Hierarchical Configurable Flows", that integrates the previous planner with the Full Multigrid Method, which solves elliptic partial differential equations using a hierarchical strategy. Our new approach produced real-time performance in large environments. Results showed that our proposal spends less than 1% of the time needed to compute a solution using our original planner in several environments. This path planner is the main contribution of this thesis.

Later in 2010, Prestes et al. (PRESTES; IDIART, 2010) proposed a new form of the planner's core equation allowing it to deal with inhomogeneous environments. This technique was able to control the curvature of the potential field creating regions with higher or lower naviga-

tion preferences. Figure 3.1 shows a schematic of all these works in chronological order.

Recently, we integrate the HFC Path Planner with preferential regions, introducing a solution to the Flatness Problem for multi-agent simulation.

3.2 Background

In the literature, the first proposal on potential fields based on BVP was made by Connolly and Grupen (CONNOLLY; GRUPEN, 1993) and is called the method of the harmonic functions. In their method, the potential fields are the solutions of a boundary value problem using the Laplace's Equation,

$$\nabla^2 \, p(\boldsymbol{r}) = \frac{\partial^2 p(\boldsymbol{r})}{\partial x^2} + \frac{\partial^2 p(\boldsymbol{r})}{\partial y^2} \, = \, 0. \tag{3.1}$$

They also proposed boundary conditions such as the potential should be one in the contours of the obstacles and zero in the region of the target. Setting up the value of the function in the boundaries is called a Dirichlet boundary condition in the language of a BVP. The agent uses the gradient descent of this potential to determine the path free of obstacles that connects its current position to the target.

The solution of the Laplace's Equation 3.1, called harmonic functions, does not have local minima (CONNOLLY; BURNS; WEISS, 1990). This means there is only one minimum defined in target position and exactly one path from any point to the target. We could imagine it in an intuitive way. For each position in the environment, there is only one gradient direction. In other words, there is only one direction that will guide an agent in its goal, at each position. This method is formally complete (CONNOLLY; GRUPEN, 1993), i.e., if there is a path that connects the agent's position to the target, then it will be found. Otherwise, the method notifies the lack of a path. The resulting path is smooth, safe and minimizes the collision probability with the obstacles.

Recently, Trevisan et al. (TREVISAN et al., 2006) came up with a path planner for agent navigation based on a family of potential field functions that do not have local minima. This family is generated through the numeric solution of a BVP using Dirichlet boundary conditions and the following equation

$$\nabla^2 \, p(\boldsymbol{r}) + \epsilon \boldsymbol{v}.\nabla p(\boldsymbol{r}) \, = \, 0 \tag{3.2}$$

where $\boldsymbol{v}$ is a bias unity vector and ϵ is a scalar value. In navigation tasks, the use of terms ϵ and $\boldsymbol{v}$ produce different behaviors for humanoids. They distort the potential field providing a

preferred direction to be followed, illustrated through the path followed by each agent.

When $\epsilon = 0$, Equation 3.2 reduces to $\nabla^2\ p(\boldsymbol{r}) = 0$ which corresponds to Laplace's Equation 3.1. This equation is used as the core of the path planner based on harmonic function developed by Connolly and Grupen (CONNOLLY; GRUPEN, 1993), as stated before. This planner produces paths that minimize the collision probability of the agent with obstacles, i.e., in an indoor environment the agent will tend to follow a path that is equidistant to the walls. This behavior is not always adequate to simulate humanoid motion since it looks very stereotyped because humans do not always walk equidistant to the walls.

To generate realistic steering behaviors, we need to conveniently adjust both parameters ϵ and $\boldsymbol{v}$. Vector $\boldsymbol{v}$, called *behavior vector*, can be thought of as an external force that pulls the agent to its direction, whenever possible, whereas the parameter ϵ can be understood as the *strength* or *influence* of this vector on the agent behavior. The allowed values of parameters ϵ and $\boldsymbol{v}$ permit to generate an expressive amount of action sequences – *displacement sequences* – that virtual humanoids can use to reach a specific target position. Figure 3.2 shows three different paths followed by the agents using Equation 3.2 and changing the parameters ϵ and $\boldsymbol{v}$.

(a) (b) (c)

Figure 3.2 – Different paths followed by agents using Equation 3.2. Paths produced using $\epsilon = 1$, $\boldsymbol{v} = (0, 0)$ (a), $\epsilon = 1$, $\boldsymbol{v} = (1, 0)$ (b) and $\epsilon = 1$, $\boldsymbol{v} = (1, sin(0.6t))$ (c).

Source: Created by the author.

After the potential field is computed, the agent moves from its current position in the direction of the gradient descent of this potential. This process is an intuitive way of controlling the agent's motion. However, it can easily fail to produce realistic steering behaviors, as those observed in the real world. One of the reasons is that the agent changes its direction based solely on the gradient descent of its position. For instance, if the field of view of the agent is small, its reaction time will be very short to treat dynamic obstacles. Then, those obstacles will produce a strong repulsion force that will change the agent's direction abruptly. To handle this problem, we use this strategy as a global planner while agents use a local planner, described in Section 3.4, to generate better quality paths.

3.3 Global Path Plannner

To solve the global path planner numerically, we can consider that the solution space is discretized into a regular grid, as illustrated in Figure 3.3. Each cell (i, j) is associated to a squared region of the real environment and stores a potential value $p(i, j)$. Using the Dirichlet boundary conditions, the cells associated with obstacles in the real environment store a potential value of 1 (*high potential*) whereas those containing the target store a potential value of 0 (*low potential*). A high potential value prevents the agent from running into obstacles whereas a low value generates an attraction basin that pulls the agent.

We can write Equation 3.2 using the approximation to the first and second derivative. Equation 3.2 becomes:

$$\frac{p_r - 2p_c + p_l}{h^2} + \frac{p_u - 2p_c + p_b}{h^2} + \epsilon \left[v_x \left(\frac{p_r - p_l}{2h} \right) + v_y \left(\frac{p_t - p_b}{2h} \right) \right] = 0$$

which implies that the potential in each cell is updated by an iterative method according to:

$$p_c = \frac{p_r + p_l + p_u + p_b}{4} + \frac{\epsilon h}{8} \left[v_x \left(p_r - p_l \right) + v_y \left(p_t - p_b \right) \right] \tag{3.3}$$

where $\boldsymbol{v} = (v_x, v_y)$, h is the cell width and p_r, p_l, p_t, p_b are illustrated in Figure 3.4.

Equation 3.3 computes the potential value at the center of each cell. That equation depends on the value of neighbor cells, that also must be computed. Enforcing the equation at the center of every cell generates a linear system of equations that need to be solved numerically. For that, we can employ a relaxation method, which is an iterative method for solving systems of equations. The relaxation methods usually employed to compute the potentials of free space cells are Gauss-Seidel (GS) and Successive Over-Relaxation (SOR).

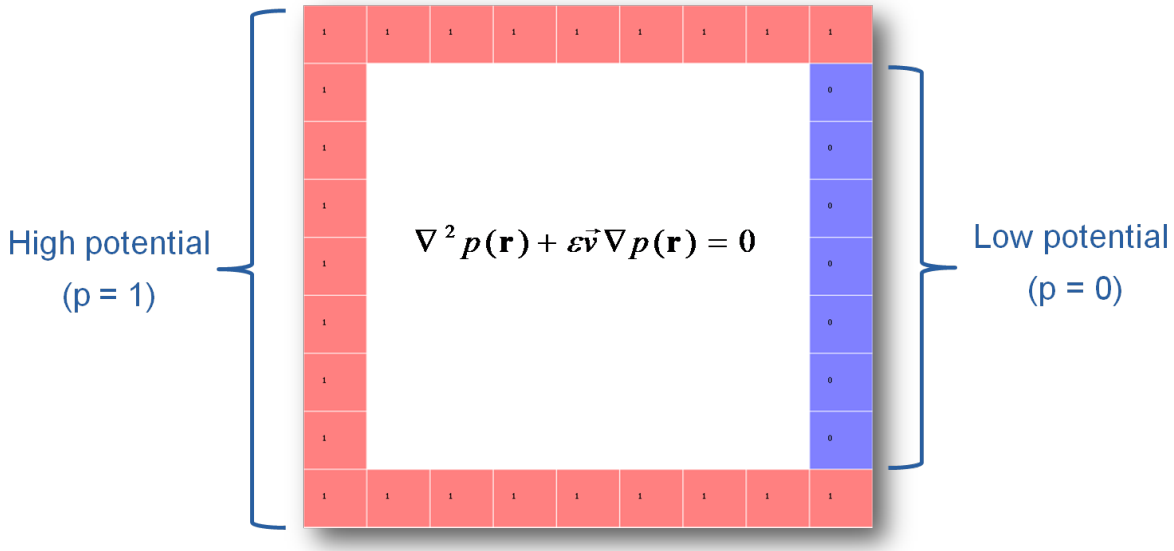


Figure 3.3 – Configurable Flows. To solve the global path planner, the potential should be one in the contours of the obstacles (red cells) and zero in the region of the target (blue cells).

Source: Created by the author.

The GS method updates the potential of a cell c through Equation 3.3, as follow:

$$p_c^{(k)} = \frac{p_b + p_t + p_r + p_l}{4} + \frac{\epsilon h((p_r - p_l)v_x + (p_t - p_b)v_y)}{8} \tag{3.4}$$

SOR is an extension of the GS method with a convergence accelerator factor. However, the error produced by SOR often grows before the convergence sets in, resulting in oscillatory potential fields during the calculation. On the other hand, the error produced by the GS method monotonically decays during the computation of the potential. This makes the GS more useful in tasks like agent exploration and navigation since the agent can use partial results as an approximation of the potential field (PRESTES et al., 2001).

The SOR method updates the potential of a cell c through:

$$p_c^{(k)} = p_c^{(k-1)} + \omega \left(\frac{p_b + p_t + p_r + p_l - 4p_c^{(k-1)}}{4} + \frac{\epsilon h[(p_r - p_l)v_x + (p_t - p_b)v_y]}{8} \right) \tag{3.5}$$

where $\omega = \frac{4}{2+\sqrt{4-c^2}}$ with $c = \cos\frac{\pi}{m} + \cos\frac{\pi}{n}$ where m and n are the grid dimensions. Equation 3.3 assumes that $h_x = h_y = h$, which implies that m and n must be a multiple of h.

After computing the potential field, the agent at position (i, j) will be guided by the gradient

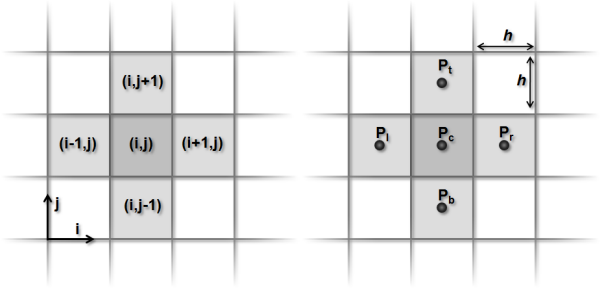


Figure 3.4 – Representation of p_c, p_b, p_t, p_r and p_l on the grid.

Source: Created by the author.

descent at its position, which is calculated by the following equation:

$$-\nabla \mathbf{p}_{(i,j)} = \left(\frac{p_{i-1,j} - p_{i+1,j}}{2h}, \frac{p_{i,j-1} - p_{i,j+1}}{2h} \right). \tag{3.6}$$

3.4 Local Path Planner

To avoid the collisions between the agents and to handle dynamic obstacles we propose, for each agent, one map that stores the current local information about the environment obtained from the agent's field of view. This local map is centered on the current agent position and represents a small fraction of the global map, as illustrated in Figure 3.5.

3.4.1 The Agent's Local Map

Each agent a_k has one map m_k that stores the current local information about the environment obtained by its own sensors. This map is centered at the agent's current position and represents a small fraction of the global map, usually about 10% of the total area covered by the global map. The map m_k has $l_x^k \times l_y^k$ cells, denoted by $\{c_{i,j}^k\}$ and is divided into three regions: the update zone (*u-zone*); the free zone (*f-zone*) and the border zone (*b-zone*), as shown in Figure 3.6. Each cell corresponds to a squared region centered at the environment coordinates $r = (r_i, r_j)$ and stores a particular potential value $p_{i,j}^k$.

The area associated with the cell of an agent's map is smaller than the area associated with the cell of the global map. The main reason is that the agent map is used to produce refined motion while the global map is used only to assist the long-term agent navigation. Hence, the smaller the size of the cell of the local maps, the better the quality of motion.

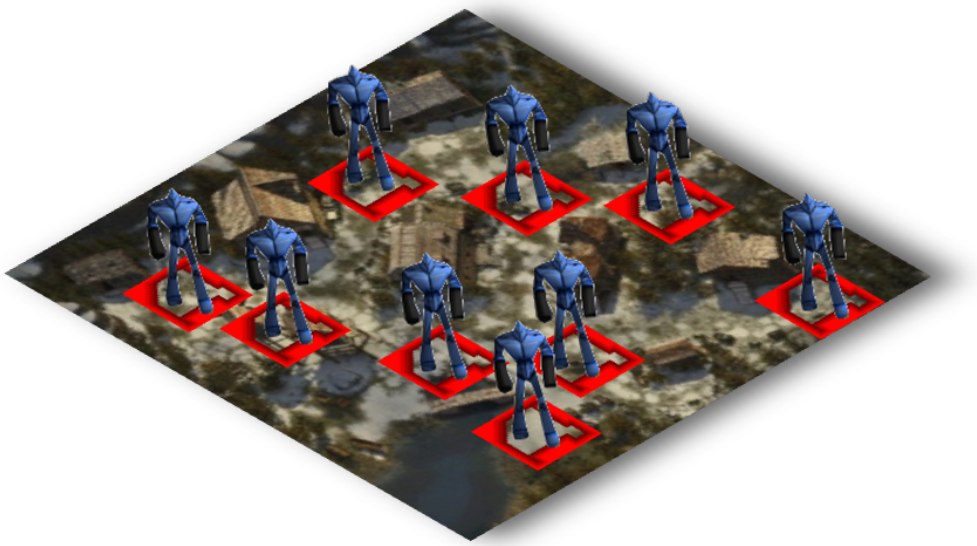


Figure 3.5 – Agents in an environment and their local maps.

Source: Created by the author.

3.4.2 Updating Local Maps from Global Maps

For each agent, a goal, a particular vector $\boldsymbol{v}$ that controls its behavior, and a ϵ should be stated. The same goal, $\boldsymbol{v}$ and ϵ can be designated to several agents. If $\mathbf{v}_k$ or ϵ_k is dynamic, then the function that controls it must also be specified.

To navigate through the environment, an agent a_k uses its sensors to perceive the world and to update its local map with information about obstacles and other agents. The agent's sensor sets a view cone with aperture α. The *u-zone* cells $c^k_{i,j}$ that are inside the view cone and correspond to obstacles or other agents have their potential value set to 1. Agents inside the *u-zone* but out of the agent's view cone is not mapped as an obstacle into the local map. This procedure assures that dynamic or static obstacles behind the agent do not interfere with its future motion.

For each agent a_k, the global gradient descent on the cell in the global map, provided by the global planner, that contains its current position is calculated. The gradient direction is used to generate an intermediate goal in the border of the local map, setting the potential values to 0 of a couple of *b-zone* cells, while the other *b-zone* cells are considered as obstacles, with their potential values set to 1. As the agent local map is delimited by obstacles, the agent is pulled towards the intermediate goal using the direction of its local gradient. The intermediate goal

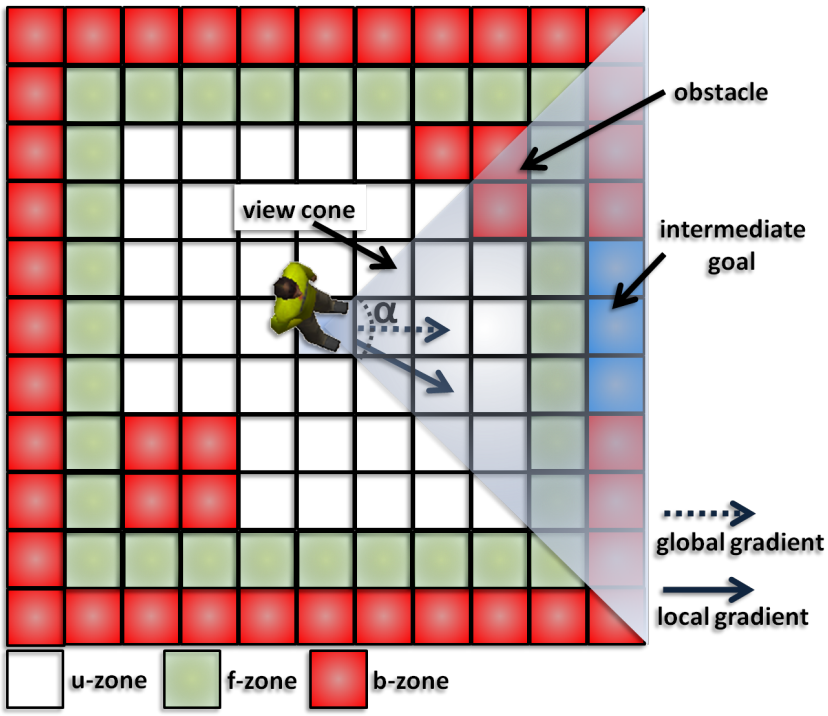


Figure 3.6 – Agent Local Map. White, green and red cells comprise the *update*, *free* and *border zones*, respectively. Blue and red cells correspond to the intermediate goal and obstacles respectively.

Source: Created by the author.

helps the agent a_k to reach its target while allowing it to produce a particular motion.

In some cases, the goal is inside both the view cone and the *u-zone*, and consequently, local map cells associated are set to 0. The intermediate goal is always projected, even if the target is mapped onto the *u-zone*. *F-zone* cells are always considered free of obstacles, even when there are obstacles inside. The absence of this zone may close the connection between the current agent cell and the intermediate goal due to the mapping of obstacles in front of the intermediate goal. When this occurs, the agent gets lost because there is no information coming from the intermediate goal to produce a path to reach it. *F-zone* cells handle that situation, always allowing the propagation of the information about the goal cells to the cells associated with the agent's position.

3.4.3 Motion Generation

After the mapping steps, the agent computes the potential value of its map cells with its pair $\boldsymbol{v}^k$ and ϵ^k. After computing the potential, the agent adjusts its current position by

$$\Delta\ \boldsymbol{d} = \upsilon\ (\cos(\varphi^t), \sin(\varphi^t))\ \Psi(|\varphi^{t-1} - \zeta^t|) \tag{3.7}$$

where function $\Psi : \Re \rightarrow \Re$ is

$$\Psi(x) = \begin{cases} 0 & \text{if } x > \pi/2 \\ \cos(x) & \text{otherwise} \end{cases} \tag{3.8}$$

and ζ is the orientation of the gradient descent computed from the potential field stored in its local map in the central position $(\lceil l_x^k/2 \rceil, \lceil l_y^k/2 \rceil)$ and φ is the agent's direction, which is updated as follows:

$$\varphi^t = \eta\ \varphi^{t-1} + (1 - \eta)\ \zeta^t \tag{3.9}$$

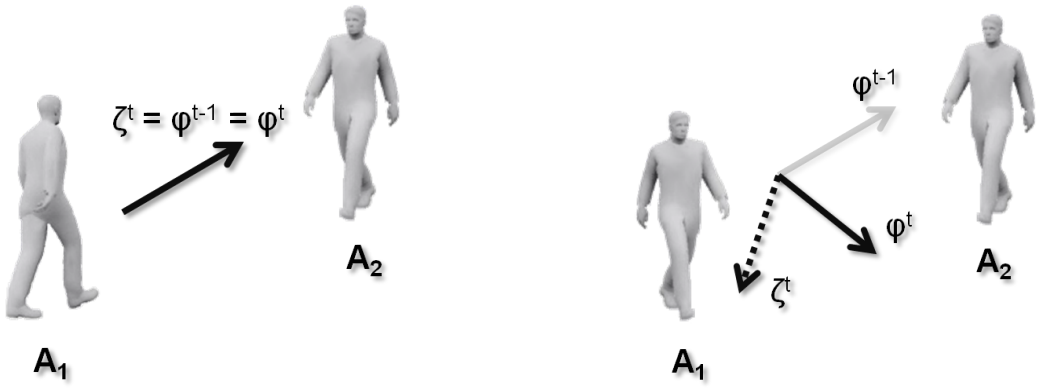
where $\eta \in [0, 1)$.

If $|\varphi^{t-1} - \zeta^t|$ is higher than $\pi/2$, then there is a high hitting probability and this function $\Psi(.)$ returns the value 0, making the agent stop. Otherwise, the agent's speed will change proportionally to the collision risk. In regions cluttered with obstacles, agents will tend to move slowly. If a given agent is about to cross the path of another, one of them will stop and wait until the other get through. When $\eta = 0$, the agent adjusts its orientation using only information about the gradient descent. Figure 3.7(a) shows a particular situation where an agent is walking in a straight line, when he faces another agent. If $\eta = 0.5$, the previous agent's direction (φ^{t-1}) and the gradient descent direction influence equally the computation of the new agent's direction. Figure 3.7(b) shows the agent orientation φ^t computed with $\eta = 0.5$.

The parameter η can be viewed as an inertial factor that tends to keep the agent direction constant insofar $\eta \rightarrow 1$. When $\eta \rightarrow 1$, the agent reacts slowly to unexpected events, increasing its hitting probability with obstacles. η is a flexible parameter that the user is able to control. However, a learning strategy could be used to specify what is the best η to a specific situation.

3.5 Algorithm

The algorithm that implements the concepts shown before and produces the humanoids' movement is presented in Algorithm 1.



(a) (b)

Figure 3.7 – Defining agent motion. (a) Situation before the agent A_2 enters in the field of view of A_1. (b) If the agent A_1 follows the direction defined by the gradient descent (ζ), it will change its direction in nearly $\pi/2$, what is undesirable. However, if the agent uses the orientation φ, it will achieve a smooth curve, which is more natural and realistic.

Source: Created by the author.

The first two steps are performed in a pre-processing phase. In relation to the second loop, each agent can execute independent and asynchronously the actions from 6 to 11. This algorithm considers that each agent must reach only one target. However, the agent is able to reach several targets orderly. In this case, the step 11 must be changed to Algorithm 2.

3.6 Preferential Regions

Robust path planners should be able to deal with inhomogeneous terrains. It is a very common situation in the real world. Imagine an inhomogeneous terrain with different type of preference or different navigation capabilities, just like the one illustrated in Figure 3.8. Each different terrain has an associated preference that indicates the difficulty of traveling, i.e., the difficulty to navigate in grassland is smaller than the difficulty to navigate on water. However, a diver can navigate easily in water than in grassland. This problem is commonly referred to as weighted region problem (MITCHELL; PAPADIMITRIOU, 1991; MITCHELL; PAPADIMITRIOU, 1987).

Using Equation 3.2, a certain but limited freedom to redesign the paths can be obtained by changing the boundary conditions from Dirichlet to Neumann. Dirichlet boundary condition specifies the values a solution needs to take on the boundary of the domain, while Neumann boundary condition specifies the values that the derivative of a solution has to take on the boundary of the domain. In our context, the domain is the environment.

Algorithm 1 Configurable Flows

1: Compute all the environment's global maps.{*one for each possible goal* o_k}
2: **for** each agent a_k **do**
3: Define the behavior vector $\mathbf{v}_k$ and ϵ_k {*Each variable can be either static or dynamic. If a variable is dynamic, then the function that controls it must be specified*}
4: **end for**
5: **for** each agent a_k **do**
6: Read its sensors to detect static and dynamic obstacles
7: Update its map with local information about obstacles and other agents
8: Compute the global gradient descent and generate the intermediate goal
9: Update the potential field
10: Compute the local gradient descent and follow the gradient direction according to Equation 3.7
11: **while** not reaching the target $o_{goal(k)}$ **do**
12: Repeat steps 6 to 11
13: **end while**
14: Stop moving
15: **end for**

Algorithm 2

while not reaching the target $o_{goal^i(k)}$ **do**
Repeat steps 6 to 11
end while
if $goal^i(k) = goal^{last}(k)$ **then**
Stop moving
else
Repeat the process with the next target $o_{goal^{i+1}(k)}$
end if

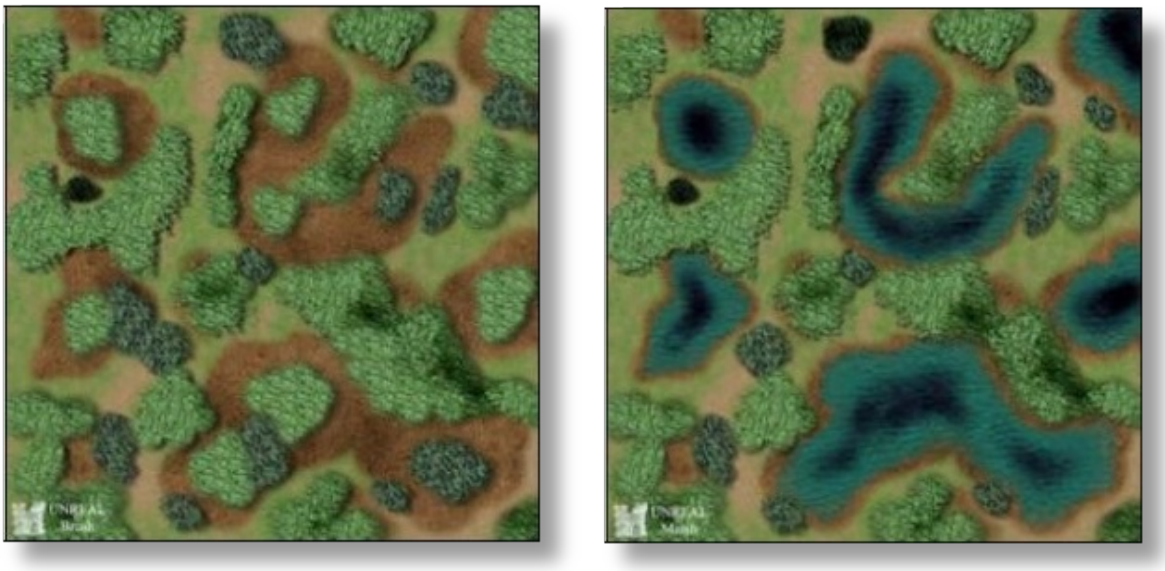


Figure 3.8 – Terrain with inhomogeneous navigation capabilities. A diver would prefer to navigate on water while ordinary agents would prefer to navigate through grassland. Also, agents may want to navigate through the road, without stepping on the grass.

Source: <http://www.matakishi.com/printedsceneryreview.htm>. Last access: February 2015.

In fact, Equation 3.2 also allows us to slightly distort the potential field without changing the boundary conditions while keeping the main property of our potential field: the lack of local minima. The parameters $\boldsymbol{v}$ and ϵ are used to control the potential field distortion. However, we need to conveniently adjust both parameters to obtain the desired behavior, and although we specify a direction, we do not have complete control of the distortion. Techniques that let the user interact and distort the potential field usually suffer from local minima. For instance, Jin's work (JIN et al., 2008), where users specify some directions, and a potential field is generated through an interpolation using Radial Basis Function.

3.6.1 Configuring the Flows

Recently, our research group refine Equation 3.2 by introducing a novel form (PRESTES; IDIART, 2009; PRESTES; IDIART, 2010) that can be easily applied to environments composed of inhomogeneous terrains with different types of preferences. The preference can be defined using the terrain type, the terrain elevation, or any other property. The "trick" is to keep $\boldsymbol{v}$ parallel with the streamlines of the potential represented by $\nabla p(\boldsymbol{r})$. An important observation is that the curvature of the potential is dependent on the angle $\theta(\boldsymbol{r})$ that $\boldsymbol{v}$ makes with the streamlines of the potential, represented by its gradient $\nabla p(\boldsymbol{r})$. By choosing a $\boldsymbol{v}$ parallel to

the gradient, i.e., $\theta(\boldsymbol{r}) = 0$, ϵ becomes a configurable parameter, which allows us to sculpt the potential field. Increasing a positive ϵ makes the potential more concave and with a faster ascent. Thus, Equation 3.2 becomes:

$$\nabla^2 p(\boldsymbol{r}) = \epsilon(\boldsymbol{r}) \left|\nabla p(\boldsymbol{r})\right|. \tag{3.10}$$

The factor $|\nabla p(\boldsymbol{r})|$ is necessary to avoid local minima since the concavity/convexity diminishes or increases proportionally to the variation of the potential. Because the computation of $|\nabla p(\boldsymbol{r})|$ is numerically time consuming (we should know the exact solution $p(\boldsymbol{r})$, i.e., we have to compute the harmonic potential), we use the triangular inequality,

$$\nabla p(\boldsymbol{r}) \leq \left|\frac{\partial p(\boldsymbol{r})}{\partial x}\right| + \left|\frac{\partial p(\boldsymbol{r})}{\partial y}\right| \tag{3.11}$$

to obtain a more efficient equation,

$$\nabla^2 p(\boldsymbol{r}) = \epsilon(\boldsymbol{r}) \left(\left|\frac{\partial p(\boldsymbol{r})}{\partial x}\right| + \left|\frac{\partial p(\boldsymbol{r})}{\partial y}\right|\right). \tag{3.12}$$

The configurable function $\epsilon(\boldsymbol{r})$ changes the curvature of the potential and changing the curvature, we can manipulate the region traversing preferences. A low preference region can be created by locally increasing its potential convexity, giving rise to an effective attractive force that pulls the trajectories towards the path. Higher preferences are linked to higher concavity, and expel agents from the distorted region. The smaller the ϵ, the more the agent will avoid a region. The higher the ϵ, the more the agent will be attracted to a region. In order to illustrate it, Fig. 3.9 shows a set of navigational paths. We used $\epsilon = 1$ to create high preference regions (the light-blue regions on Fig. 3.9(a)), and $\epsilon = -1$ to create low preference regions (the light-red regions on Fig. 3.9(c)). Fig. 3.9(c) presents an environment with obstacles and preferential regions, with $\epsilon = -0.5$ and $\epsilon = 0.5$, respectively. Observe that an essential difference between a low preference region and an obstacle is that low preference regions can be crossed, if necessary. But obstacles will never be crossed.

Fig. 3.9(b) and (d) show how these preferential regions sculpts the potential field. We can configure the navigational preference on the global planner, or in the local planner, producing very individual paths for the agents. The parameter ϵ can also be configurable in real time. It enables the planner to deal with unexpected situations as a sudden landslide, that forces some of the agents to avoid this region. If an agent must follow any specific path, we can configure the potential to produce this path, locally changing its curvature. As mentioned, to avoid local

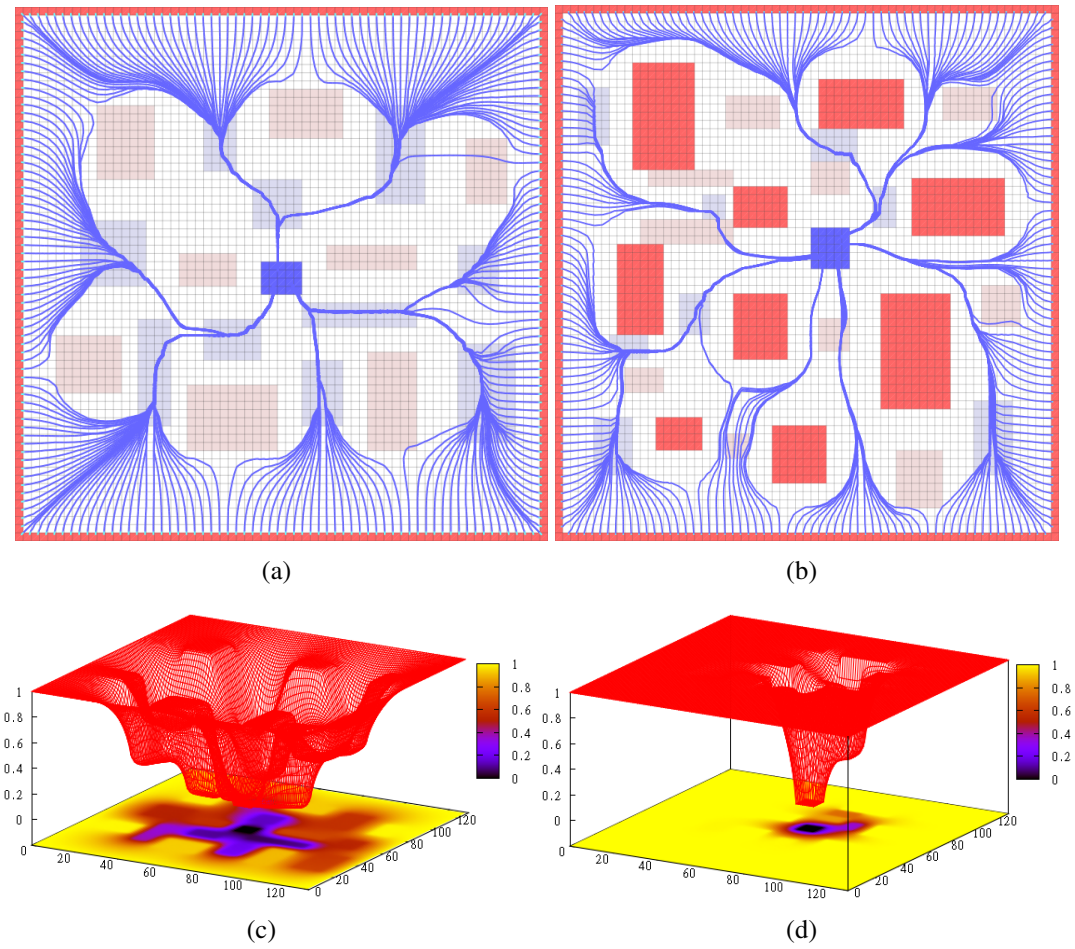


Figure 3.9 – Paths on heterogeneous terrains. The potential field flow can be configured, handling different traversal preferences. The light-blue and the light-red areas represent regions with high and low preferences, respectively. The dark blue area in the middle of the terrain represents the target position for all agents that are placed along the borders.

Source: Created by the author.

minima $|\epsilon| < 2$.

Using the ability to deal with the inhomogeneous terrain facilitates the integration of the path planning stage with terrain reasoning (STERREN, 2001), a making the planner more robust and creating broad application possibilities. The integration with a hierarchical method enables the real-time performance, allowing dynamic changes in the regions' preference.

3.6.2 Paths Following Terrain Elevation

An interesting application is to generate paths trying to preserve the terrain elevation. To be able to do it, we must change ϵ_i appropriately based on the terrain elevation h_i at grid cell

c_i, where $h_{min} \leq h_i \leq h_{max}$. To specify how hard the potential should keep the same terrain elevation during the path generation, one must to define a lower ϵ_{min} value and a higher ϵ_{max} value, where $-2 \leq \epsilon_{min} \leq \epsilon_{max} \leq 2$. Each cell c_i will have its ϵ_i value updated according to:

$$\epsilon_i = \left(\frac{h_i - h_{min}}{h_{max} - h_{min}} \right) \times (\epsilon_{max} - \epsilon_{min}) + \epsilon_{min}. \tag{3.13}$$

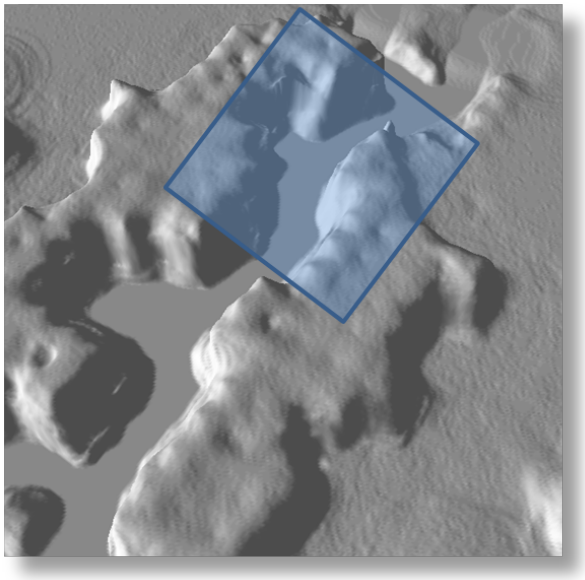
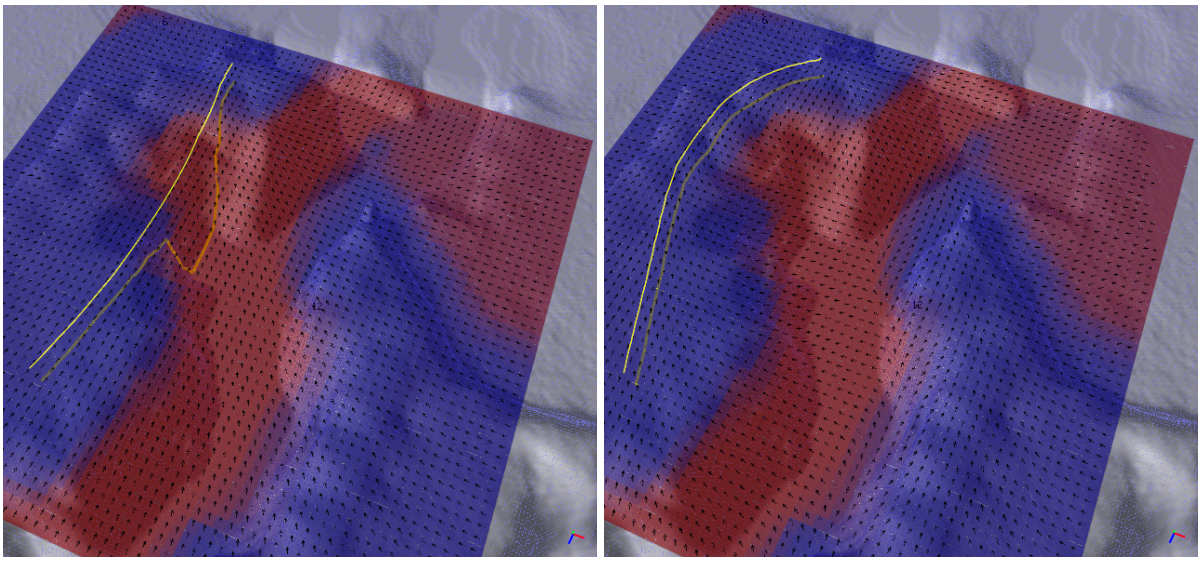


Figure 3.10 – Terrain. The blue square is a region where the planner will act, producing paths following the terrain elevation.

Source: Created by the author.

Figure 3.10 and 3.11 shows this situation. Figure 3.10 shows a terrain and a square region where the planner will act. Figure 3.11 highlight this region. To illustrate the terrain elevation, each cell in this region is colored based on its difference of height from the cell which represents the start path position. The intensity of blue represents a similar height and the red intensity represents the difference of heights. The path starts at one extreme point of the yellow curve and ends in the another extreme point. Figure 3.11(a) shows a path produced by the planner with $\epsilon_i = 0$. The terrain elevation is then ignored during the path generation. Figure 3.11(b) shows the path produced updating ϵ_i correctly, according to Equation 3.13. The path produced keep the terrain elevation, bypassing the fall.



(a) (b)

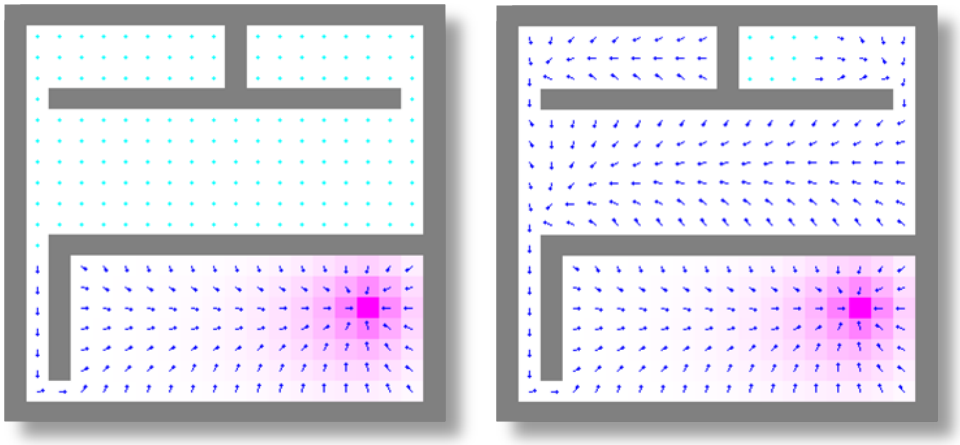
Figure 3.11 – Path following the terrain elevation. The terrain elevation is ignored during the path generation, $\epsilon_i = 0$ (a) and the path produced updating ϵ_i correctly, according to Equation 3.13 (b).

Source: Created by the author.

3.7 The Flatness Problem

Despite not having local minima, planners based on BVP (TREVISAN et al., 2006) present an implementation problem due to the finite numerical precision of the computers, that we call here the *flatness problem*. In certain regions, due to the exponential nature of the solution, the potential is very close to 1 in regions separated from the goal position by narrow passages, so the gradient is no longer correctly calculated, i.e., the potential is rounded to 1. Figure 3.12 shows the potential field computed using single-precision floating-point (32 bits) and double-precision floating-point precision. The vector gradient has length equals to zero in the regions represented by the light-blue dots.

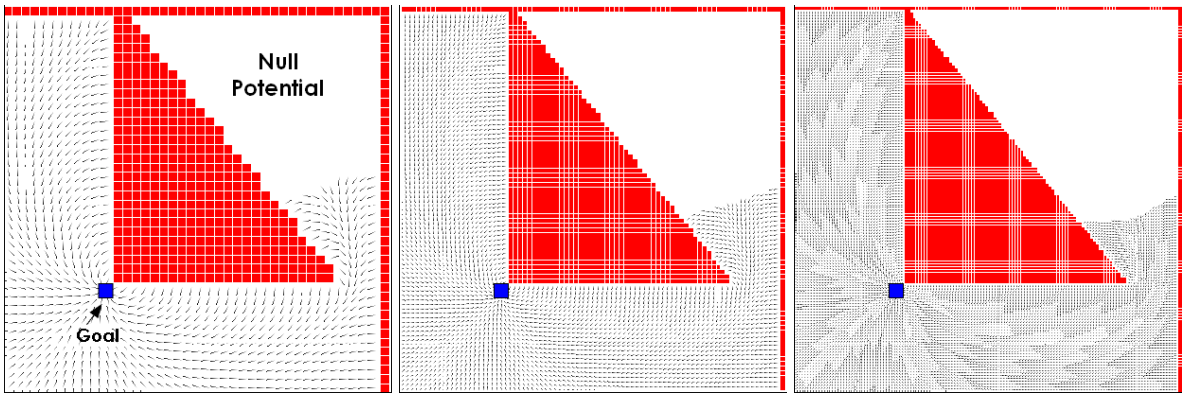
The flatness problem does not depend on the grid resolution, but the environment geometry. Fig. 3.13 shows a situation in which an environment is discretized using grids with different resolutions. Regions with null gradient due to the flatness effect are unwanted because they prevent agents located there from moving.



(a) Single-precision floating-point (32 bits) (b) Double-precision floating-point (64 bits)

Figure 3.12 – Potential flatness due to floating-point precision.

Source: Created by the author.



(a) (b) (c)

Figure 3.13 – Effect of flatness with different resolutions: (a) 60×60, (b) 120×120 and (c) 200×200.

Source: Created by the author.

4 HIERARCHICAL CONFIGURABLE FLOWS

A solution to improve the performance of path planners is to reduce the complexity of the problem by means of a hierarchical approach (PAI; REISSELL, 1998; WU; LEE; TSAI, 1997; HYUN; SUH, 1995; JUNG; RATTI; TSIOTRAS, 2009). The environment is represented in different resolutions and used as input to the path planners, that will decide which resolution is better for a given situation, using criteria such as time restriction or memory limitations.

A common hierarchical representation is the quadtree decomposition (NOLBORIO; NANIWA; ARIMOTO, 1990), where the environment is first represented using a coarse grid which is then subdivided generating patches with different resolutions. After that, any graph search algorithm may be used to obtain the path.

Graph search algorithms used for generate paths, as A^* or D^*, mentioned in a previous chapter, also have hierarchical versions, called Hierarchical A^* (HARABOR; BOTEA, 2008b) and Hierarchical D^* (CAGIGAS, 2005). Another way to generate paths hierarchically is using wavelet functions for the environment decomposition (TSIOTRAS; BAKOLAS, 2007). The wavelet transform is a very fast approach that allows the decomposition of the environment at different levels of resolution. The smooth path is achieved using the information provided by the coefficients in the wavelet expansion.

4.1 Hierarchical Configurable Flows

We propose a new method – so called *Hierarchical Configurable Flows*, or HCF – whose core is based on the *Full Multigrid Method* (FMG), proposed by Brandt (BRANDT, 1977). The FMG method is a technique to develop efficient solvers. This method solves elliptic partial differential equations through a combination of solutions at several resolution levels. Basically, it takes an instance of the problem on a grid of pre-specified fineness and generates coarser grids containing a cruder problem representation. The method solves the problem on the coarsest grid, which is easy and cheaper, and obtains successive solutions on finer and finer grids.

In the HCF algorithm, the entire environment is represented by a hierarchy of homogeneous meshes $\{\mathcal{M}_k\}$, where each mesh $\mathcal{M}_k$ has $L_x^k \times L_y^k$ cells, denoted by $\{c_{i,j}^k\}$. Each cell $c_{i,j}^k$ corresponds to a squared region centered at the environment coordinates $r = (r_i, r_j)$ and stores a particular potential value $p_{i,j}^k$, a configurable parameter $\epsilon_{i,j}^k$ and an error $e_{i,j}^k$. Fig. 4.1 shows the hierarchy of grids. Potential fields in all grids are computed using the FMG method.

In the remaining of this chapter we present the *Full Multigrid Method*, the *Hierarchical Configurable Flows* algorithm, and how should one proceed to use the algorithm efficiently.

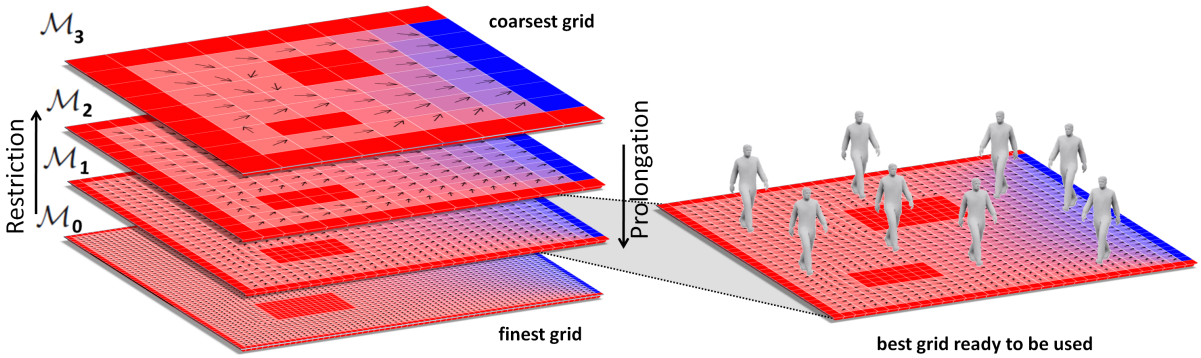


Figure 4.1 – Hierarchical environment representation. The environment is represented by a hierarchy of 4 grids $\mathcal{M}_k$ with different resolution. Red and blue cells correspond to obstacles and goals, respectively, while the arrows illustrate the vector field.

Source: Created by the ahtor.

4.2 Multigrid Methods for Boundary Value Problems

Multigrid methods are usually considered the fastest numerical methods for developing solvers for elliptic partial differential equations. In addition, they are among the fastest methods for solving many other problems, like other types of partial differential equations or integral equations. They are a family of methods for solving linear systems $\boldsymbol{Ax} = \boldsymbol{f}$ that arise in the process to solve discretized computational models. Unfortunately, there is not a single multigrid algorithm that solves all problems. Rather, there is a multigrid technique that provides the framework for solving these problems. We have to adjust the various components of the algorithm within this framework.

In this thesis, we are interested in solving variations of the Laplace's Equation 3.1 with Dirichlet boundary conditions. For simplicity, we will show how to use the multigrid method to solve Poisson's Equation which is a more general case of Laplace's Equation, with the right-hand side of the equation equals to a function f. We are seeking a function $p(\boldsymbol{r})$ that satisfies the following system:

$$\begin{cases} \nabla^2\, p(\boldsymbol{r}) = \; f, & \boldsymbol{r} \in \Omega \\ p(\boldsymbol{r}) = 1, & \boldsymbol{r} \in \partial\Omega \\ p(\boldsymbol{r}) = 0, & \text{if } \boldsymbol{r} \in \Omega \text{ and } \boldsymbol{r} \text{ is a goal position.} \end{cases} \tag{4.1}$$

where Ω represents an open set in plane $\Re^2$ and $\partial\Omega$ represents the set's boundary. For conve-

nience, we consider Ω as the rectangle $(0, L_x) \times (0, L_y)$.

4.2.1 Problem Discretization

To computationally solve Equation 4.1, we turn it into a system of linear equations whose solution is an approximation of the solution $p(\boldsymbol{r})$ which we can obtain through numerical methods (i.e., Jacobi, Gauss-Seidel, SOR). Instead of working with the continuous domain Ω, we consider only a finite set of points in Ω. Since by convenience we are dealing with a rectangular Ω, we can use the pattern illustrated in Figure 4.2. Each row has a sequence $\{0 = x_0 < x_1 < ... < x_{i-1} < x_i < x_{i+1} < ... < x_{n-1} < x_n = L_x\}$ with $n + 1$ equally spaced points. We call h the distance between two consecutive points x_i's which implies $x_i = x_{i-1} + h$ for all i from 1 to n) in the domain$[0, L_x]$. Similarly, each column has a sequence $\{0 = y_0 < y_1 < ... < y_{i-1} < y_i < y_{i+1} < ... < y_{n-1} < y_n = L_y\}$ with $n + 1$ equally spaced points with distance h between two consecutive points. This set is called a *uniform grid* with mesh size h on $(0, L_x) \times (0, L_y)$.

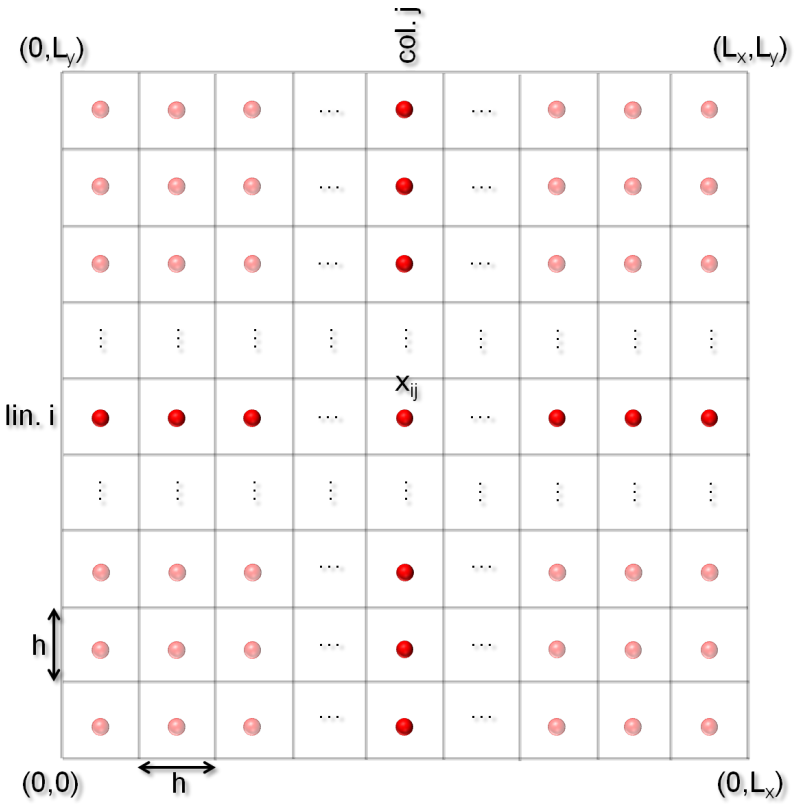


Figure 4.2 – Grid discretization. Uniform grid $(0, L_x) \times (0, L_y)$ with mesh size h.

Source: Created by the ahtor.

Using Taylor series, we obtain approximations of values that $\frac{\partial^2 p(\boldsymbol{r})}{\partial x^2}$ and $\frac{\partial^2 p(\boldsymbol{r})}{\partial y^2}$ take at a point $\boldsymbol{r}_{ij} = (x_i, y_j)$ (remember that $x_i = y_j = ih$ and jh) of the grid in terms of values that p takes at a neighborhood of x_{ij}. By Taylor series we have:

$$p(x_i + h, y_j) = p(x_i, y_j) + p_x(x_i, y_j)h + \frac{1}{2}p_{xx}(x_i, y_j)h^2 + O(h^3) \tag{4.2}$$

$$p(x_i - h, y_j) = p(x_i, y_j) - p_x(x_i, y_j)h + \frac{1}{2}p_{xx}(x_i, y_j)h^2 + O(h^3) \tag{4.3}$$

and

$$p(x_i, y_j + h) = p(x_i, y_j) + p_y(x_i, y_j)h + \frac{1}{2}p_{yy}(x_i, y_j)h^2 + O(h^3) \tag{4.4}$$

$$p(x_i, y_j - h) = p(x_i, y_j) - p_y(x_i, y_j)h + \frac{1}{2}p_{yy}(x_i, y_j)h^2 + O(h^3). \tag{4.5}$$

Adding term by term Equation 4.2 with Equation 4.3 and Equation 4.4 with Equation 4.5, and putting $p_{xx}(x_i, y_j)$ and $p_{yy}(x_i, y_j)$ in evidence, we get:

$$p_{xx}(x_i, y_j) = \frac{p(x_i - h, y_j) - 2p(x_i, y_j) + p(x_i + h, y_j)}{h^2} + O(h^3) \tag{4.6}$$

$$p_{yy}(x_i, y_j) = \frac{p(x_i, y_j - h) - 2p(x_i, y_j) + p(x_i, y_j + h)}{h^2} + O(h^3) \tag{4.7}$$

Denoting, for convenience, $p(x_i, y_j) = p_{ij}$, then Equation 4.6 and Equation 4.7 lead to the formula of *five points* because the left term is approximated using five points of the grid:

$$p_{xx}(x_i, y_j) + p_{yy}(x_i, y_j) \approx \frac{1}{h^2}(p_{i-1,j} + p_{i,j-1} - 4p_{i,j} + p_{i+1,j} + p_{i,j+1}). \tag{4.8}$$

Combining this expression with the requirements of the problem (Equation 4.1 and denoting $f(x_i, y_j) = f_{ij}$, we have:

$$\begin{cases} \frac{1}{h^2}(p_{0,1} + p_{1,0} - 4p_{1,1} + p_{2,1} + p_{1,2}) = f_{1,1} \\ \frac{1}{h^2}(p_{0,2} + p_{1,1} - 4p_{1,2} + p_{2,2} + p_{1,3}) = f_{1,2} \\ \vdots \\ \frac{1}{h^2}(p_{(i-1),j} + p_{i,(j-1)} - 4p_{i,j} + p_{(i+1),j} + p_{i,(j+1)}) = f_{i,j} \\ \vdots \end{cases} \tag{4.9}$$

This system of linear equations written in the matrix form will present a sparse pentadiagonal matrix, i.e., with five diagonals (around the main diagonal) having some non-zero entries and all other entries being zero. Thus, the matrix is usually represented by the stencil notation $\boldsymbol{A}^h$, which consist in having coefficients $p_{(i-1),j}$, $p_{i,(j-1)}$, $p_{i,j}$, $p_{(i+1),j}$ and $p_{i,(j+1)}$ of the five points of Equation 4.8 with the same position where the points $x_{(i-1),j}$, $x_{i,(j-1)}$, $x_{i,j}$, $x_{(i+1),j}$ and $x_{i,(j+1)}$ are on the grid. Figure 4.3 illustrates the correspondence of points on the grid with the points in the stencil notation. We can think that the discretization matrix $\boldsymbol{A}^h$ as a discretization (or approximation) of the ∇ operator.

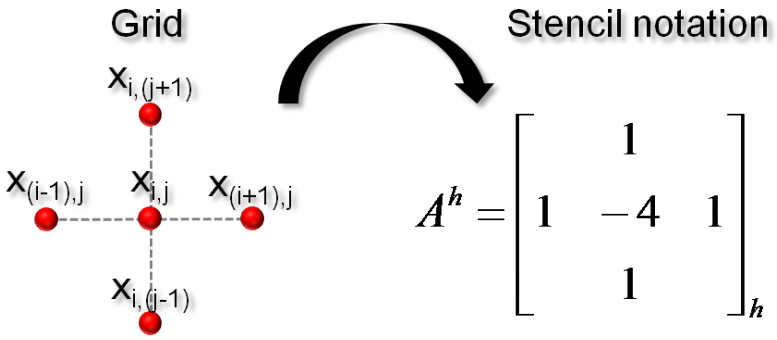


Figure 4.3 – Grid representation and stencil notation.

Source: Created by the ahtor.

With this notation, we can represent the system of equations 4.9 as:

$$\underbrace{\frac{1}{h^2}\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}_h}_{\boldsymbol{A}^h} \boldsymbol{p}^h = \boldsymbol{f}^h \tag{4.10}$$

If we ensure that the system of linear equations 4.9 has a unique solution, then we can apply iterative numerical methods such as Jacobi or Gauss-Seidel to obtain our solution. However, these methods show very slow when the discretization matrix is large, i.e., when we take many points in $[0, L_x] \times [0, L_y]$ to compose a finer grid and then obtain a discrete solution $\boldsymbol{p}^h$ closest to the continuous solution $p(\boldsymbol{r})$. The multigrid algorithm uses these properties efficiently to find the numerical solution.

4.2.2 From One-Grid, through Two-Grid to Multigrid

The key idea of the multigrid method can be understood by considering the simplest case of a two-grid method (PRESS et al., 2007). As mentioned on the previous section, we must use a grid with as many points as possible for the solution $\boldsymbol{p}^h$ of the discretized problem $\boldsymbol{A}^h\boldsymbol{p}^h = \boldsymbol{f}^h$, presented in Equation 4.9, to be a good approximation of the solution of the continuous problem $p(\boldsymbol{r})$ (Equation 4.1). However, this means that we have a large array of discretization $\boldsymbol{A}_h$, then the iterative methods to find $\boldsymbol{p}_h$ will be very slow, even though several of them have the property to quickly eliminate the high-frequency components of the initial error. Figure 4.7 illustrates the two grid representation.

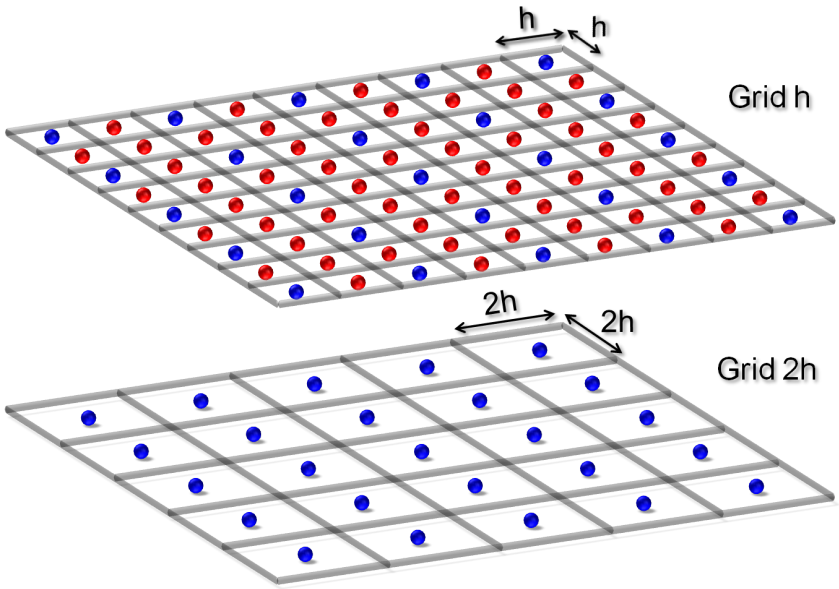


Figure 4.4 – Two-grids representation. The method makes use of grid $2h$ to quickly solve the problem in *grid h*.

Source: Created by the ahtor.

This smoothing property of iterative methods will be of great importance to the multigrid method because low-frequency components in *grid h* are high-frequency components in *grid 2h* (grid with cells two times larger than the original grid). Because an iterative smoothing method quickly eliminates high-frequency components of the initial error $[e^h]^{(0)}$ and leaves the low frequency components almost unchanged, it would be interesting that, after a few iterations to eliminate the high-frequency components, we move the problem to a subgrid with fewer points. In a subgrid with fewer points, the method is able to quickly eliminate components that were low-frequency in the original grid.

To this end, comes the *Residual Equation*. Note that the algebraical *error*

$$\boldsymbol{e}^h = \boldsymbol{p}^h - [\boldsymbol{p}^h]^{(k)} \tag{4.11}$$

of an iterated $[\boldsymbol{p}]^{(k)}$ has only theoretical importance since we do not know it (for this, we should know the exact solution $\boldsymbol{p}^h$). Thus, to measure the "distance" between the iterated $\boldsymbol{p}^{(k)}$ and the exact solution $\boldsymbol{p}^h$ we use what is called *residual*:

$$\boldsymbol{r}^h = \boldsymbol{f}^h - \boldsymbol{A}^h[\boldsymbol{p}^h]^{(k)} \tag{4.12}$$

We can see that the algebraical error $\boldsymbol{e}^h$ and the residual $\boldsymbol{r}^h$ are related. Since $\widehat{\boldsymbol{p}}^h$ is an approximate solution to Equation 4.10 and $\boldsymbol{p}^h$ is the exact solution, we can rewrite Equation 4.10 as

$$\begin{aligned}
\boldsymbol{A}^h(\underbrace{\widehat{\boldsymbol{p}}^h + \boldsymbol{e}^h}_{\boldsymbol{p}^h}) &= \boldsymbol{f}^h \\
\boldsymbol{A}^h\widehat{\boldsymbol{p}}^h + \boldsymbol{A}^h\boldsymbol{e}^h &= \boldsymbol{f}^h \\
\boldsymbol{A}^h\boldsymbol{e}^h &= \boldsymbol{f}^h - \boldsymbol{A}^h\widehat{\boldsymbol{p}}^h \\
\boldsymbol{A}^h\boldsymbol{e}^h &= \boldsymbol{r}^h
\end{aligned} \tag{4.13}$$

resulting in the *Residual Equation* 4.13.

Solve $\boldsymbol{A}^h\boldsymbol{p}^h = \boldsymbol{f}^h$ is equivalent to solve Equation 4.13 related to an iterated $\boldsymbol{p}^h$ since in both cases we need to know the exact solution $\boldsymbol{p}^h$ (in the second case we have $\boldsymbol{p}^h = [\boldsymbol{p}^h]^{(k)} + \boldsymbol{e}^h$). The process to solve $\boldsymbol{A}^h\boldsymbol{p}^h = \boldsymbol{f}^h$ through the Residual Equation can be schematically represented as

$$[\boldsymbol{p}^h]^{(k)} \rightarrow \boldsymbol{r}^h = \boldsymbol{f}^h - \boldsymbol{A}^h[\boldsymbol{p}^h]^{(k)} \rightarrow \boldsymbol{A}^h\boldsymbol{e}^h = \boldsymbol{r}^h \rightarrow \boldsymbol{p}^h = [\boldsymbol{p}^h]^{(k)} + \boldsymbol{e}^h \tag{4.14}$$

The scheme 4.14 does not present any numerical advantage. However, if we can approximate $\boldsymbol{A}^h$ by a more simple $\boldsymbol{A}^h$, then the solution of $\widehat{\boldsymbol{A}}^h\widehat{\boldsymbol{e}}^h = \boldsymbol{r}^h$ is an approximation of $\boldsymbol{e}^h$, so we can use scheme 4.14 to obtain a new iterate $[\boldsymbol{p}^h]^{(k+1)}$, resulting in the following iterative method:

$$[\boldsymbol{p}^h]^{(k)} \rightarrow \boldsymbol{r}^h = \boldsymbol{f}^h - \boldsymbol{A}^h[\boldsymbol{p}^h]^{(k)} \rightarrow \widehat{\boldsymbol{A}}^h\widehat{\boldsymbol{e}}^h = \boldsymbol{r}^h \rightarrow \boldsymbol{p}^h = [\boldsymbol{p}^h]^{(k)} + \widehat{\boldsymbol{e}}^h \tag{4.15}$$

To obtain this approximation, we can approximate $\boldsymbol{A}^h\boldsymbol{e}^h = \boldsymbol{r}^h$ through $\boldsymbol{A}^{2h}\boldsymbol{e}^{2h} = \boldsymbol{r}^{2h}$. For this, we need a way to transfer information from the *grid h* to the *grid 2h*. To convert a vector

$\boldsymbol{x}^h$ from *grid h* to a vector $\boldsymbol{x}^{2h}$ of *grid 2h*, we define the following *restriction operator*:

$$I_{h\to 2h} : \left\{ \boldsymbol{x}^{2h}_{i,j} = \boldsymbol{x}^{h}_{2i,2j}. \right. \tag{4.16}$$

Figure 4.5 illustrate the *restriction operator* that constructs the vector $\boldsymbol{x}^{2h}$ taking the components of $\boldsymbol{x}^h$ associated with the points of the *grid 2h*.

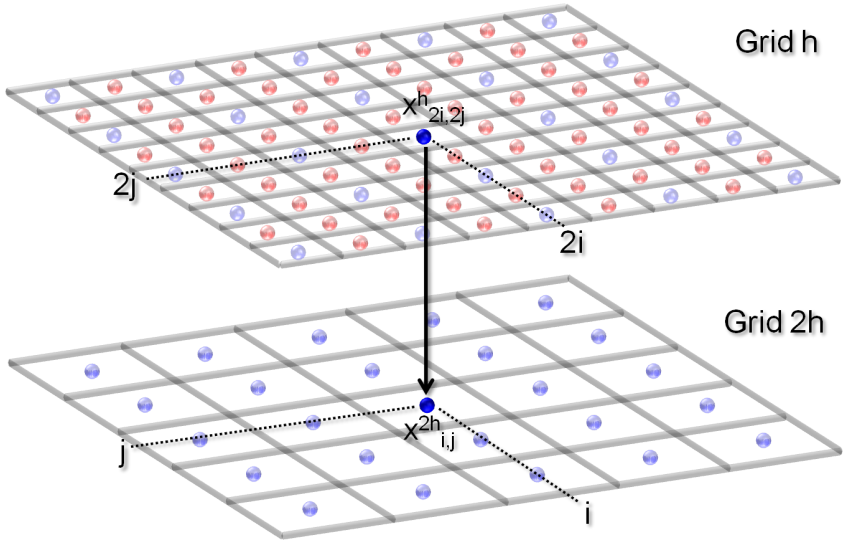


Figure 4.5 – Restriction operator. This operator constructs the vector $\boldsymbol{x}^{2h}$ taking the components of $\boldsymbol{x}^h$ associated with the points of the *grid 2h*.

Source: Created by the ahtor.

Similarly, to convert a vector $\boldsymbol{y}^{2h}$ from *grid 2h* to a vector $\boldsymbol{y}^h$ of the *grid h*, we define the *prolongation operator* $I_{2h\to h}$, such that $\boldsymbol{y}^h = I_{2h\to h}\boldsymbol{y}^{2h}$. The most used is to take $I_{2h\to h}$ to be the following linear interpolation operator:

$$I_{2h\to h} : \begin{cases} \boldsymbol{y}^h_{2i,2j} = \boldsymbol{y}^{2h}_{i,j} & 1 \le i,j \le \frac{n}{2} - 1 \\ \boldsymbol{y}^h_{2i+1,2j} = \frac{1}{2}(\boldsymbol{y}^{2h}_{i,j} + \boldsymbol{y}^{2h}_{i+1,j}) & 0 \le j \le \frac{n}{2} - 1 \\ \boldsymbol{y}^h_{2i,2j+1} = \frac{1}{2}(\boldsymbol{y}^{2h}_{i,j} + \boldsymbol{y}^{2h}_{i,j+1}) & 0 \le j \le \frac{n}{2} - 1 \\ \boldsymbol{y}^h_{2i+1,2j+1} = \frac{1}{4}(\boldsymbol{y}^{2h}_{i,j} + \boldsymbol{y}^{2h}_{i+1,j} + \boldsymbol{y}^{2h}_{i,j+1} + \boldsymbol{y}^{2h}_{i+1,j+1}) & 0 \le j \le \frac{n}{2} - 1 \end{cases} \tag{4.17}$$

that constructs $\boldsymbol{y}^h$ taking the components of $\boldsymbol{y}^{2h}$ associated with the points of the *grid h*, which are the points of *grid h* that are present in *grid 2h*, illustrated in Figure 4.6(a). Also, taking the average of the adjacent components of $\boldsymbol{y}^{2h}$ to be the odd components of $\boldsymbol{y}^{2h}$, which are associated with the points of *grid h* that are not present in *grid 2h*, illustrated in Figure 4.6(b)-

(d).

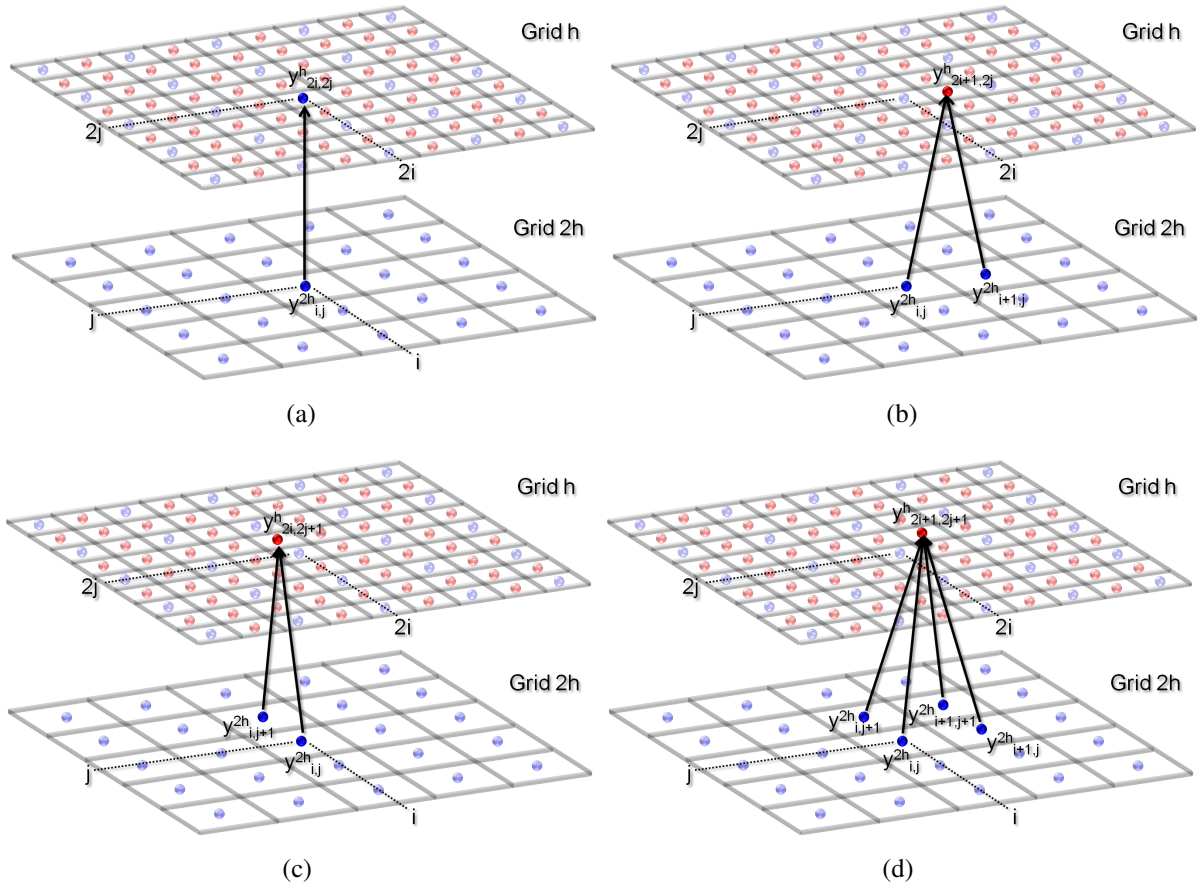


Figure 4.6 – Prolongation operator. This operator constructs $\boldsymbol{y}^h$ taking the components of $\boldsymbol{y}^{2h}$ associated with the points of the *grid h*. Two or more arrows represent the average of the respective components of $\boldsymbol{y}^{2h}$.

Source: Created by the ahtor.

We can approximate $\boldsymbol{A}^h\boldsymbol{e}^h = \boldsymbol{r}$ as follow. First, we transfer the residual $\boldsymbol{r^h}$ to the *grid 2h* through $\boldsymbol{r}^{2h} = I_{h\rightarrow 2h}\boldsymbol{r}^h$; Then, we exactly solve $\boldsymbol{A}^{2h}\boldsymbol{e}^{2h} = \boldsymbol{r}^h$ to get $\widehat{\boldsymbol{e}}^{2h} = [\boldsymbol{A}^{2h}]^{-1}\boldsymbol{r}^{2h}$; Finally, we transfer back $\widehat{\boldsymbol{e}}^{2h}$ to *grid h*, through $\widehat{\boldsymbol{e}}^h = I_{2h\rightarrow h}\widehat{\boldsymbol{e}}^{2h}$.

If $\widehat{\boldsymbol{e}}^h$ is a good approximation to the solution $\boldsymbol{A}^h\boldsymbol{e}^h = \boldsymbol{r}^h$, then this equation is well approximated by the simpler equation $\boldsymbol{A}^{2h}\boldsymbol{e}^{2h} = \boldsymbol{r}^{2h}$. If the error e^h associated with an iterated $[\boldsymbol{p}^h]^{(k)}$ is smooth, i.e., it does not have high frequency components, then the interpolation of $\widehat{e}^{2h}$ gives a good approximation to e^h (PRESS et al., 2007). With this approach, the iterative process

becomes:

$$\begin{array}{lccc} \text{Grid h:} & [\boldsymbol{p}^h]^{(k)} \longrightarrow \boldsymbol{r}^h = \boldsymbol{f}^h - \boldsymbol{A}^h[\boldsymbol{p}^h]^{(k)} & & [\boldsymbol{p}^h]^{(k+1)} = [\boldsymbol{p}^h]^{(k)} + \widehat{\boldsymbol{e}}^h \\ & \downarrow \boldsymbol{r}^{2h} = I_{h\to 2h}\boldsymbol{r}^h & & \uparrow \widehat{\boldsymbol{e}}^h = I_{2h\to h}\widehat{\boldsymbol{e}}^{2h} \\ \text{Grid 2h:} & \boldsymbol{A}^{2h}\boldsymbol{e}^{2h} = \boldsymbol{r}^{2h} & \longrightarrow & \widehat{\boldsymbol{e}}^{2h} = [\boldsymbol{A}^{2h}]^{-1}\boldsymbol{r}^{2h} \end{array} \tag{4.18}$$

which is commonly called *Two-Grid Correction Scheme* (PRESS et al., 2007). With this scheme, we obtain the *Two-Grid cycle*, which will be the basis for any multigrid method. This cycle consists of three steps:

1. **Pre-smoothing**. Apply an iterative smoothing method α_1 times in $\boldsymbol{A}^h\boldsymbol{p}^h = \boldsymbol{f}^h$ with initial approximation $[\boldsymbol{p}^h]^{(0)}$ to obtain $[\boldsymbol{p}^h]^{(k_1)}$.
2. **Two-Grid Correction Scheme**. Apply the *Two-Grid Correction Scheme* 4.18 in $[\boldsymbol{p}^h]^{(k_1)}$ to obtain $[\boldsymbol{p}^h]^{(k_1+1)}$.
3. **Post-smoothing**. Apply an iterative smoothing method α_2 times in $\boldsymbol{A}^h\boldsymbol{p}^h = \boldsymbol{f}^h$ with initial approximation $[\boldsymbol{p}^h]^{(k_1+1)}$ to obtain $[\boldsymbol{p}^h]^{(k_2)}$.

The *Two-Grid Correction Scheme* is responsible for eliminating the low-frequency components of the error while pre and post-smoothing steps are responsible for eliminating high-frequency components of the error. The convergence of the *Two-Grid cycle* strongly depends on the following components:

- The smoothing method;
- The number of smoothing iterations α_1 and α_2;
- The subgrid, i.e., the discretization matrix;
- The inter-grid transfer operators.

4.2.3 Multigrid Method

In the *Two-Grid cycle*, we exactly solve the *Residual Equation* $\boldsymbol{A}^{2h}\boldsymbol{e}^{2h} = \boldsymbol{r}^{2h}$ in the subgrid $2h$. However, if the grid h has too many points, then the subgrid $2h$ will also have too many points. Thus, solving the equation in the subgrid $2h$ will be as difficult as solving the original problem $\boldsymbol{A}^h\boldsymbol{p}^h = \boldsymbol{f}^h$. This suggests that we apply the *Two-Grid cycle* again, but now in $\boldsymbol{A}^{2h}\boldsymbol{e}^{2h} = \boldsymbol{r}^{2h}$ following to subgrid $4h$. This will take us to the problem of solving $\boldsymbol{A}^{4h}\boldsymbol{e}^{4h} = \boldsymbol{r}^{4h}$, to which we can apply again the *Two-Grid cycle*. Figure 4.7 illustrates this hierarchy of grids.

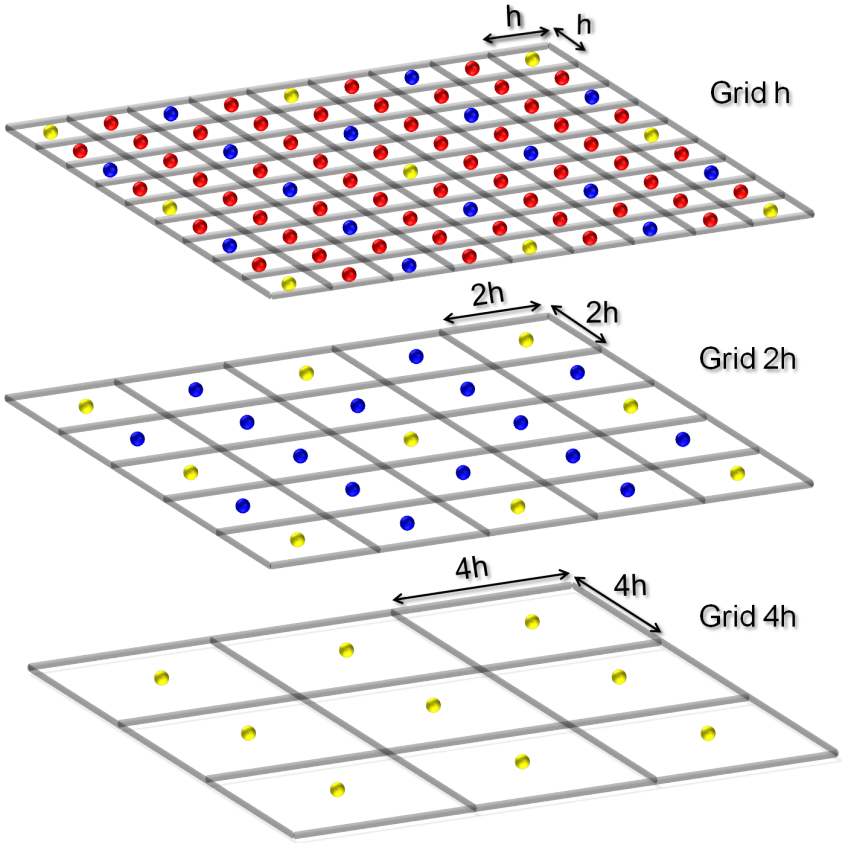


Figure 4.7 – Multigrid representation. A hierarchy of grids where each grid has twice the resolution of its previous grid.

Source: Created by the ahtor.

We continue this process until we reach the subgrid with only one interior point, where the problem is reduced to find e in $\boldsymbol{A}e = r$, which can be solved exactly. Thus, the procedure can be described by the following steps:

1. Apply, α_1 times, an iterative smoothing method to $\boldsymbol{A}^h\boldsymbol{p}^h = \boldsymbol{f}^h$, on grid h, with initial approximation $[\boldsymbol{p}^h]^{(0)}$ to obtain $[\boldsymbol{p}^h]^{(k)}$;
2. Obtain the residual $\boldsymbol{r}^h = \boldsymbol{f}^h - \boldsymbol{A}^h[\boldsymbol{p}^h]^{(k)}$ and transfer it to the subgrid $2h$, through $\boldsymbol{f}^{2h} = I_{h\to 2h}\boldsymbol{r}^h$;
3. Apply, α_1 times, an iterative smoothing method to $\boldsymbol{A}^{2h}\boldsymbol{p}^{2h} = \boldsymbol{f}^{2h}$, on the grid $2h$, with initial approximation $[\boldsymbol{p}^{2h}]^{(0)}$ to obtain $[\boldsymbol{p}^{2h}]^{(k)}$;
4. Obtain the residual $\boldsymbol{r}^{2h} = \boldsymbol{f}^{2h} - \boldsymbol{A}^{2h}[\boldsymbol{p}^{2h}]^{(k)}$ and transfer it to the subgrid $4h$, through $\boldsymbol{f}^{4h} = I_{2h\to 4h}\boldsymbol{r}^{2h}$;
5. $\vdots$

6. Exactly solve $\boldsymbol{A}^{th}\boldsymbol{p}^{th} = \boldsymbol{f}^{th}$ in grid th. The grid th is the smallest grid possible.
7. $\vdots$
8. Transfer $\widehat{\boldsymbol{p}}^{4h}$ to the grid $2h$ through $\widehat{\boldsymbol{p}}^{2h} = I_{4h\rightarrow 2h}\widehat{\boldsymbol{p}}^{4h}$ obtaining $[\boldsymbol{p}^{2h}]^{(k+1)} = [\boldsymbol{p}^{2h}]^{(k)} + \widehat{\boldsymbol{p}}^{2h}$;
9. Apply, α_2 times, an iterative smoothing method to $\boldsymbol{A}^{2h}\boldsymbol{p}^{2h} = \boldsymbol{f}^{2h}$, on the grid $2h$, with initial approximation $[\boldsymbol{p}^{2h}]^{(0)} = [\boldsymbol{p}^{2h}]^{(k+1)}$ to obtain $\widehat{\boldsymbol{p}}^{2h}$;
10. Transfer $\widehat{\boldsymbol{p}}^{2h}$ to the grid h through $\widehat{\boldsymbol{p}}^{h} = I_{2h\rightarrow h}\widehat{\boldsymbol{p}}^{2h}$ obtaining $[\boldsymbol{p}^{h}]^{(k+1)} = [\boldsymbol{p}^{h}]^{(k)} + \widehat{\boldsymbol{p}}^{h}$;
11. Apply, α_2 times, an iterative smoothing method to $\boldsymbol{A}^{h}\boldsymbol{p}^{h} = \boldsymbol{f}^{h}$, on the grid h, with initial approximation $[\boldsymbol{p}^{h}]^{(0)} = [\boldsymbol{p}^{h}]^{(k+1)}$ to obtain $\widehat{\boldsymbol{p}}^{h}$.

4.2.3.1 V-cycle method

The previous steps define the *V-cycle method* because its schematic representation presents the "V" shape. The "V" shape is illustrated in Figure 4.8. This cycle efficiently eliminates all the initial error components since it lets the "heavier" computation, i.e., those involved with the resolution of $\boldsymbol{A}^{th}\boldsymbol{p}^{th} = \boldsymbol{f}^{th}$, to the grids with lower resolution.

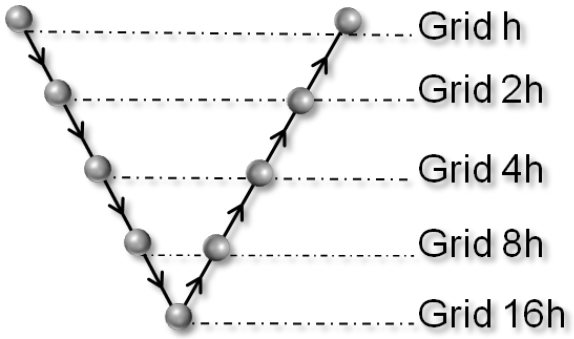


Figure 4.8 – V-cycle representation.

Source: Created by the ahtor.

V-cycle method can be described more compactly as follows:

1. Apply, α_1, times an iterative smoothing method to $\boldsymbol{A}^{h}\boldsymbol{p}^{h} = \boldsymbol{f}^{h}$, on the grid h, with initial approximation $[\boldsymbol{p}^{h}]^{(0)}$ to obtain $[\boldsymbol{p}^{h}]^{(k)}$;
2. If the grid h is the lower resolution grid, then go to step 5 using $[\boldsymbol{p}^{h}]^{(k+1)} = [\boldsymbol{p}^{h}]^{(k)}$. Otherwise, continue.
3. Obtain the residual $\boldsymbol{r}^{h} = \boldsymbol{f}^{h} - \boldsymbol{A}^{h}[\boldsymbol{p}^{h}]^{(k)}$ and transfer it to the subgrid $2h$, through $\boldsymbol{f}^{2h} = I_{h\rightarrow 2h}\boldsymbol{r}^{h}$;

4. Apply the *V-cycle* method to the grid $2h$, i.e., start again the step 1 but this time, for the grid $2h$, $\mu = 1$ times.
5. Transfer $\widehat{\boldsymbol{p}}^{2h}$ to the grid h through $\widehat{\boldsymbol{p}}^{h} = I_{2h\rightarrow h}\widehat{\boldsymbol{p}}^{2h}$ obtaining $[\boldsymbol{p}^{h}]^{(k+1)} = [\boldsymbol{p}^{h}]^{(k)} + \widehat{\boldsymbol{p}}^{h}$;
6. Apply, α_2 times, an iterative smoothing method to $\boldsymbol{A}^{h}\boldsymbol{p}^{h} = \boldsymbol{f}^{h}$, on the grid h, with initial approximation $[\boldsymbol{p}^{h}]^{(0)} = [\boldsymbol{p}^{h}]^{(k+1)}$ to obtain $\widehat{\boldsymbol{p}}^{h}$.

4.2.3.2 W-cycle method

The multigrid $V-cycle$ method is just one of a family of multigrid cycles, which are called μ*-cycle method*. A μ*-cycle* is obtained when we use an arbitrary μ instead of 1 in step 4 of *V-cycle* method, expressed previously. Commonly uses only $\mu = 1$ (*V-cycle*) and $\mu = 2$ which generates the *W-cycle*. Figure 4.9 illustrates the *W-cycle* method. To generate the *W-cycle*, we replace the steps of the *V-cycle* with $\mu = 2$, described previously, which lead us to obtain a solution $\widehat{\boldsymbol{p}}^{h}$ very close to the exact solution when applied only a few times to the original problem $\boldsymbol{A}^{h}\boldsymbol{p}^{h} = \boldsymbol{f}^{h}$.

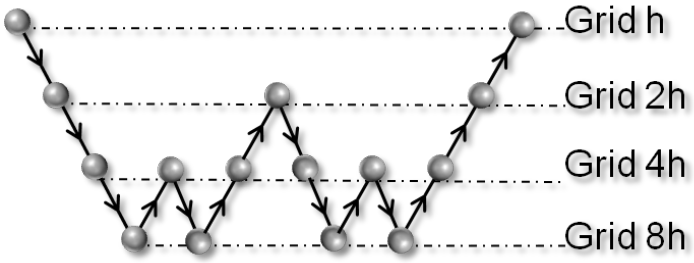


Figure 4.9 – W-cycle representation.

Source: Created by the ahtor.

4.2.4 Full Multigrid Method

We can improve the μ-cycle method if we use a better initial approximation $[\boldsymbol{p}^{h}]^{(0)}$, i.e., values closer to $\boldsymbol{p}^{h}$ in the iteration over $\boldsymbol{A}^{h}\boldsymbol{p}^{h} = \boldsymbol{f}^{h}$ of the original grid h (step 1 of the V-cycle method). For this, a feasible alternative is to transfer $\boldsymbol{f}^{h}$ to the subgid $2h$ through $\boldsymbol{f}^{2h} = I_{h\rightarrow 2h}\boldsymbol{f}^{h}$. Then, solve $\boldsymbol{A}^{2h}\boldsymbol{p}^{2h} = \boldsymbol{f}^{2h}$ approximately to obtain $\widehat{\boldsymbol{p}}^{2h}$, using the μ-cycle. After, transfer $\widehat{\boldsymbol{p}}^{2h}$ to the grid h through $\widehat{\boldsymbol{p}}^{h} = I_{2h\rightarrow h}\widehat{\boldsymbol{p}}^{2h}$ and use this $\widehat{\boldsymbol{p}}^{h}$ as an initial approximation for $\boldsymbol{A}^{h}\boldsymbol{p}^{h} = \boldsymbol{f}^{h}$. Because we use the μ-cycle in grid $2h$ to solve $\boldsymbol{A}^{2h}\boldsymbol{p}^{2h} = \boldsymbol{f}^{2h}$, we again will need

a better initial approximation for $[\boldsymbol{p}^{2h}]^{(0)}$.

This process can be condensed into the following steps:

1. Transfer $\boldsymbol{f}^h$ to the subgrid $2h$ through $\boldsymbol{f}^{2h} = I_{h\rightarrow 2h}\boldsymbol{f}^{2h}$;
2. Transfer $\boldsymbol{f}^{2h}$ to the subgrid $4h$ through $\boldsymbol{f}^{4h} = I_{2h\rightarrow 4h}\boldsymbol{f}^{4h}$;
3. $\vdots$
4. Solve $\boldsymbol{A}^{th}\boldsymbol{p}^{th} = \boldsymbol{f}^{th}$ in the grid th, which is the lower resolution grid, to obtain $\widehat{\boldsymbol{p}}^{th}$;
5. $\vdots$
6. Transfer $\widehat{\boldsymbol{p}^{4h}}$ to the subgrid $2h$ through $[\boldsymbol{p}^{2h}]^{(0)} = I_{4h\rightarrow 2h}\widehat{\boldsymbol{p}}^{4h}$;
7. Apply the μ-cycle η_1 times to $\boldsymbol{A}^{2h}\boldsymbol{p}^{2h} = \boldsymbol{f}^{2h}$ using $[\boldsymbol{p}^{2h}]^{(0)}$ as initial approximation to obtain $\widehat{\boldsymbol{p}}^{2h}$;
8. Transfer $\widehat{\boldsymbol{p}}^{2h}$ to grid h through $[\boldsymbol{p}^{h}]^{(0)} = I_{2h\rightarrow h}\widehat{\boldsymbol{p}}^{2h}$
9. Apply the μ-cycle η_2 times to $\boldsymbol{A}^{h}\boldsymbol{p}^{h} = \boldsymbol{f}^{h}$ using $[\boldsymbol{p}^{h}]^{(0)}$ as initial approximation to obtain $\widehat{\boldsymbol{p}}^{h}$;

This process to obtain appropriated initial approximations to the higher resolution grids using the coarser grids is called *Nested Iteration* and its fusion with the μ-cycle method produces the *Full Multigrid Method*. Figure 4.10 illustrates the Full Multigrid Method, using the v-cycle method. Currently, the Full Multigrid Method is the most powerful method to develop solvers for differential equations.

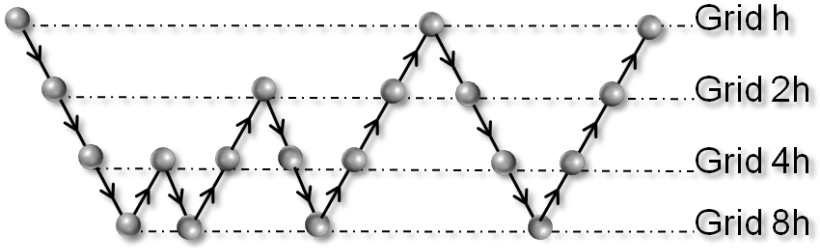


Figure 4.10 – Full Multigrid V-cycle representation.

Source: Created by the ahtor.

4.3 Developing Our Solver

In order to use the FMG method in our planner, we consider that Equation 3.2 is discretized into a uniform grid with cell size h. Let the linear elliptical operator $(\nabla^2 - \epsilon(\boldsymbol{r}).|\nabla|)$ be $\boldsymbol{A}^h$

and Equation 3.2 in matrix form, becomes

$$\boldsymbol{A}^h \boldsymbol{p}^h(\boldsymbol{r}) = 0, \tag{4.19}$$

with $\boldsymbol{f}^h = 0$, for better understanding the algorithm.

Assuming that $\tilde{\boldsymbol{p}}^h$ is an approximate solution to Equation 4.19 and $\boldsymbol{p}^h$ is the exact solution, we can define the error $\boldsymbol{e}^h = \boldsymbol{p}^h - \tilde{\boldsymbol{p}}^h$ and the residual $\boldsymbol{r}^h = \boldsymbol{f}^h - \boldsymbol{A}^h\tilde{\boldsymbol{p}}^h$.

With this information, we can rewrite Equation 4.19 as

$$\boldsymbol{A}^h \boldsymbol{e}^h = \boldsymbol{r}^h, \tag{4.20}$$

which is the *Residual Equation*.

We can obtain numerical advantage if we approximate $\boldsymbol{A}^h$ by a simpler operator, whose solution is an approximation of $\boldsymbol{e}^h$. To obtain this approximation we need a manner to transfer information from the grid with discretization h to a coarser grid.

To propagate the information through the grid hierarchy, we must define two operators: the *Restriction Operator* $\boldsymbol{R}$, and the *Prolongation Operator* $\boldsymbol{P}$. $\boldsymbol{R}$ takes the information in the fine grid, level i in the hierarchy, and restricts it to the coarser grid, level $i+1$ in the hierarchy. Level i represents the grid with discretization h, while level $i+1$ represents the grid with discretization $2h$. $\boldsymbol{P}$ takes the information in the coarser grid, level $i+1$, and interpolates it to the finer grid, level i. Fig. 4.1 illustrates the direction of information propagation of these operators through the grid hierarchy. With $\boldsymbol{e}^h$ solved at level $i+1$, we transfer it back to the grid i and compute the potential through $\boldsymbol{p}^h = \tilde{\boldsymbol{p}}^h + \boldsymbol{e}^h$. Extending this idea to more levels on the hierarchy, we have what is known as *Multigrid V-cycle*, which is a fundamental step of the FMG method.

4.4 Computing the Potential Field

To compute the potential field, initially, all grids $\{\mathcal{M}_1, \mathcal{M}_2, .., \mathcal{M}_N\}$ will represent the environment in a specific resolution, where $\mathcal{M}_N$ is the coarsest representation. The planner receives as input the hierarchy of grids, and computes the potential in the coarsest grid $\mathcal{M}_N$ through Equation 3.4 using any relaxation method. That results in a coarse representation of the potential field.

The potential of the other grids $\mathcal{M}_{i+1}$ is prolongated to $\mathcal{M}_i$, with $N-1 \geq i \geq 2$. This initial approximation is smoothened α_1 iterations using Eq 3.4. Then, the residual of the grid

$\mathcal{M}_i$ is computed through the equation:

$$r_c = \frac{4p_c - p_l - p_r - p_b - p_t}{h^2} + \frac{\epsilon_c \cdot (|p_r - p_l| + |p_t - p_b|)}{2h} \tag{4.21}$$

The *Multigrid V-cycle*, aforementioned, is then executed to compute the error used to correct the potential of the grid $\mathcal{M}_i$, which is smoothened α_2 iterations. The grid $\mathcal{M}_i$ is then ready to be used in the navigation process. These steps: smooth the potential α_1 iterations, process *Multigrid V-cycle* and smooth the new potential α_2 iterations are executed until the potential convergence, with a tolerance $||\tilde{\boldsymbol{e}}||_2 < \gamma$. These steps are illustrated in Algorithm 3.

The fundamental step, *Multigrid V-cycle*, illustrated in Algorithm 4, receives as input the grid $\mathcal{M}_i$ and acts on the error $\tilde{\boldsymbol{e}}$ from the potential convergence for each grid $\mathcal{M}_j$, with $i \leq j \leq N$. Initially, the *Multigrid V-cycle* minimizes the error associated to the grid $\mathcal{M}_j$ through

$$e_c = \frac{e_b + e_t + e_r + e_l}{4} - \epsilon h \cdot \frac{(|e_r - e_l| + |e_b - e_t|)}{8}, \tag{4.22}$$

that is smoothened α_1 iterations. Notice that this process acts on the error. If the current grid is the coarsest grid $\mathcal{M}_N$ then the algorithm solves the error until its convergence, i.e., $||\tilde{\boldsymbol{e}}||_2 < \gamma$. Otherwise, the residual of the error is computed in the grid $\mathcal{M}_j$, through

$$r_c^k = r_c^{k-1} + \frac{4e_c - e_l - e_r - e_b - e_t}{h^2} + \frac{\epsilon_c \cdot (|e_r - e_l| + |e_t - e_b|)}{2h}, \tag{4.23}$$

and restricted to the grid $\mathcal{M}_{j+1}$. The *Multigrid V-cycle* is recursively called to compute the error of the grid $\mathcal{M}_{j+1}$. Finally, the potential of the grid $\mathcal{M}_j$ is corrected with the error computed from the coarse grid $\mathcal{M}_{j+1}$ and smoothened α_2 iterations. Several operators can be used as restriction and prolongation operators. The most commonly used is

$$\boldsymbol{R} = \begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix} \text{ and } \boldsymbol{P} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$$

because it is a good approximation Equation 3.2 (BRANDT, 1977). These operators, as well as the residual and potential computation, are applied only to the free-space cells.

The FMG method obtains a solution in $O(n)$ time (BRIGGS; HENSON; MCCORMICK, 2000), which implies that the time complexity of HCF is also $O(n)$, scaling linearly with the number of free-space cells.

The HCF is used as global planner. When a goal is set, the HCF starts running. Often, in

Algorithm 3 HCF

1: {*Make sure all the grids have the correct environment representation.*}
2: Solve exactly the coarsest grid $\mathcal{M}_N$. {*GS or SOR*}
3: Set $\mathcal{M}_N$ as the finer grid ready to use.
4: **for** i$\leftarrow$ N to 2 **do**
5: Set initial guess $\tilde{p}^{i-1} \leftarrow I.\tilde{p}^{i}$
6: **for** j $\leftarrow$ 1 to η **do**
7: Relax α_1 times on $\mathcal{A}^{i-1}\tilde{p}^{i-1} = 0$.
8: $r^i \leftarrow R.r^{i-1}$
9: $\tilde{e}^i \leftarrow 0$
10: calls V-cycle($\mathcal{M}_i$)
11: Correct $\tilde{p}^{i-1} \leftarrow \tilde{p}^{i-1} + P.\tilde{e}^i$
12: Relax α_2 times on $\mathcal{A}^{i-1}\tilde{p}^{i-1} = 0$.
13: **end for**
14: Set $\mathcal{M}_{i-1}$ as the finer grid ready to use.
15: **end for**

Algorithm 4 V-cycle

1: Relax α_1 times on $\mathcal{A}^i\tilde{e}^i = -r^i$.
2: **if** $\mathcal{M}_i$ is the coarsest grid **then**
3: Go to line 11.
4: **else**
5: $r^i \leftarrow -\mathcal{A}^i\tilde{e}^i$.
6: $r^{i+1} \leftarrow R.r^i$
7: $\tilde{e}^{i+1} \leftarrow 0$
8: calls V-cycle($\mathcal{M}_{i+1}$)
9: **end if**
10: Correct $\tilde{p}^i \leftarrow \tilde{p}^i + P.\tilde{e}^{i+1}$
11: Relax α_2 times on $\mathcal{A}^i\tilde{p}^i = r^i$.

the collision avoidance step, described later, a direction from the global planner will be needed to guide the agents. The HCF uses the higher resolution grid where its potential field is already computed to provide this direction, through Equation 3.6, while the potential field of better resolution grids is still being computed.

5 RESULTS

5.1 Configurable Flows

This section presents several features of the HCF, including quantitative and qualitative analysis. Also, we show the algorithm's performance comparing the proposed technique with the previous one, as well as the A^* algorithm, which is one of the fastest techniques commonly used in agent navigations.

In order to illustrate the potentialities and validate our path-planning approach, we made a set of experiments considering some usual situations. Taking into account the scenario described bellow, we have simulated different behaviors for agents in order to verify some of the considerations made before, as: how to accomplish the same task in different ways; or how different agents avoid collisions with static or dynamic obstacles, for example. In another set of tests, we have run the algorithm considering a variable number of agents with random objectives, behaviors and velocities. Our goal with these experiments was to verify the motion diversity. Experiments simulating realistic paths were also performed.

5.1.1 Reaching the Same Goal in Different Ways

As previously mentioned, for each pair ϵ and $\boldsymbol{v}$ there is only one path and this path can vary according to the information gathered by the agent (e.g. dynamic obstacles). To illustrate the generation of different paths using different pairs ϵ and $\boldsymbol{v}$, we generate 15 paths produced by the same agent to reach a goal, without reproducing any specific behavior. In order to produce each path, we fix $\epsilon = 1$ and vary randomly each $\boldsymbol{v}$ component in the interval $[-1, 1]$. We use the local map composed of 15×15 cells. Figure 5.1 illustrates this situation.

Figure 5.2 shows two frames of a simple animation of two agents. One agent walks from the north to the south while the other one walks from the south to the north. Using our algorithm, the collision between the two agents is automatically avoided since each agent is considered as a dynamic obstacle by the other. However, the final path definition can be more or less natural, depending on the parameters' definition. In the sequence presented on Figure 5.2(a), ϵ was set as 0. In this way, the behavior vector $\mathbf{v}$ is not considered. For the animation shown in Figure 5.2(b), both agents begin the animation with $\epsilon = 0.0$. When the proximity is detected ϵ is changed to 0.6 and the behavior vector $\mathbf{v}$ of each agent is oriented orthogonally to the collision direction, forcing the movement to its right direction.

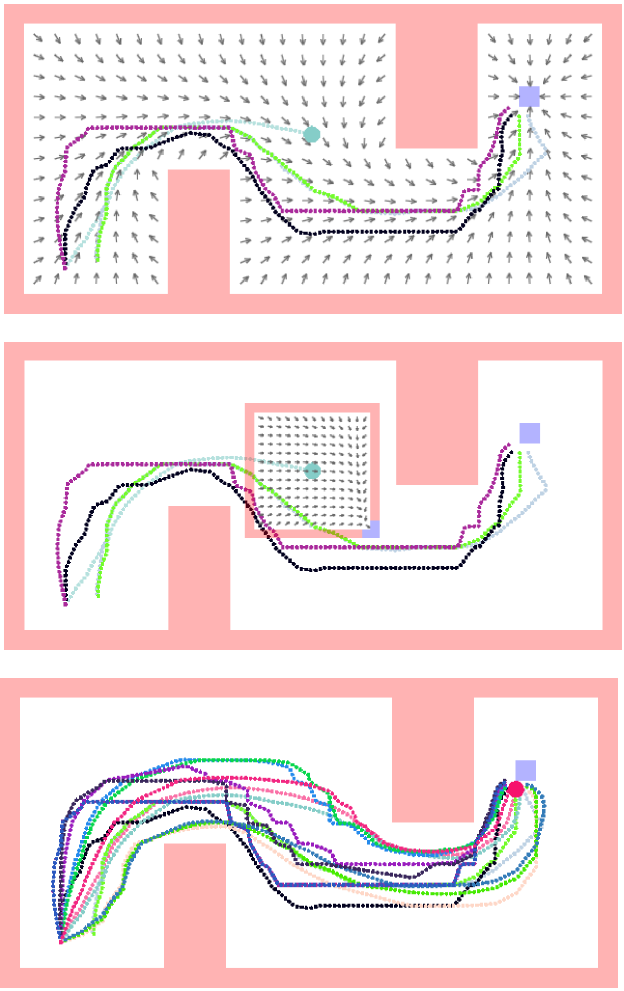


Figure 5.1 – Different paths followed by the same agent from a fixed start position to the goal (blue cell) just varying the parameter $\boldsymbol{v}$ of his local map. 15 paths were generated setting $\epsilon = 1$ and randomly varying $\boldsymbol{v}$ in the interval $[-1, 1]$.

Source: Created by the author.

5.1.2 Varying the Size of the Agent Map

Interesting results can be produced in the way that agents interact with each other in the environment by varying the size of the agent map. The more information on the environment is available to the agent the more it will tend to change its behavior to avoid regions cluttered with obstacles. Figure 5.3 shows two situations where an agent finds a group in its path. In Figure 5.3(a), the agent map has 25x25 cells, whereas, in Figure 5.3(b), the agent map has 50x50 cells. Each cell represents a mapping of 1x1 units of the whole environment. In the first case, the agent crosses the group in the middle, whereas in the second case, the agent avoids the group trying to pass beside it. Based on these experiments, we can see that the map size influences directly the way that the agent behaves in the environment.

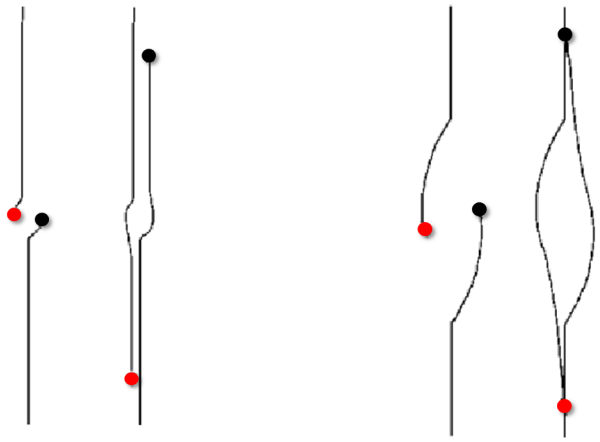


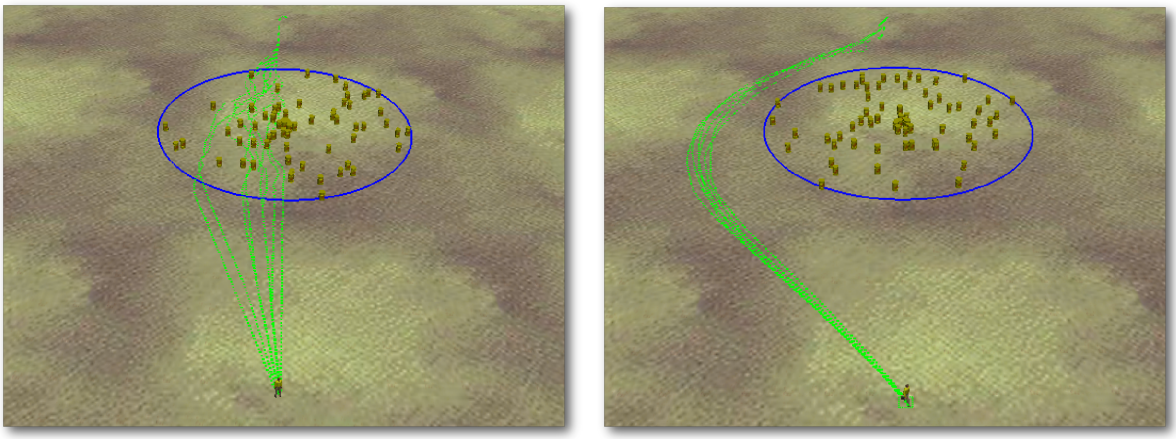
Figure 5.2 – Two animation sequences illustrating collision avoidance produced with different values for the behavior vector and ϵ.

Source: Created by the author.

In order to explain this influence objectively, we performed a couple of experiments varying parameters as the size of the agent map, the density of the obstacle region, and the size of the obstacle region. The relationship between these parameters was then extracted. We have also identified which parameters should be changed to ensure the agent will pass inside a region with plenty of obstacles, or to guarantee it will pass outside. In all simulated situations, the agent starts in the same fixed position and goes towards the same fixed goal position.

We assume the agent map is a squared region with length L, where $15 \leq L \leq 55$ cells. We also consider the agent will find a group of other agents randomly distributed in a circular region with radius r, where $5 \leq r \leq 30$ units. The circular region occupancy varies in a γ rate, where $3\% \leq \gamma \leq 50\%$. For each tuple (r, L, γ) we made 10 experiments to compute the average minimal distance from the agents to the center of the obstacle region while walking through the crowd. Figure 5.4 shows the relationship between parameters L and γ for an obstacle region with $r = 10$ units. In this case, we observe that some pairs (L, γ) will always force the agents to walk through the obstacle region when the average minimal distance is less than 10 (blue zone).

There is an intermediate region in the graph where agents sometimes pass inside, and other time pass outside the obstacles' region. Figure 5.5 shows the results presented in Figure 5.4 from another perspective. We partitioned the surface in Figure 5.4 in three regions according to the values of the average minimal distance produced by each pair (L, γ). If the pair (L, γ) generates, in all experiments, a minimal distance lesser than the radius r, this pair is classified as a point that belongs to the region ρ_0. If the pair generates, in all experiments, a minimal



(a) (b)

Figure 5.3 – (a) Agent map with 25x25 cells. The agent always walks inside the obstacle region. (b) Agent map with 50x50 cells. The agent always walks outside the obstacle region. In both situations, the obstacle region density is equal to 0.1 and the obstacle region radius is equal to 10 units.

Source: Created by the author.

distance greater than the radius r, this pair is classified as a point that belongs to the region ρ_2. Otherwise, the pair belongs to the region ρ_1 – where some experiments produce minimal distances smaller than r and others produce minimal distances greater than r.

Analyzing this classification, we can observe a very interesting result. The intermediate region has an exponential shape defined by the tuple (L,γ, r)

$$\gamma = e^{-(L-\beta)/r} + \alpha \tag{5.1}$$

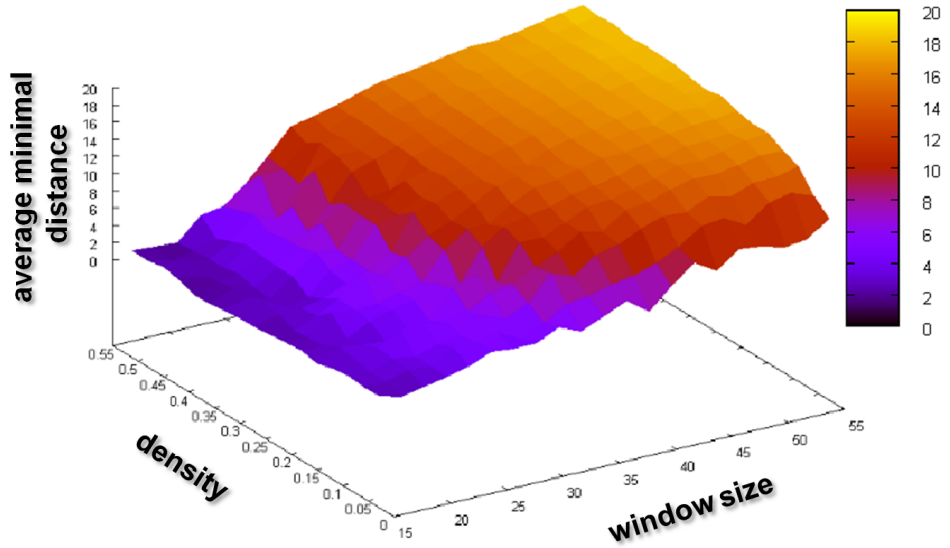
where α e β are constants.

Fitting this region with an exponential function, we get an expression that relates the agent local map, the density of the obstacle region, and the radius of the obstacle region with the agent's behavior. The relationship between these values is given by the equation:

$$f(L, \gamma, r) = e^{-(L-\beta)/r} + \alpha - \gamma \tag{5.2}$$

where the parameters are the same as in Equation 5.1. Using this information we found two functions that classify the parameters' space in ρ_0, ρ_1 and ρ_2:

$$f_i(L, \gamma, r) = e^{-(L-\beta_i)/r} + \alpha_i - \gamma$$



(a)

Figure 5.4 – Relationship between the size of the agent map L and the density γ in a region with radius $r = 10$ units. $L \in [15, 55]$, $\gamma \in [3\%, 51\%]$.

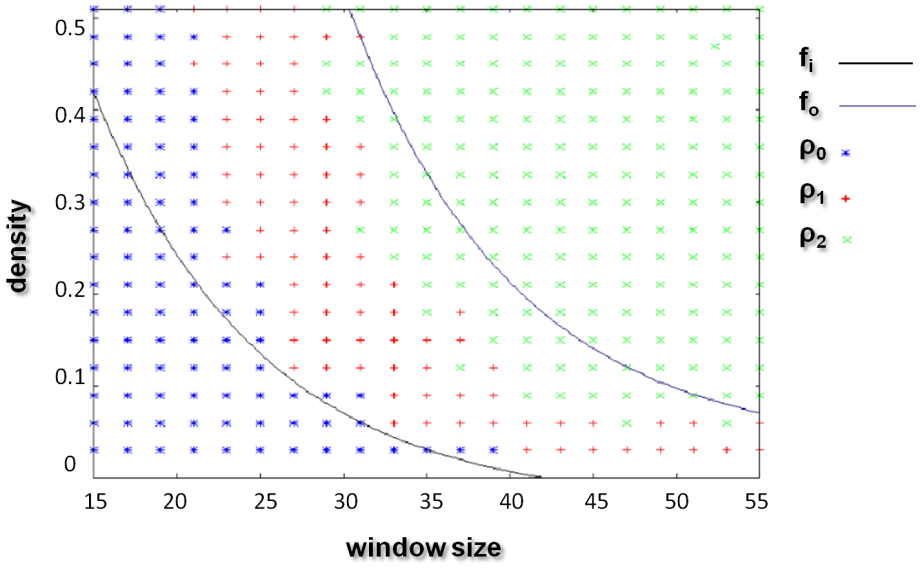
Source: Created by the author.

$$f_o(L, \gamma, r) = e^{-(L-\beta_o)/r} + \alpha_o - \gamma$$

where $\beta_i < \beta_o$ and $\alpha_i = -\alpha_o$.

To specific values of α and β, these functions limit the regions where the agent will have a predetermined behavior, always passing outside the obstacle's region, or always passing inside the obstacle region. If $f_i(L, \gamma, r) < 0$ then the tuple (L, γ) belongs to ρ_0, that is, the agent always crosses the other agents region. If $f_o(L, \gamma, r) > 0$ then the tuple (L, γ) belongs to ρ_2 region, that is, the agent always passes along the other agents. When $f_i(L, \gamma, r) > 0$ and $f_o(L, \gamma, r) < 0$ it is not possible to predict the agent's behavior. The agent may occasionally go inside or outside the obstacle's region, depending on the arrangement of obstacles. Therefore, the pair (L, γ) belongs to ρ_1. In Figure 5.5, functions f_o and f_i use the following parameters: $\alpha_o = 0.3$, $\beta_o = 23$, $\alpha_i = -0.3$ and $\beta_i = 7$.

With these experiments we conclude that, if the agent has a small map – $L << r$ – it will tend to cross through the obstacle's region, unless that region's density is large. On the other hand, if the agent has a large map – $L >> r$ – it tends to consider obstacle's region a single obstacle, passing outside it, unless the density of that region is very small. For the same density, the agent's map size will determine the way the agent will walk through the region. The size of the local map can also be controlled adaptively using, for example, information about the



(a)

Figure 5.5 – Fitting with an exponential function over a circular obstacle region with radius $r = 10$ units. The blue region ρ_0 guarantees that the agent always crosses the obstacle region. The green region ρ_2 guarantees that the agent always passes outside the obstacle region. The red region ρ_1 defines a region where is not possible to predict the agent's behavior.

Source: Created by the author.

internal status of the agent.

5.1.3 Dealing with Dynamic Obstacles

As mentioned above, this technique naturally handles dynamic obstacles. Since each agent is considered by the others as an obstacle, we can map these dynamic obstacles on the agent local map in the same way that other agents are mapped.

When an obstacle is in the agent's field of view, the corresponding cells in the agent's local map are changed to obstacle cells. It means that the agent will continue in a direction free of obstacles. Figure 5.6(a) and (b) shows a situation where the agent must pass through a corridor. In this corridor, there are obstacles moving from one side to another.

When the agent is about to collide, it stops and waits until it is able to continue. Figure 5.6(b) shows the path followed by the agent. The peaks shown in the path means that the agent stops to avoid collision.

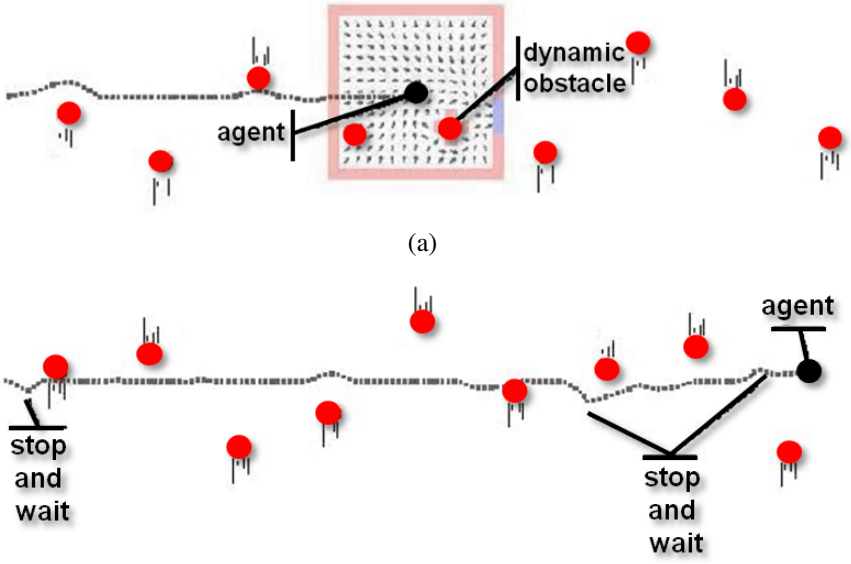


Figure 5.6 – Simulation of an agent passing through a corridor with many dynamic obstacles moving vertically. Dynamic obstacles are seen by an agent as other agents, as you can see by the individual map representation (a). In (b), the path followed by the agent (black circle) after crossing the corridor is shown.

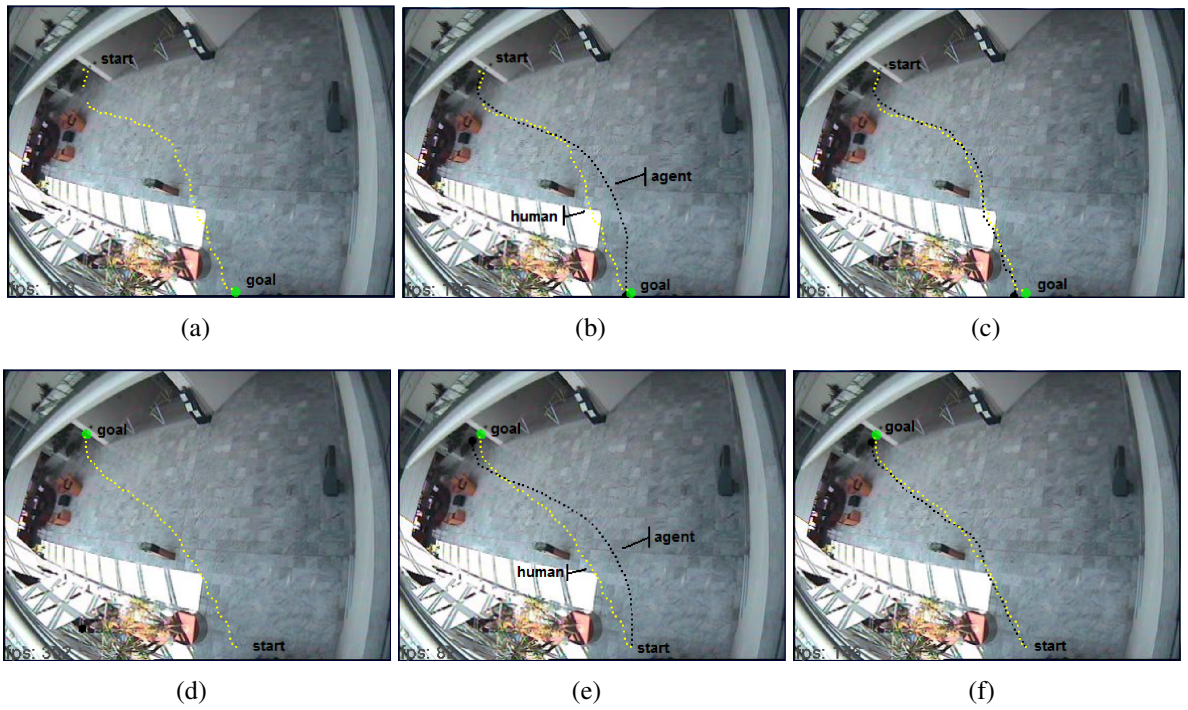
Source: Created by the author.

5.1.4 Generating Realistic Paths

In order to demonstrate that the proposed algorithm produces realistic behaviors for humanoid agents, we compared the paths produced by us with a real human's paths. Real data was obtained from the CAVIAR (CAVIAR, 2011) project database. The CAVIAR project had the objective of improving image-based recognition processes. As a product of this project, a database of human paths extracted from video sequences were generated and is freely accessible over the internet.

To generate realistic steering behaviors we need to conveniently adjust both parameters: ϵ and $\boldsymbol{v}$. Figure 5.7(a) shows a path extracted from a human walking in the INRIA Labs at Grenoble, France. We discretized this environment and applied the CF planner. Figure 5.7(b) shows the potential field for this environment after the goal position was defined. The experiment consisted of an agent starting at the same human start position and trying to reach the same goal position, following a similar path.

When the parameter $\epsilon = 0$ or $\boldsymbol{v} = (0, 0)$ we are not specifying any specific behavior to the agent and the path followed by the agent is produced by Laplace's Equation, as we can see in Figure 5.7(c). This path differs significantly from the real human's path. However, if we



(a) (b) (c)

(d) (e) (f)

Figure 5.7 – We adjusted the parameters of our planner to generate a path that leads the agent to the same goal of the real person with a similar behavior. In each row of this figure we show: the path followed by a human being (a, e); the representation of the environment generated by our planner (b, f); the agent path (in black) calculated by the Laplace's Equation compared with the real path (in yellow) (c, g); and the agent path (in black) calculated by our planner after the parameters adjusting (d, h).

Source: Created by the author.

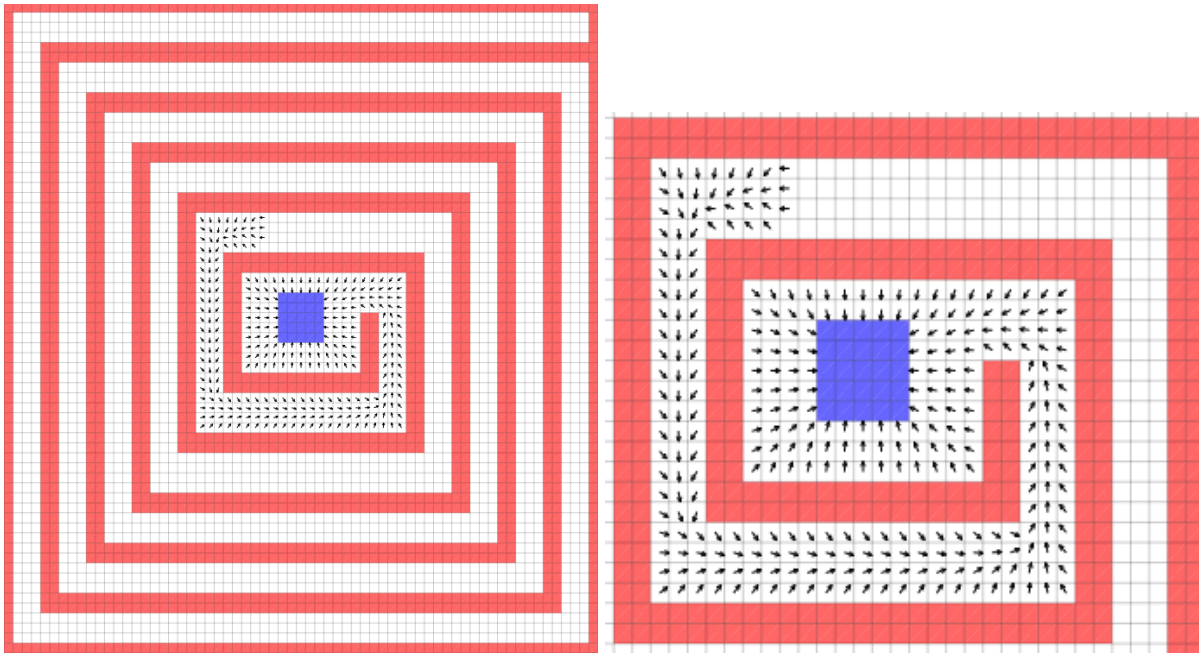
associate specific values to the parameters ϵ and $\boldsymbol{v}$, the agent's path produced by our planner approaches the human's path. Figure 5.7(d) shows a path generated by dynamically changing ϵ and $\boldsymbol{v}$. Up to the half of the path, the parameters $\epsilon = 0.1$ and $\boldsymbol{v} = (0, -1)$ were used. Half the path forward, the parameters were changed to $\epsilon = 0.7$ and $\boldsymbol{v} = (1, 0)$. These values were empirically obtained. We can visually compare and observe that the calculated path is very close to the real one.

Figure 5.7(e) illustrates another situation: the human walks back to his start position. We changed the goal position in the CF planner (see Figure 5.7(f)) and the agent also walks imitating the human. Figure 5.7(c,g) shows the path followed by the agent using Laplace's Equation, i.e., when $\epsilon = 0$ or $\boldsymbol{v} = (0, 0)$. We have dynamically defined again two different behaviors. Up to the half of the path, the parameters $\epsilon = 1.5$ and $\boldsymbol{v} = (1, 0)$ were used. Half the path forward, the parameters were changed to $\epsilon = 1.5$ and $\boldsymbol{v} = (1, -1)$. As before, the path generated by the agent looks very similar to the real human's path. With this simple and qualitative analysis, we can see that our planner is able to generate natural and realistic paths, that can be confounded

with a real one. Also, we can ensure that agents follow a specific path creating regions with high traversal preference over that path.

5.2 Solution to the Flatness Problem

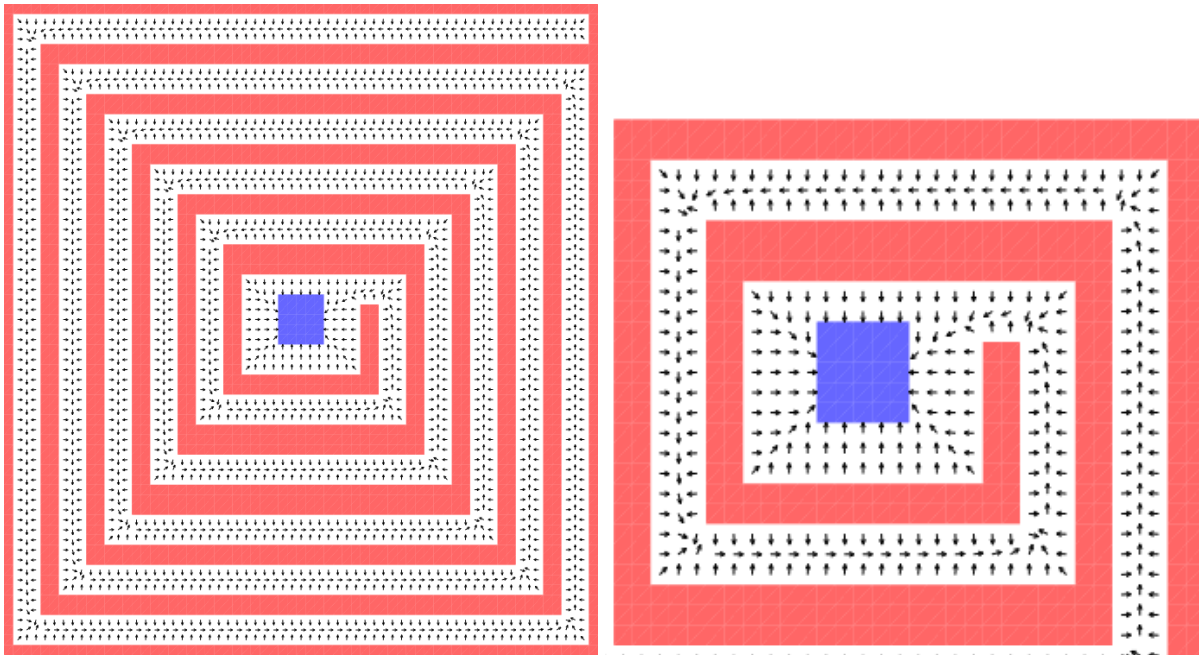
Zhang et al. (ZHANG et al., 2010) proposed a solution to solve this problem manually changing the boundary conditions. However, although their solution is able to change the potential flow, the values to be used depend upon the configuration of the obstacles. We solve this problem automatically, increasing $\epsilon(\boldsymbol{r})$ on free cells. Fig. 5.8 shows a situation where flatness occurs and Fig. 5.9 shows the potential computed using $\epsilon = 1.6$. The higher the ϵ value, the more concave the potential becomes, increasing the influence of the goal position. To choose the ϵ value, we started with $\epsilon = 1.0$ and, during the potential field computation, if a cell presented a null potential value, we increased ϵ in 0.1 units and continue with the potential field calculation until convergence.



(a) (b)

Figure 5.8 – Flatness problem: potential field generated by BVP based planners (a); and a zoom at the central region (b).

Source: Created by the author.



(a) (b)

Figure 5.9 – Flatness solution: potential field generated by the HCF using $\epsilon = 1.6$; and a zoom at the central region (b).

Source: Created by the author.

5.2.1 Considerations about Performance

Considering the visualization of pedestrian simulations, for each new step the agent do, the motion planner should provide its new position and orientation. According to Mazarakis and Avaritsiotis (MAZARAKIS; AVARITSIOTIS, 2005), the frequency of human steps varies from 0.9 to $1Hz$ for someone walking slowly to $3.5Hz$ for someone walking very fast, with a mean of $2Hz$. Then, the performance of a real-time algorithm should be enough to at least: calculate up to 3.5 (2 as a mean) steps per second per agent; animate it; and render the complete scene.

In the experiments performed by us, we were not yet concerned with the rendering quality, but only with the quality of the generated behaviors and the number of agents handled by the algorithm. Figure 5.10 illustrates the results obtained on an ATHLON 64 3500+ 2.21GHz computer with 2.0 GB RAM and graphics board nVidia 7800 GTX. For each step of each agent, considering the mean of 2 steps per second, we have done 60 relaxations of the matrix representing the local map. This allows the management of up to 3300 agents concurrently. If we consider the max of 3.5 steps per second, almost 2500 agents are allowed at the same time.

However, this performance evaluation is simplistic since 3D animation and rendering is not

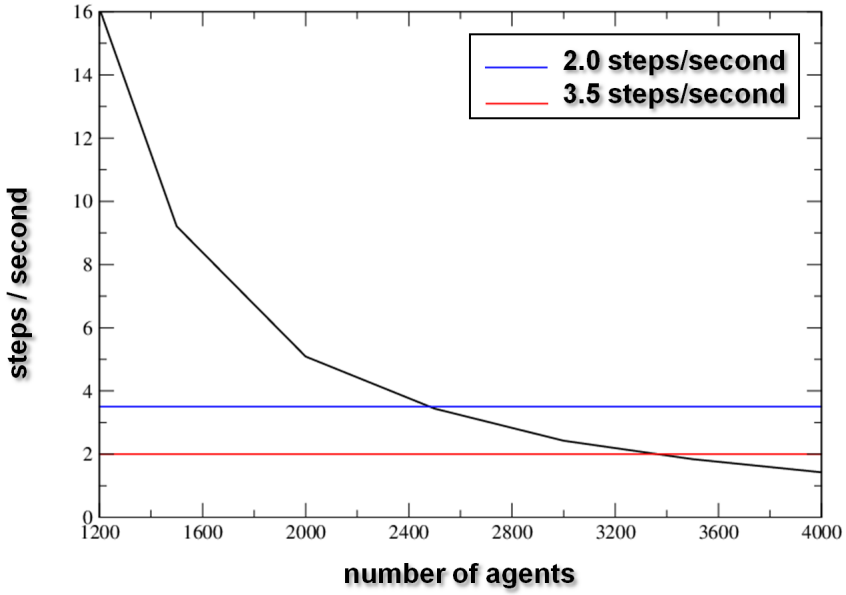


Figure 5.10 – Performance evaluation. The black line shows the relationship between the number of agents with their steps per second. 3300 agents are allowed to move at the same time without affecting the quality of the simulation.

Source: Created by the author.

being considered, as well as algorithm optimizations. Besides, a better compromise between rendering, animation and path planning algorithms can be obtained by reducing the number of relaxations for the local maps. In several examples presented in the previous section, we used 30 relaxations per step done, instead of 60. As an example of our planner potentials, we simulate a crowd with 600 robots walking in real-time in an interactive 3D environment. We also simulate two group of agents, each one with 45 agents. Figure 5.11 and Figure 5.12 shows a snapshot of this situations.

5.3 Hierarchical Configurable Flows

This section presents several features of the HCF, including quantitative and qualitative analysis. Also, we show the algorithm performance comparing the proposed technique with the previous one, as well as the A^* algorithm, which is one of the fastest techniques commonly used in agent navigations.

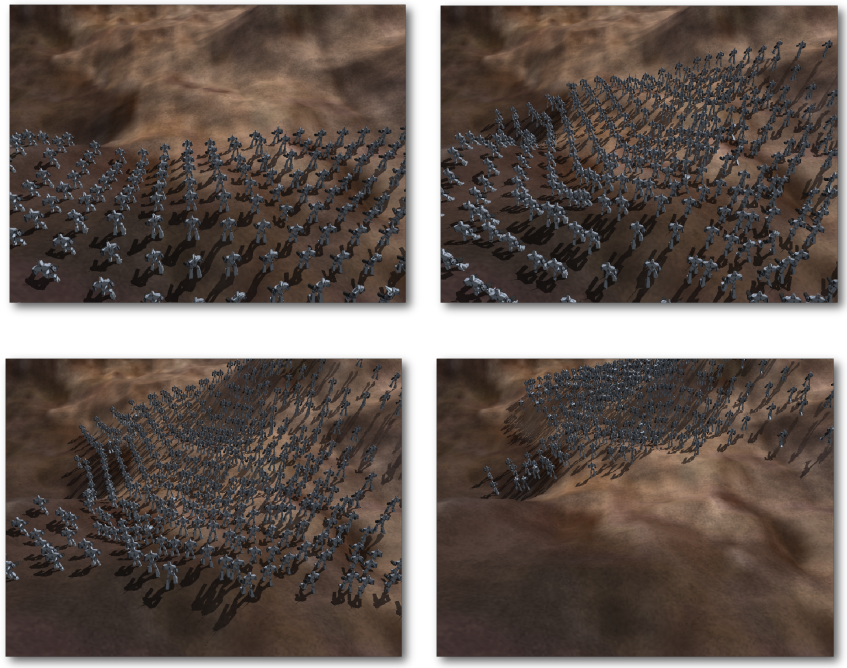


Figure 5.11 – An example of crowd simulation with 600 walking robots.

Source: Created by the author.

5.3.1 Tuning the Algorithm

In order to obtain a better performance, parameters α_1, α_2 and γ, mentioned above, must be carefully set. Parameters α_1 and α_2 are related, respectively, to the number of pre and post-iterations required to smooth the potential while γ is the error allowed to reaches the convergence. These parameters depend on the obstacle and goal configurations.

In our experiments using arbitrary environments with 9×9 cells up to 257×257 cells, both parameters (α_1, α_2) are in the interval $[1, 5]$ and $\gamma = 10^{-3}$. We considered grids with dimensions $d = 2^{N-i+1} + 1$ cells with size $h = \frac{1}{d}$, where i is the i-th grid on the hierarchy.

However, the HCF is not restricted to this grid size. According to Brandt (BRANDT, 1977), this size is more convenient to be used with the prolongation and restriction operators previously mentioned. Usually, the greater the number of coarse grids, the faster is the potential convergence. However, coarse grids do not represent correctly the environment, but even so it should be used in the calculation because it provides a good initial approximation for the potential field, as demonstrated in Chapter 5, but it must never be used during navigation.

5.3.2 Choosing the Number of Grids

During the calculation of the potential field, the convergence rate depends on the number of pre and post iterations and the number of grids used. Usually, the more the number of grids with lower resolution than a particular grid, the faster is the potential convergence in this grid. If the lower resolution grid does not represent correctly the environment, even then it should be used in the calculation of the next level potential field, because it provides a good initial approximation of the potential field, accelerating the convergence. Figure 5.13 shows an environment represented in five different levels of resolutions. The environment is correctly represented in Figure 5.13(a,b,c), but in Figure 5.13(d), the discretization is not sufficient to represent the obstacles. Still, Figure 5.13(e) must be used to accelerate the calculation of the potentia fieldl in the other grids.

5.3.3 Path quality

The HCF has two planning steps: a global and a local planning. The global planner provides a free from obstacles path from any position in the environment to the goal, to guide the agents. The local planner is used to move each agent avoiding dynamic obstacles and to allow that each agent has an individual behavior. We analyze the paths generated by each path planner.

5.3.3.1 Paths produced by the global planner

We compared the paths produced by the HCF with those produced by the CF using a hierarchy of grids with resolution 17×17, 33×33, 65×65 and 129×129 cells. In all experiments we used the parameters ($\alpha_1 = 3$, $\alpha_2 = 4$), and η, allowing the error $\|e\|_2 \leq 10^{-3}$. Figures 5.14(a)-(d) show the quality of the path produced by the CF in an arbitrary environment with several different resolutions. In low-resolution grids, with 17×17 or 33×33 cells, the potential field converges quickly but the path produced has low-quality. In high resolution grids, 65×65 or 129×129 cells, we achieved high-quality paths. The borders of the maps were represented only to delimit the environment and it is essential to the operation of the algorithm. The border on the coarse map occupies a larger area in the environment than the border on the finest map. The discretization size of the coarse map has to be chosen, ensuring that the whole environment will be represented within the map.

Figure 5.15(a) shows the path produced by four levels in the HCF grid hierarchy in the same environment of Figure 5.14. Red, green, dark blue and light blue illustrate the path segments

produced by grids with 17×17, 33×33, 65×65 and 129×129 cells, respectively. Figures 5.15(b) and (c) show the path produced in other different environments. When the potential field of a coarse map converges, it becomes available to be used by the agent. Meanwhile, the potential field in other grids are being calculated and, when it converges, the agent can use it. The better the resolution of the grid, the better the path quality. We can see that the union of paths in grids with different resolutions is smooth, continuous and has almost the same quality of the path computed using the finest grid resolution (Figure 5.14(d)).

5.3.3.2 Paths produced by the local planner

The global planner is consulted and a direction free from obstacle is given each time an agent moves. The agent uses this direction to update his local map to continue the navigation process. Figure 5.17 and Figure 5.16(a,b,c) show fifty agents using the HCF to navigate in arbitrary environments. Red, green, dark blue and light blue illustrate the agent path using the HCF grids with 17×17, 33×33, 65×65 and 129×129 cells, respectively. The agents start to move immediately after the coarsest grid is computed.

Fig. 5.16(a,b,c) show fifty agents using the HCF to navigate in arbitrary environments. Red, green, dark-blue and light-blue lines illustrate the agent path using the HCF grids with 17×17, 33×33, 65×65 and 129×129 cells, respectively. Agents start to move immediately after the coarsest grid is computed, with 17×17 cells (Fig. 5.16(a)), whose potential is computed very fast. When the grid with 33×33 cells is ready, the agents switch to this grid, producing the green path shown in Fig. 5.16(b). Then, while agents navigate, the HCF computes a better resolution grid (Fig. 5.16(c) and (d)). In all experiments we used the parameters ($\alpha_1 = 3$, $\alpha_2 = 4$), and $\gamma = 10^{-3}$. We can see that transitions between paths in grids with different resolutions are smooth and continuous. Our planner does not produce paths that minimize the distances. Figure 5.17 highlights that.

Figure 5.18 presents a path with a non-smooth stretch. The non-smoothness is caused when agents avoid collisions with others and stop moving until the corridor was unobstructed. Fig. 5.19 shows five hundred agents walking down the street. The goal was set in the bottom left corner. Fig 5.19(a-d) shows the situation where there is no preferential regions, $\epsilon(\boldsymbol{r}) = 0$. Agents cross the street without respecting crosswalks. In Fig 5.19(e-h), we use $\epsilon(\boldsymbol{r}) = 0.6$ in the crosswalks, causing the agents to prefer these regions. To handle agent collisions we use the RVO (BERG; LIN; MANOCHA, 2008) collision avoidance.

5.3.4 Considerations about Performance

To compute the performance of the HCF, we use 10 different environments with arbitrary obstacles and goal configurations. In all cases, the parameters α_1 and α_2 used in Algorithms 1 and 2, were set as 3 and 4, respectively, whereas the parameter η was calculated based on the error $\|e\|_1 \leq 10^{-3}$.

Table 5.1 shows the performance of the HCF, the original CF (using GS, SOR and a Quadtree discretization), and the A^* algorithm in an Intel Quad Core 2.4 GHz with 4 GB RAM. The environments have different sizes and were represented using grids with 9×9, 17×17, 33×33, 65×65, 129×129 and 257×257 cells. We can observe that the HCF far surpasses the original CF using both GS and SOR algorithms. It is also important to stress that the time spent by the HCF is the time required to compute the potential in every lower level grid in the hierarchy. For a map with 129×129 cells, we need to generate and compute the potential of all maps in the hierarchy, i.e., maps with 9×9, 17×17, 33×33, 65×65, and 129×129 cells.

Table 5.1 – *Performance evaluation. Comparing the HCF with the CF using SOR, GS, GS in a Quadtree subdivision and the A* algorithm. To compute the A* performance, we considered the largest path found in each environment simulation.*

Resolution	Technique				
	HCF	CF (Quadtree)	CF (SOR)	CF (GS)	A^*
9×9	0.00004s	0.00013s	0.00006s	0.00068s	0.00008s
17×17	0.00033s	0.00082s	0.00097s	0.00126s	0.00022s
33×33	0.00176s	0.00379s	0.00191s	0.00776s	0.00057s
65×65	0.00755s	0.03746s	0.01430s	0.14758s	0.00174s
129×129	0.03948s	0.08331s	0.11827s	2.10312s	0.00557s
257×257	0.13049s	0.58274s	3.00010s	113.00941s	0.01981s

Source: Created by the author.

Most experiments show that the HCF performance is about 100 to 700 times faster than the original CF. As the HCF is computationally efficient, any changes in the environment can be mapped into the grids and the potential field is quickly recomputed. Despite the time required to compute the potential field in a large environment (as grids over 217×217 cells) is not acceptable for real-time simulation, the HCF guarantees the real-time since the agent can move immediately when the potential field in lower level grids converge. Using a Quadtree structure to discretize the environment we achieved better results than the original CF. Although a quadtree has fewer cells than the regular homogeneous grids used in the HCF, the HCF sur-

passed the performance of the quadtree in all experiments and still produces better quality paths, as we can see in Figure 5.20(a),(b).

In most cases, for a single agent, the A^* algorithm is faster than the HCF. The reason is that the A^* computes only one path from the agent current position to its goal, while the HCF computes all paths from any free position of the environment to the goal, i.e., we have 16,384 paths produced by HCF, as we can see in Figure 5.20(b),(c). This allows the planning of different paths for many agents simultaneously and with no additional cost. Moreover, in the HCF the agent starts moving immediately when the coarsest grid is computed, i. e., the HCF coarsest grid is computed while the A^* was being processed. Furthermore, the HCF generates smooth paths, whereas A^* needs a post-processing step to smooth the path generated in order to ensure a high-quality navigation.

The performance of the HCF also depends on the obstacle configurations. Table 5.2 shows the performance of HCF insofar we increase the amount of obstacles in the environment. In this experiment, we consider 0% up to 60% of the environment area covered by obstacles. In each situation, we use 10 different environments with arbitrary obstacles and goal configurations. We can see that HCF's performance varies depending on the amount of obstacles. The best case was when the environment has no obstacles. In this situation, the restriction and prolongation operators efficiently approximate the solution. The worst case was when the obstacles cannot be properly represented in the grids with lower resolutions. This situation occurred in the grid with 10% of its area covered by obstacles, where there are obstacles too small to be represented in all grids. It is important to stress that despite the variation in performance for computing the finer resolution grids (grids over 127×127 cells), the navigation process starts after the calculation of the potential field on the grids with the lowest resolution possible, which takes place in real-time regardless the amount of obstacles in the environment.

Table 5.2 – *HCF performance considering a percentage of the environment covered by obstacles.*

Resolution	Percentage of the environment covered by obstacles						
	0%	10%	20%	30%	40%	50%	60%
33×33	0.00145s	0.00391s	0.00130s	0.00165s	0.00146s	0.00173s	0.00146s
65×65	0.00624s	0.00738s	0.00889s	0.00850s	0.00827s	0.00810s	0.00546s
129×129	0.02890s	0.03273s	0.06647s	0.06423s	0.05446s	0.05222s	0.03031s
257×257	0.239716s	0.75886s	0.71448s	0.42049s	0.60265s	0.63038s	0.34974s

Source: Created by the author.

In our implementation, we are only considering grids with size equal to $2^i + 1$, where i is

the i-th grid on the hierarchy. The HCF is not restricted to this grid size. However, according to Brandt (BRANDT, 1977), this size is more convenient and economic than any other sizes in the prolongation and restriction processes and accelerates the convergence.

5.4 Discussion

We qualitatively compare our technique with classically known techniques: Pelechano, Allbeck and Badler (2007), Treuille, Cooper and Popović (2006), Berg et al. (2008), Yeh et al. (2008), Patil et al. (2011), Harabor and Botea (2008a) and our HCF planner.

Most of the techniques used in current games are property of industries and are not revealed. Were chosen for comparison techniques that can be used for virtual agents navigation. The items selected for comparison were those significant for someone to choose the technique for use in an application. The comparison is shown in Table 5.3.

There are two important points that must be commented about the HCF. Obstacles must be correctly represented in all grids. In our implementation, we defined the obstacles on the finest grid and assumed that the reduction operator will naturally propagate it to other maps during the algorithm execution. It is important to observe that the coarse grid should be discretized enough to represent all obstacles, ensuring a consistent and smooth path during navigation. The second limitation is that the HCF algorithm only handles regions with higher or lower traversal preferences but not directional preferences. Until now we cannot configure regions with preferential directions.

Table 5.3 – *Comparison of agent navigation models.*

	Pelechano	Treuille	van den Berg	Yeh	Patil	Hara	**HCF**
Usability	complex	medium	easy	easy	medium	medium	complex
Hardware Support	–	–	parallelizable	–	–	–	parallelizable
Performance	thousands	thousands	thousands	thousands	thousands	hundreds of thousands	thousands
Preprocessing	no	no	yes	no	no	yes	no
Integration (other techs.)	no	no	no	yes	yes	no	yes
Path quality	medium	good	good	good	good	medium	good
Group of agents	no	yes	no	no	yes	no	yes
Multi-Size agents	no	no	no	no	no	yes	no
Agent's individuality	yes	no	no	yes	yes	yes	yes
Dynamic obstacles	handle	handle	handle	handle	may have local minima	handle	handle
Environment	large/complex	mapped to a grid	large/complex	large/complex	mapped to a grid	mapped to a grid	mapped to a grid
Interactive	no	yes	no	no	yes	no	yes
Terrain trav. capabilites	no	no	no	no	no	yes	yes

Source: Created by the author.

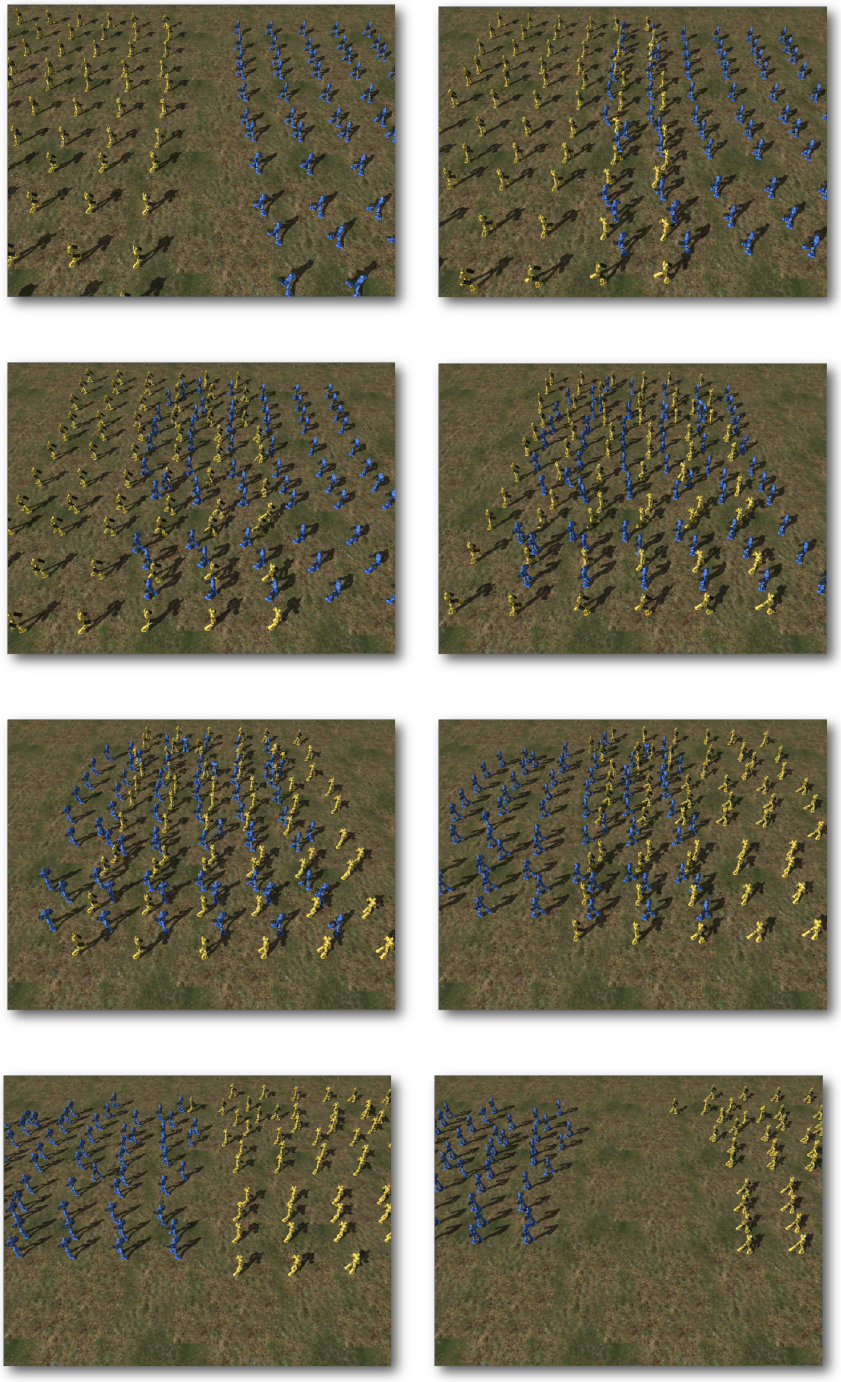
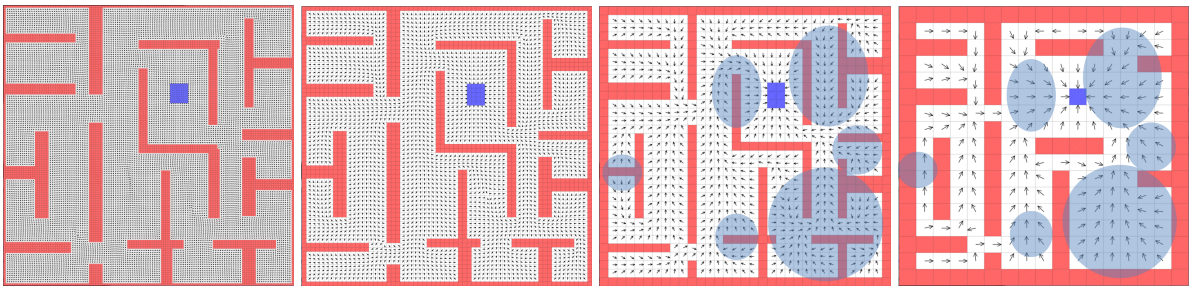


Figure 5.12 – Two groups of agents passing through each other.

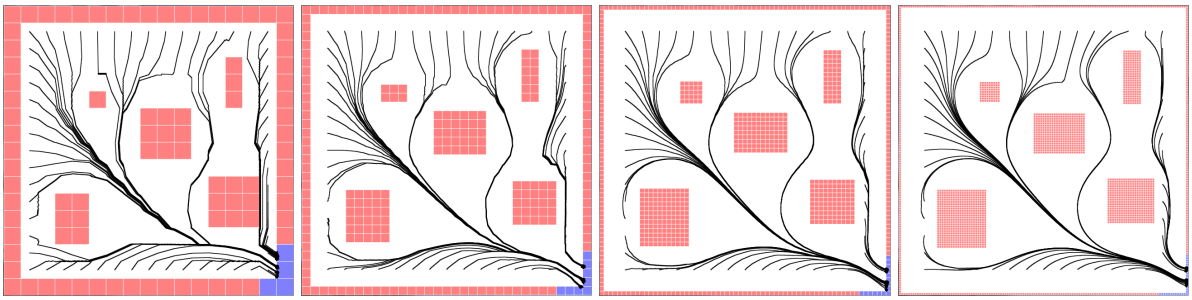
Source: Created by the author.



(a) 129×129 (b) 65×65 (c) 33×33 (d) 17×17

Figure 5.13 – Different levels of discretization of an environment. In (a), (b) and (c) the environment is correctly represented. From (c) to (d) there is a loss of information in the environment's representation. The circular regions highlighted these losses.

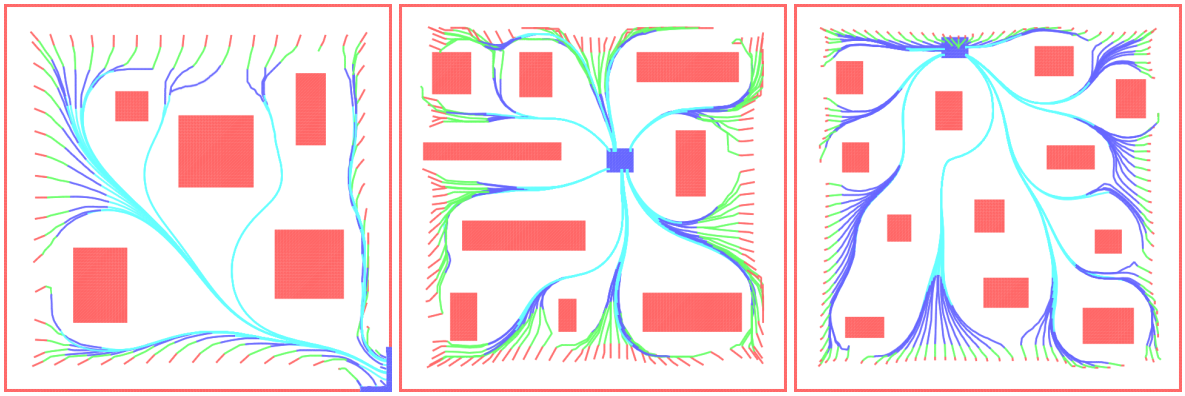
Source: Created by the author.



(a) 17×17 cells (b) 33×33 cells (c) 65×65 cells (d) 129×129 cells

Figure 5.14 – Paths produced by the CF different resolutions. The path produced by HCF is a combination of these paths.

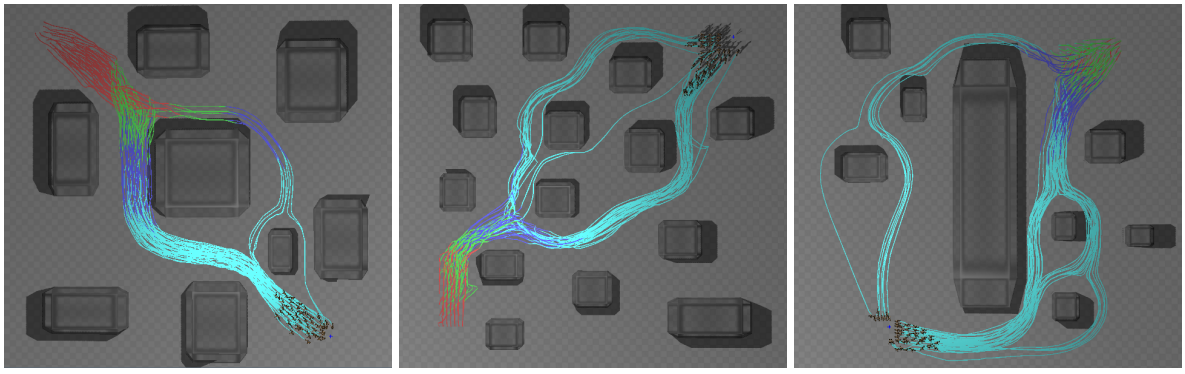
Source: Created by the author.



(a) (b) (c)

Figure 5.15 – Path produced by the HCF. Red, green, dark blue and light blue line segments illustrate the paths using grids with 17×17, 33×33, 65×65 and 129×129 cells, respectively.

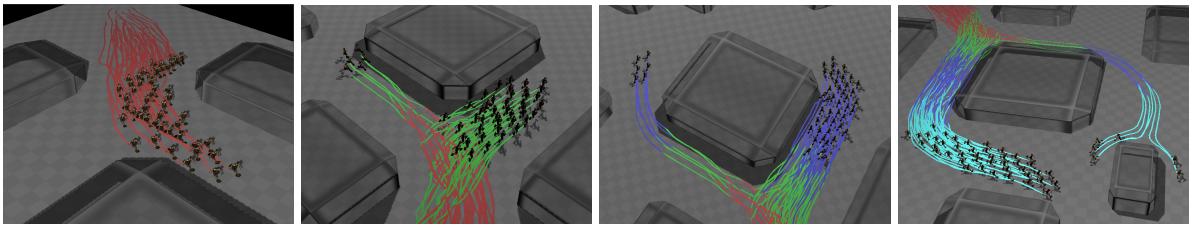
Source: Created by the author.



(a) (b) (c)

Figure 5.16 – Paths produced by the agents using the HCF. The red, green, dark blue and light blue lines illustrate the agent's path using HCF grids with 17×17, 33×33, 65×65 and 129×129 cells, respectively.

Source: Created by the author.



(a) (b) (c) (d)

Figure 5.17 – Paths produced by the agents using the HCF. The agents start to move using the coarsest grid and switch to grids with finer resolution insofar it was computed. Red, green, dark blue and light blue lines illustrate the agents path using grids with 17×17, 33×33, 65×65 and 129×129 cells, respectively.

Source: Created by the author.

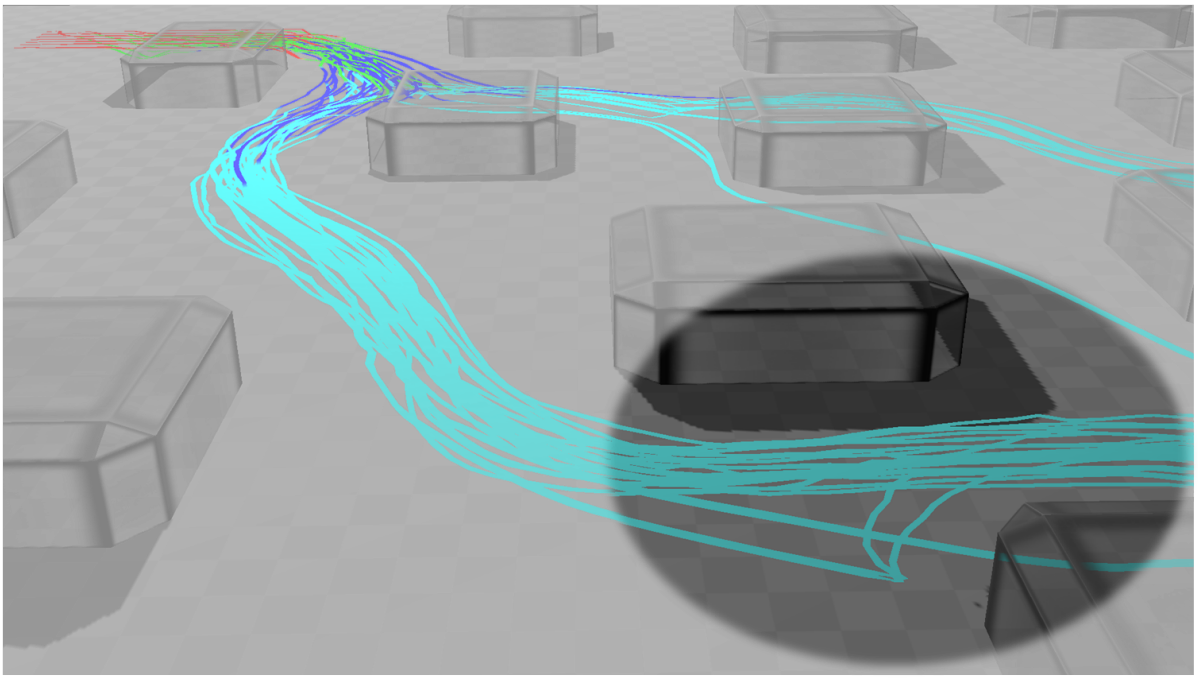
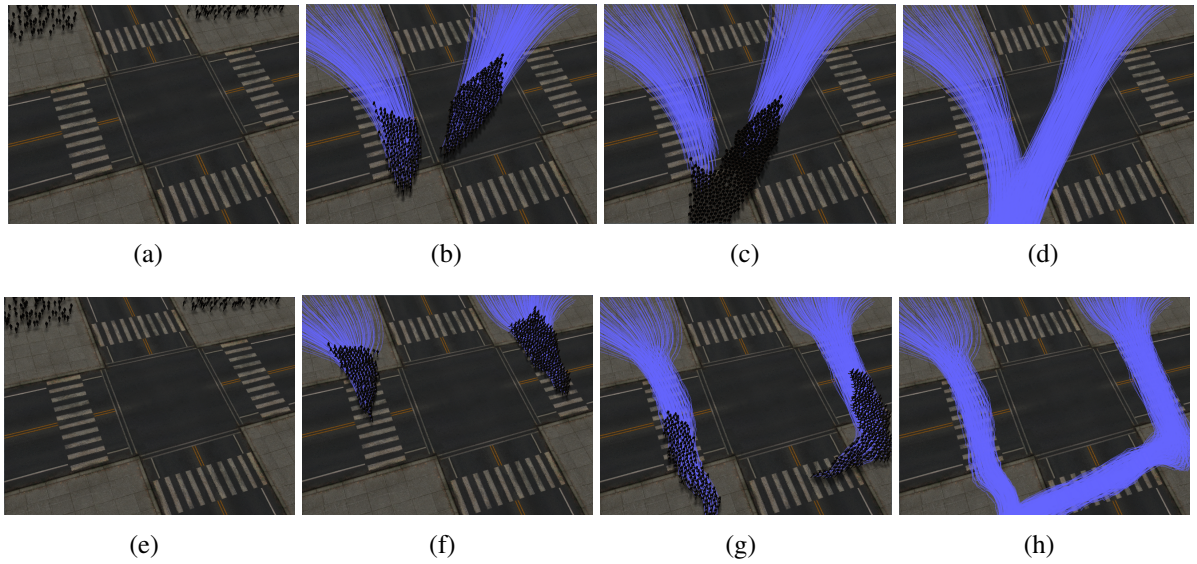


Figure 5.18 – Non-smooth path. This happens because the agents avoid collisions with others, diverting them.

Source: Created by the author.

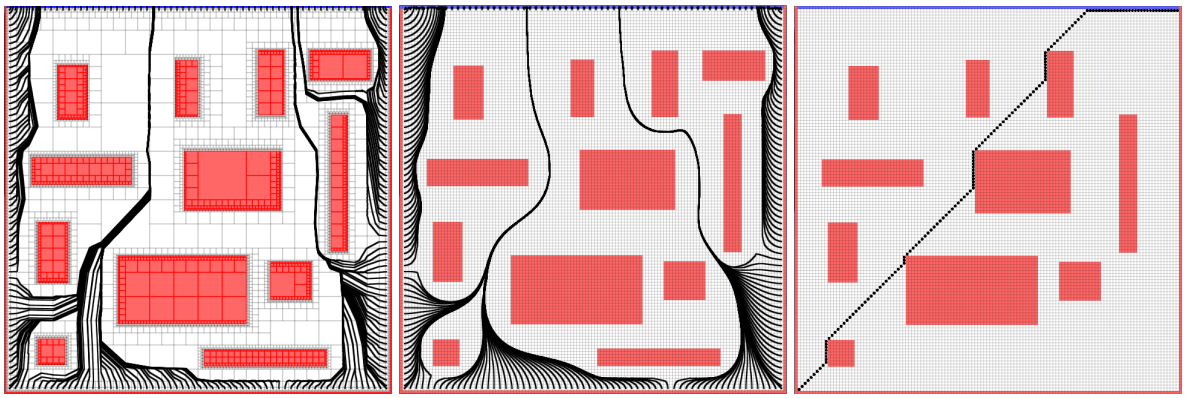


(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 5.19 – Agents walking down the street. Agents walking without respecting crosswalks, with $\epsilon(\boldsymbol{r}) = 0$ everywhere in the environment (a,b,c,d). Using $\epsilon(\boldsymbol{r}) = 0.6$ in crosswalk regions causes the agents to walk through it.

Source: Created by the author.



(a) Quadtree path (b) HCF path (c) A^* path

Figure 5.20 – Paths produced by the CF in a quadtree structure (a), in a homogeneous grid (b) and the largest path produced by the A^* algorithm in this same grid (c).

Source: Created by the author.

6 CONCLUSIONS

Our proposal consists in a novel method of agent navigation in environments that can be mapped to a grid. We had punctual contributions on the Configurable Flows, a path planner based on the numerical solution of boundary value problems that can generate realistic and natural steering behaviors for virtual humans. The main advantage of the proposed technique compared with existing agent navigation techniques is that the same principle is used to achieve a wide variety of features required for the agent navigation and simulation.

We demonstrate that the correct adjustment of the behavior vector $\boldsymbol{v}$ and the parameter ϵ can produce interesting behaviors. These behaviors can be interchanged to produce complex motions, revealing the agent personality, mood and intentions.

The guiding potential is free of local minima, what constitutes a great advantage when compared to the traditional potential fields method. Furthermore, the method proposed is complete (TREVISAN et al., 2006) and generates smooth and safe paths that can be directly used in mobile robots and virtual agents. The local information gathered by agent sensors allows treating dynamic obstacles, as other agents navigating in the environment. Experiments using real human paths obtained from the CAVIAR (CAVIAR, 2011) project shows that our technique can generate natural and realistic paths.

A drawback is that the potential gets flat far from the target position due to numerical precision. In these regions, the gradient's length becomes equal to zero. We call this the "Flatness Problem". We solve this problem by introducing a novel form to the core equation to control the potential field's curvature that can be easily applied to environments composed of inhomogeneous terrains with different types of preferences. The preference can be defined using the terrain type, the terrain elevation, or any other property.

Using the ability to deal with the inhomogeneous terrain facilitates the integration of the path planning stage with terrain reasoning (STERREN, 2001), a making the planner more robust and creating broad application possibilities

Our main contribution is a hierarchical path planner that combines our Configurable Flows Planner with the Full Multigrid method. The main weakness of this CF Planner is the computational cost to find a solution. On the other hand, the Full Multigrid method solves elliptic PDEs efficiently – as Laplace's Equation – through a combination of solutions at several resolution levels.

Basically, this method takes an instance of the problem on a grid of pre-specified fineness and generates coarser grids containing a cruder problem representation. The method solves the

problem on the coarsest grid, which is easy and cheaper, and obtains successive solutions on finer and finer grids. This combination improves the efficiency of the CF Planner expressively. Results in simulation shown that the HBVP PP spends less than 1% of the time needed to compute a solution using the original planner. Our proposal can contribute to several areas of research including agent navigation, game pathfinding, crowd simulation and robotics. Our publications reinforce the relevance of this contribution.

6.1 Future Work

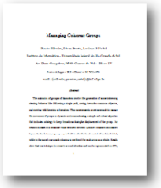
As future work, there are several branches where we can improve the technique presented here. We suggest:

- A parallel version of the HCF. A GPU-based version seems to be effective because the grid hierarchy can be directly acquired through the mipmapping process.
- Extend to 3D environments. The technique proposed here handles 2D environments and surfaces that can be mapped into a grid. However, we are restricted to navigate on the surface, not taking into account the degrees of freedom of a 3D environment. There is no mathematical limitation to extend the technique to handle 3D environments, but the computational cost could be a limitation.
- Extend to planning over triangular meshes. Using the triangular mesh as the environment discretization can have significant improvements. First, we can perfectly fit the obstacles' shape and we can deal with looping surfaces, since the environment is discretized in set of triangular cells and the neighborhood are connected. Also, we can use the mesh Level of Detail (LOD) the represent the hierarchy of grids on the HCF.
- Implement the HCF using a Quadtree since it is already a hierarchical structure and has much fewer cells to be used to compute the potential field. Because it is an adaptative structure, it best fits the obstacle's shape.
- Deal with very large environments. We can use a hybrid technique or a composition of multigrid to handle large environments, as those presented in a Massive Multiplayer Online Role-Playing Game.
- We are able to handle preferential regions controlling the curvature of the potential field. We can include a directional parameter to control the direction of the curvature distortion, handling directional preferential regions.
- Explore agent's behavior. We show that is possible to generate realistic and natural paths.

Using the preferential regions, we can explore a family of functions based on agent's personality, mood and reasoning.

7 CONTRIBUTIONS

Five papers about the CF Path Planner have been proposed during the course. In addition, another paper is being revised to be sent to a journal yet to be chosen. The last one is about HCF integration with Preferential Regions and exposing the solution of the Flatness Problem, i. e., a problem that occurs when the precision is not sufficient to represent the potential field, causing the gradient to vanish. Publications achieved during the Ph.D. time are listed below. The fist four papers are related to the thesis. The last two papers are directly about the thesis.



Manage Coherent Groups (SILVEIRA; PRESTES; NEDEL, 2008), proposing a method to handled group of virtual characters;

GPU Accelerated Path Planning for Multi-agent in Virtual Environments (FISCHER; SILVEIRA; NEDEL, 2009), presenting a GPU-based implementation of the BVP Path Planner;

Natural Steering Behaviors for Virtual Pedestrians (SILVEIRA et al., 2010a), describing the BVP Path Planner with several analysis;

Path-Planning for RTS Games Based on Potential Fields (SILVEIRA et al., 2010b), showing the application of the BVP Path Planner in a RTS game and comparing its performance with the native RTS planner;

Papers directly about the thesis:

Fast Path Planning using Multi-Resolution Boundary Value Problems (SILVEIRA; PRESTES; NEDEL, 2010), proposing the use of the HBVP Path Planner in the robotics domain;

Hierarchical Configurable Flows: a path-planning method for real-time agents. HFC Path Planner integrated with Preferential Regions as well as the solution to the Flatness Problem for multi-agent simulation, submitted to journal IEEE Transactions on Visualization and Computer Graphics.

8 SUPPLEMENTARY MATERIAL: FUNDAMENTAL CONCEPTS

The study of systems of linear equations are very important on science computation, since they result from discrete models of various types of applications, such as fluid dynamics, weather prediction, climate simulation, motion planner, linear programming, combinatorial optimization and so on. This chapter presents some methods that can be used to solve systems of linear equations obtained from the discretization of partial differential equations.

8.1 System of Linear Equations

A system of linear equations can be defined as a set of m equations with n unknowns. In the general form, it can be written as:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = & b_m, \end{array}$$

where $x_1, x_2, ..., x_n$ are the unknowns, $a_{11}, a_{12}, ..., a_{mn}$ are the coefficients of the system, and $b_1, b_2, ..., b_m$ are the constant terms.

Another helpful view is the matrix form. In this form, the linear system above can be

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{8.1}$$

where $\boldsymbol{A}$ is an $m \times n$ matrix, $\boldsymbol{x}$ is a column vector with n entries, and $\boldsymbol{b}$ is a column vector with m entries, i. e.,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \ddots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}. \tag{8.2}$$

A solution for a linear system is an assignment of numbers to the variables $x_1, x_2, ..., x_n$ such that each of the equations is simultaneously satisfied. There are several algorithms for solving a system of linear equations, however, there are two basic classes of methods for solving a system. The first class is represented by *direct methods*. They theoretically give an exact

solution in a (predictable) finite number of steps. Unfortunately, this does not have to be true in computational solvers due to rounding errors: an error made in one step spreads in all following steps. Direct methods are robust and can be used to solve any type of system. However, they are inadequate for solving sparse systems since they do not take advantage of the sparsity of the coefficient matrix $vectA$, making this approach impractical due to storage problems.

The second class is called *iterative methods*. The idea is to start with an initial approximation to the solution (which does not have to be accurate at all), changing this approximation in several steps to bring it closer to the true solution. Once the approximation is sufficiently accurate, this is taken to be the solution to the system. Unlike direct methods and due to its storage efficiency, iterative methods are often used to solve large and sparse systems of equations. Many scientific applications use differential equations in its formulation, and when discretized, resulting in large and sparse systems of equations. The use of iterative methods is more appropriate in these cases.

8.2 Iterative Methods

The basic idea of iterative methods is to construct a sequence of vectors $\boldsymbol{x^k}$ from an initial guess $\boldsymbol{x^0}$ which converges to the exact solution $\boldsymbol{x}$ of the matrix equation $\boldsymbol{Ax} = \boldsymbol{b}$. For a convergent sequence of vectors $\boldsymbol{x^k}$, we must ensure that the matrix equation has a unique solution, or that matrix $\boldsymbol{A}$ is invertible. These methods are commonly called smoothing methods, due to their capacity to remove the eigenfunctions with high-frequencies.

8.2.1 Jacobi Method

The Jacobi method is an iterative technique that solves the left-hand side of this expression for $\boldsymbol{x}$, using the previous value for $\boldsymbol{x}$ on the right-hand side. Analytically, this may be written as:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(bi - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, ..., n. \tag{8.3}$$

It is important to observe that the computation of $x_i^{(k+1)}$ requires each element in $x^{(k)}$ except itself. We cannot overwrite $x_i^{(k)}$ with $x_i^{(k+1)}$, as that value will be needed by the rest of the computation. Equation 8.3 is then iterated until it converges.

8.2.2 Gauss-Seidel Method

The Gauss-Seidel method (GS) is an iterative technique that solves the left-hand side of this expression for $\boldsymbol{x}$, using the previous value for $\boldsymbol{x}$ on the right-hand side. Analytically, this may be written as:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(bi - \sum_{j>i} a_{ij} x_j^{(k)} - \sum_{j<i} a_{ij} x_j^{(k+1)} \right), \quad i = 1, 2, ..., n. \tag{8.4}$$

The formula for the Gauss-Seidel method is extremely similar to that of the Jacobi method. The computation of $x_i^{(k+1)}$ uses only the elements of $\boldsymbol{x}^{(k+1)}$ that have already been computed, and only the elements of $\boldsymbol{x}^{(k)}$ that have yet to be advanced to iteration $k + 1$. This means that, unlike the Jacobi method, only one storage vector is required as elements can be overwritten as they are computed, which can be advantageous for very large problems.

8.2.3 Successive Over-Relaxation Method

The method of Successive Over-Relaxation (SOR) is a variant of the Gauss-Seidel method for solving a linear system of equations, resulting in faster convergence. Like the GS method, SOR is an iterative technique that solves the left-hand side of this expression for $\boldsymbol{x}$, using the previous value for $\boldsymbol{x}$ on the right-hand side. Analytically, this may be written as:

$$x_i^{(k+1)} = (1 - \omega)\, x_i^{(k)} \frac{\omega}{a_{ii}} + \left(bi - \sum_{j>i} a_{ij} x_j^{(k)} - \sum_{j<i} a_{ij} x_j^{(k+1)} \right), \quad i = 1, 2, ..., n. \tag{8.5}$$

where ω is a relaxation factor that accelerates the convergence.

The method is convergent only for $0 < \omega < 2$. The weak point of this method is the choice of the ω value. The acceleration of the SOR method occurs only in a fairly narrow window around the correct value of ω. As a heuristic, it is better to take ω slightly too large, rather than slightly too small. The optimum ω value depends on the system of linear equations we are solving. There are some heuristics in the literature (PRESS et al., 2007) to choose the optimum value.

REFERENCES

BANDI, S.; THALMANN, D. Space discretization for efficient human navigation. **Computer Graphics Forum**, v. 17, p. 195–206, 1998.

BARRAQUAND, J.; LANGLOIS, B.; LATOMBE, J.-C. Numerical potential field techniques for robot path planning. **IEEE Transactions on Systems, Man, and Cybernetics**, v. 22, p. 224–241, 1992.

BARTEL, A. et al. Real-time outdoor trail detection on a mobile robot. In: **Proceedings of the 13th IASTED International Conference on Robotics and Applications**. Anaheim, CA, USA: ACTA Press, 2007. (RA '07), p. 477–482. Available from Internet: <http://dl.acm.org/citation.cfm?id=1659997.1660096>.

BAYAZIT, O. B.; LIEN, J.-M.; AMATO, N. M. Roadmap-based flocking for complex environments. In: **Proceedings of the 10th Pacific Conference on Computer Graphics and Applications**. Washington, DC, USA: IEEE Computer Society, 2002. (PG '02), p. 104–. Available from Internet: <http://portal.acm.org/citation.cfm?id=826030.826615>.

BERG, J. van den; LIN, M. C.; MANOCHA, D. Reciprocal velocity obstacles for real-time multi-agent navigation. In: **IEEE International Conference on Robotics and Automation**. [S.l.]: IEEE, 2008. (ICRA'08), p. 1928–1935.

BERG, J. van den et al. Interactive navigation of multiple agents in crowded environments. In: **Proceedings of the 2008 symposium on Interactive 3D graphics and games**. New York, NY, USA: ACM, 2008. (I3D '08), p. 139–147. Available from Internet: <http://doi.acm.org/10.1145/1342250.1342272>.

BERSETH, G.; KAPADIA, M.; FALOUTSOS, P. Steerplex: Estimating scenario complexity for simulated crowds. In: **Proceedings of Motion on Games**. New York, NY, USA: ACM, 2013. (MIG '13), p. 45:67–45:76. ISBN 978-1-4503-2546-2. Available from Internet: <http://doi.acm.org/10.1145/2522628.2522650>.

BICHO, A. de L. et al. Simulating crowds based on a space colonization algorithm. **Computers & Graphics**, v. 36, n. 2, p. 70–79, 2012. Available from Internet: <http://dblp.uni-trier.de/db/journals/cg/cg36.html#BichoRMJPM12>.

BLEIWEISS, A. Gpu accelerated pathfinding. In: **Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware**. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008. (GH '08), p. 65–74. Available from Internet: <http://portal.acm.org/citation.cfm?id=1413957.1413968>.

BOUVIER, E.; GUILLOTEAU, P. Crowd simulation in immersive space management. In: **Proceedings of the Eurographics workshop on Virtual environments and scientific visualization**. London, UK: Springer-Verlag, 1996. (WVESE'96), p. 104–110. Available from Internet: <http://portal.acm.org/citation.cfm?id=276034.276045>.

BRANDT, A. Multi-level adaptive solutions to boundary-value problems. **Mathematics of Computation**, v. 31, p. 330–390, 1977.

BRIGGS, W. L.; HENSON, V. E.; MCCORMICK, S. F. **A multigrid tutorial (2nd ed.)**. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

CAGIGAS, D. Hierarchical d* algorithm with materialization of costs for robot path planning. In: **Robotics and Autonomous Systems**. [S.l.: s.n.], 2005. (RAS'05), p. 190–208.

CAPCOM. Dead rising. 2006.

CAVIAR. **EC Funded CAVIAR project/IST 2001 37540, http://homepages.inf.ed.ac.uk/rbf/CAVIAR/, last access: october (2011)**. 2011.

CINEMA, N. L. Lord of the rings: The return of the king. 2003.

CONNOLLY, C. I.; BURNS, J. B.; WEISS, R. Path planning using laplace's equation. In: **In Proceedings of the 1990 IEEE International Conference on Robotics and Automation**. [S.l.: s.n.], 1990. (ICRA'90), p. 2102–2106.

CONNOLLY, C. I.; GRUPEN, R. A. On the applications of harmonic functions to robotics. **International Journal of Robotic Systems**, v. 10, p. 931–946, 1993.

DAPPER, F. et al. **Simulating Pedestrian Behavior with Potential Fields**. Springer Berlin Heidelberg, 2006. 324-335 p. (Lecture Notes in Computer Science, v. 4035). Available from Internet: <http://dx.doi.org/10.1007/11784203_28>.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische Mathematik**, Springer Berlin / Heidelberg, v. 1, n. 1, p. 269–271, dec. 1959. Available from Internet: <http://dx.doi.org/10.1007/BF01386390>.

DUTRA, T. B. et al. A Multipotential Field Model for Crowds with Scalable Behaviors. In: **26th Conference on Graphics, Patterns and Images**. IEEE, 2013. (SIBGRAPI'13), p. 31–38. Available from Internet: <http://www.ucsp.edu.pe/sibgrapi2013/eproceedings/technical/114093_2.pdf>.

ENTERTAINMENT, B. Starcraft 2: Wings of liberty. 2010.

FIORINI, P.; SHILLERT, Z. Motion planning in dynamic environments using velocity obstacles. **International Journal of Robotics Research**, v. 17, p. 760–772, 1998.

FISCHER, L.; NEDEL, L. Semi-automatic navigation on 3d triangle meshes using bvp based path-planning. In: **Proceedings of the 2011 24th SIBGRAPI Conference on Graphics, Patterns and Images**. Washington, DC, USA: IEEE Computer Society, 2011. (SIBGRAPI '11), p. 33–40. Available from Internet: <http://dx.doi.org/10.1109/SIBGRAPI.2011.30>.

FISCHER, L. G.; SILVEIRA, R.; NEDEL, L. Gpu accelerated path-planning for multi-agents in virtual environments. In: **Proceedings of the 2009 VIII Brazilian Symposium on Games and Digital Entertainment**. Washington, DC, USA: IEEE Computer Society, 2009. (SBGAMES '09), p. 101–110. Available from Internet: <http://dx.doi.org/10.1109/SBGAMES.2009.20>.

FOUDIL, C.; NOUREDDINE, D. Collision avoidance in crowd simulation with priority rules. **European Journal of Scientific Research**, EuroJournals, v. 15, n. 1, oct. 2006.

FULGENZI, C.; SPALANZANI, A.; LAUGIER, C. Dynamic obstacle avoidance in uncertain environment combining pvos and occupancy grid. In: **in Proc. IEEE Int. Conf. on Robotics and Automation**. [S.l.: s.n.], 2007. (ICRA'07), p. 1610–1616.

GARCIA, F.; KAPADIA, M.; BADLER, N. Gpu-based dynamic search on adaptive resolution grids. In: **Robotics and Automation, 2014 IEEE International Conference on**. [S.l.: s.n.], 2014. (ICRA'14), p. 1631–1638.

GERAERTS, R.; OVERMARS, M. Clearance based path optimization for motion planning. In: **IEEE International Conference on Robotics and Automation**. [S.l.: s.n.], 2004. (ICRA'04), p. 2386–2392.

GERAERTS, R.; OVERMARS, M. H. The corridor map method: a general framework for real-time high-quality path planning. **Journal of Visualization and Computer Animation**, v. 18, p. 107–119, 2007.

GOLAS, A.; NARAIN, R.; LIN, M. Hybrid long-range collision avoidance for crowd simulation. In: **Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games**. New York, NY, USA: ACM, 2013. (I3D '13), p. 29–36. Available from Internet: <http://doi.acm.org/10.1145/2448196.2448200>.

GOLAS, A.; NARAIN, R.; LIN, M. C. Continuum modeling of crowd turbulence. **Phys. Rev. E**, American Physical Society, v. 90, p. 042816, Oct 2014. Available from Internet: <http://link.aps.org/doi/10.1103/PhysRevE.90.042816>.

GUY, S. J. et al. Pledestrians: a least-effort approach to crowd simulation. In: **Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2010. (SCA '10), p. 119–128. Available from Internet: <http://portal.acm.org/citation.cfm?id=1921427.1921446>.

HARABOR, D.; BOTEA, A. Hierarchical Path Planning for Multi-Size Agents in Heterogeneous Environments. In: **Proceedings of the IEEE Symposium on Computational Intelligence and Game**. Perth, Australia: [s.n.], 2008. (CIG'08), p. 258–265.

HARABOR, D.; BOTEA, A. Hierarchical Path Planning for Multi-Size Agents in Heterogeneous Environments. In: **IEEE Symposium on Computational Intelligence and Games**. Perth, Australia: [s.n.], 2008. (CIG'08), p. 258–265.

HART, N. J. N. P. E.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE Transactions on Systems, Science, and Cybernetics**, SSC-4, n. 2, p. 100–107, 1968.

HELBING, D. et al. Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. **Transportation Science**, INFORMS, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, v. 39, p. 1–24, February 2005. Available from Internet: <http://portal.acm.org/citation.cfm?id=1247226.1247227>.

HELBING, D.; MOLNáR, P. Social force model for pedestrian dynamics. **Physical Review E**, American Physical Society, v. 51, n. 5, p. 4282–4286, may 1995. Available from Internet: <http://dx.doi.org/10.1103/PhysRevE.51.4282>.

HODGINS, J.; BROGAN, D. C. Robot herds: Group behaviors for systems with significant dynamics. In: **In Proc. A-Life IV**. [S.l.]: The MIT Press, 1994. (A-LIFE '04), p. 319–324.

HUANG MUBBASIR KAPADIA, N. I. B. T.; KALLMANN, M. Path planning for coherent and persistent groups. In: **Proceedings of the IEEE International Conference on Robtics and Automation**. [S.l.]: IEEE, 2014. (ICRA '14).

HYUN, W. K.; SUH, I. H. A hierarchical collision-free path planning algorithm for robotics. In: **International Conference on Intelligent Robots and Systems**. Washington, DC, USA: IEEE Computer Society, 1995. (IROS'95), p. 2488.

JIN, X. et al. Interactive control of large-crowd navigation in virtual environments using vector fields. **IEEE Comput. Graph. Appl.**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 28, p. 37–46, November 2008. Available from Internet: <http://portal.acm.org/citation.cfm?id=1477045.1477081>.

JU, E. et al. Morphable crowds. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 29, p. 140:1–140:10, December 2010. Available from Internet: <http://doi.acm.org/10.1145/1882261.1866162>.

JUNG, D.; RATTI, J.; TSIOTRAS, P. Real-time implementation and validation of a new hierarchical path planning scheme of uavs via hardware-in-the-loop simulation. **J. of Intell. and Robotic Systems**, Kluwer Academic Publishers, Hingham, MA, USA, v. 54, n. 1-3, p. 163–181, 2009.

KALLMANN, M. Shortest paths with arbitrary clearance from navigation meshes. In: **Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2010. (SCA '10), p. 159–168. Available from Internet: <http://portal.acm.org/citation.cfm?id=1921427.1921451>.

KALLMANN, M.; BIERI, H.; THALMANN, D. Fully dynamic constrained delaunay triangulations. In: BRUNNETT B. HAMANN, H. M. L. L. G. (Ed.). **Geometric Modelling for Scientific Visualization**. first. Heidelberg, Germany: Springer-Verlag, 2003. p. 241–257.

KAMPHUIS, A. et al. Automatic construction of roadmaps for path planning in games. In: **In Proc. Int. Conf. Computer Games: Artificial Intelligence, Design and Education**. [S.l.: s.n.], 2004. (CGAIDE'05), p. 285–292.

KANG, S.-J.; KIM, Y.; KIM, C.-H. Live path: adaptive agent navigation in the interactive virtual world. **Vis. Comput.**, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 26, p. 467–476, June 2010. Available from Internet: <http://dx.doi.org/10.1007/s00371-010-0457-7>.

KAPADIA, M. et al. Multi-domain real-time planning in dynamic environments. In: **Proceedings of the 13th ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. New York, NY, USA: ACM, 2013. (SCA '13), p. 115–124. Available from Internet: <http://doi.acm.org/10.1145/2485895.2485909>.

KAPADIA, M. et al. Dynamic search on the GPU. In: **Intelligent Robots and Systems, 2013 IEEE/RSJ International Conference on**. [S.l.: s.n.], 2013. (IROS'13), p. 3332–3337.

KAPADIA, M. et al. Constraint-aware navigation in dynamic environments. In: **Proceedings of the Seventh international conference on Motion in games**. [S.l.: s.n.], 2013. (MIG'13).

KARAMOUZAS, I.; OVERMARS, M. H. Adding variation to path planning. **Comput. Animat. Virtual Worlds**, John Wiley and Sons Ltd., Chichester, UK, v. 19, p. 283–293, September 2008. Available from Internet: <http://portal.acm.org/citation.cfm?id=1410368.1410380>.

KAVRAKI, L. E. et al. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. **IEEE Transactions on Robotics and Automation**, v. 12, n. 4, p. 566–580, 1996.

KHATIB, O. **Commande dynamique dans l'espace opérational des robots manipulaters en présence d'obstacles**. Thesis (PhD) — École Nationale Supérieure de l'Aéronatique et de l'Espace, France, 1980.

KOREN, Y.; BORENSTEIN, J. Potential field methods and their inherent limitations for mobile robot navigation. In: **In Proc. IEEE Int. Conf. Robotics and Automation**. [S.l.: s.n.], 1991. (ICRA'91, v. 2), p. 1398–1404.

LAMARCHE, F. TopoPlan: a topological path planner for real time human navigation under floor and ceiling constraints. **Computer Graphics Forum**, Blackwell Publishing, v. 28, n. 2, p. 649–658, 2009. Available from Internet: <http://dx.doi.org/10.1111/j.1467-8659.2009.01405.x>.

LAMARCHE, F.; DONIKIAN, S. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. **Computer Graphics Forum**, v. 23, n. 3, p. 509–518, 2004. Available from Internet: <http://dx.doi.org/10.1111/j.1467-8659.2004.00782.x>.

LAVALLE, S. M. **Rapidly-Exploring Random Trees: A New Tool for Path Planning**. [S.l.], 1998.

LAVALLE, S. M. **Planning Algorithms**. Cambridge, U.K.: Cambridge University Press, 2006. Also available at http://planning.cs.uiuc.edu/.

LEE, K. H.; CHOI, M. G.; LEE, J. Motion patches: building blocks for virtual environments annotated with motion data. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 25, p. 898–906, July 2006. Available from Internet: <http://doi.acm.org/10.1145/1141911.1141972>.

LERNER et al. Crowds by example. **Computer Graphics Forum**, Blackwell Publishing, v. 26, n. 3, p. 655–664, sep. 2007. Available from Internet: <http://www.cs.tau.ac.il/~{}alan/Publications/Publications.h>.

LERNER, A. et al. Fitting behaviors to pedestrian simulations. In: **Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. New York, NY, USA: ACM, 2009. (SCA '09), p. 199–208. Available from Internet: <http://doi.acm.org/10.1145/1599470.1599496>.

LOZANO-PéREZ, T.; WESLEY, M. A. An algorithm for planning collision-free paths among polyhedral obstacles. **Commun. ACM**, ACM, New York, NY, USA, v. 22, p. 560–570, October 1979. Available from Internet: <http://doi.acm.org/10.1145/359156.359164>.

MAZARAKIS, G. P.; AVARITSIOTIS, J. N. A prototype sensor node for footstep detection. In: **Proc. of the Second European Workshop on Wireless Sensor Networks**. [S.l.]: IEEE Press, 2005. (EWSN'05), p. 415–418.

MITCHELL, J.; PAPADIMITRIOU, C. The weighted region problem. In: **Proceedings of the third annual symposium on Computational geometry**. New York, NY, USA: ACM, 1987. (SCG '87), p. 30–38. Available from Internet: <http://doi.acm.org/10.1145/41958.41962>.

MITCHELL, J. S. B.; PAPADIMITRIOU, C. H. The weighted region problem: finding shortest paths through a weighted planar subdivision. **J. ACM**, ACM, New York, NY, USA, v. 38, p. 18–73, January 1991. Available from Internet: <http://doi.acm.org/10.1145/102782.102784>.

MOLNAR, P.; STARKE, J. Control of distributed autonomous robotic systems using principles of pattern formation in nature and pedestrian behavior. **IEEE Trans Syst Man Cybern B Cybern**, v. 31, n. 3, p. 433–5, 2001.

MUSSE, S. R.; CASSOL, V. J.; JUNG, C. R. Towards a quantitative approach for comparing crowds. **Computer Animation and Virtual Worlds**, John Wiley & Sons, Ltd, v. 23, n. 1, p. 49–57, 2012. Available from Internet: <http://dx.doi.org/10.1002/cav.1423>.

MUSSE, S. R.; THALMANN, D. A model of human crowd behavior: Group inter-relationship and collision detection analysis. In: **Proc. Workshop of Computer Animation and Simulation of Eurographics**. [S.l.: s.n.], 1997. (WCASE'97), p. 39–51.

NARAIN, R. et al. Aggregate dynamics for dense crowd simulation. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 28, p. 122:1–122:8, December 2009. Available from Internet: <http://doi.acm.org/10.1145/1618452.1618468>.

NIEUWENHUISEN, D.; KAMPHUIS, A.; OVERMARS, M. H. High quality navigation in computer games. **Sci. Comput. Program.**, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 67, p. 91–104, June 2007. Available from Internet: <http://portal.acm.org/citation.cfm?id=1265616.1265864>.

NINOMIYA, K. et al. Planning approaches to constraint-aware navigation in dynamic environments. **Computer Animation and Virtual Worlds**, p. n/a–n/a, 2014. Available from Internet: <http://dx.doi.org/10.1002/cav.1622>.

NOLBORIO, H.; NANIWA, T.; ARIMOTO, S. A quadtree-based path-planning algorithm for a mobile robot. v. 7, p. 555–574, 1990.

Ó'DÚNLAING, C.; YAP, C.-K. A "retraction" method for planning the motion of a disc. **J. Algorithms**, v. 6, n. 1, p. 104–111, 1985.

PAI, D. K.; REISSELL, L.-M. Multiresolution rough terrain motion planning. **IEEE T. Robotics and Automation**, v. 14, n. 1, p. 19–33, 1998. Available from Internet: <http://dblp.uni-trier.de/db/journals/trob/trob14.html#PaiR98>.

PARIS, S.; PETTRE, J.; DONIKIAN, S. Pedestrian reactive navigation for crowd simulation: a predictive approach. **Computer Graphics Forum**, Blackwell Publishing, v. 26, n. 3, p. 665–674, sep. 2007. Available from Internet: <http://www.irisa.fr/prive/donikian/articles/EG07/EG07_final.pdf>.

PATIL, S. et al. Directing crowd simulations using navigation fields. **IEEE Trans. Vis. Comput. Graph.**, v. 17, n. 2, p. 244–254, 2011. Available from Internet: <http://doi.ieeecomputersociety.org/10.1109/TVCG.2010.33>.

PEARL, J. **Heuristics: intelligent search strategies for computer problem solving**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1984.

PELECHANO, N.; ALLBECK, J. M.; BADLER, N. I. Controlling individual agents in high-density crowd simulation. In: **Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation**. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007. (SCA '07), p. 99–108. Available from Internet: <http://portal.acm.org/citation.cfm?id=1272690.1272705>.

PETTRé, J.; LAUMOND, J.-P.; THALMANN, D. A navigation graph for real-time crowd animation on multilayered and uneven terrain. In: **Proc. The First International Workshop on Crowd Simulation**. [S.l.: s.n.], 2005. (V-CROWDS'05).

PRESS, W. H. et al. **Numerical Recipes 3rd Edition: The Art of Scientific Computing**. 3. ed. New York, NY, USA: Cambridge University Press, 2007.

PRESTES, E. et al. Exploration technique using potential fields calculated from relaxation methods. In: **Intelligent Robots and Systems. Proceedings. 2001 IEEE/RSJ International Conference on**. [S.l.: s.n.], 2001. (IROS'12, v. 4), p. 2012–2017 vol.4.

PRESTES, E.; IDIART, M. A. Sculping potential fields in the bvp path planner. In: **IEEE Int. Conf. on Robotics and Biomimetics**. [S.l.: s.n.], 2009. (ROBIO'09).

PRESTES, E.; IDIART, M. A. Computing navigational routes in inhomogeneous environments using bvp path planner. In: **IEEE/RSJ International Conference on Intelligent Robots and Systems**. [S.l.: s.n.], 2010. (IROS'10).

PRESTES, E. et al. Exploration technique using potential field calculated from relaxation methods. **Intelligent Robots and Systems (IROS)**, v. 4, p. 2012–2017, 2001.

REYNOLDS, C. Steering behaviors for autonomous characters. In: **Game Developers Conference 1999**. [s.n.], 1999. (GDC'99). Available from Internet: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.8035>.

REYNOLDS, C. W. Flocks, Herds, and Schools: A Distributed Behavioral Model. **Computer Graphics**, v. 21, n. 4, p. 25–34, 1987. Available from Internet: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.7069>.

SALOMON, B. et al. Interactive navigation in complex environments using path planning. In: **Proceedings of the 2003 symposium on Interactive 3D graphics**. New York, NY, USA: ACM, 2003. (I3D '03), p. 41–50. Available from Internet: <http://doi.acm.org/10.1145/641480.641491>.

SCHELHORN, T. et al. Streets: An agent-based pedestrian model. In: RIZZI, P. (Ed.). **Proc. of the Conference on Computers in Urban Planning and Modelling**. [S.l.: s.n.], 1999. (CUMPUM'99).

SCHWARTZ, J. T.; SHARIR, M. On the piano movers' problem: III. coordinating the motion of several independent bodies. **Int. J. Robot. Res.**, v. 2, n. 3, p. 97–140, 1983.

SEWALL, J. et al. Continuum traffic simulation. **Computer Graphics Forum**, v. 29, p. 439–448, 2010.

SHAO, W.; TERZOPOULOS, D. Autonomous pedestrians. In: SIGGRAPH: ACM SPECIAL INTEREST GROUP ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES. **SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation**. New York, NY, USA: ACM Press, 2005. (SCA '05), p. 19–28.

SHILLER, Z.; YAMANE, K.; NAKAMURA, Y. Planning motion patterns of human figures using a multi-layered grid and the dynamics filter. In: **IEEE International Conference on Robotics and Automation**. [S.l.: s.n.], 2001. (ICRA '01), p. 1–8.

SILVEIRA, R. et al. Natural steering behaviors for virtual pedestrians. **Vis. Comput.**, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 26, p. 1183–1199, September 2010. Available from Internet: <http://dx.doi.org/10.1007/s00371-009-0399-0>.

SILVEIRA, R. et al. Path-planning for rts games based on potential fields. In: **Proceedings of the Third international conference on Motion in games**. Berlin, Heidelberg: Springer-Verlag, 2010. (MIG'10), p. 410–421. Available from Internet: <http://portal.acm.org/citation.cfm?id=1948395.1948446>.

SILVEIRA, R.; PRESTES, E.; NEDEL, L. Fast path planning using multi-resolution boundary value problems. In: **IEEE/RSJ International Conference on Intelligent Robots and Systems**. [S.l.: s.n.], 2010. (IROS'10), p. 4710–4715.

SILVEIRA, R.; PRESTES, E.; NEDEL, L. P. Managing coherent groups. **Comput. Animat. Virtual Worlds**, John Wiley and Sons Ltd., Chichester, UK, v. 19, p. 295–305, September 2008. Available from Internet: <http://portal.acm.org/citation.cfm?id=1410368.1410399>.

STENTZ, A.; MELLON, I. C. Optimal and efficient path planning for unknown and dynamic environments. **Int. J. of Robotics and Automation**, v. 10, p. 89–100, 1993.

STERREN, W. van der. Terrain reasoning for 3D action games. **Game Programming Gems 2**, p. 307–316, 2001.

SUD, A. et al. Real-time navigation of independent agents using adaptive roadmaps. In: **Proceedings of the 2007 ACM symposium on Virtual reality software and technology**. New York, NY, USA: ACM, 2007. (VRST '07), p. 99–106. Available from Internet: <http://doi.acm.org/10.1145/1315184.1315201>.

SUGIYAMA; NAKAYAMA, A.; HASEBE, K. 2-dimensional optimal velocity models for granular flows. p. 155–160, 2001.

THALMANN, D. Populating virtual environments with crowds. In: **Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications**. New York, NY, USA: ACM, 2006. (VRCIA '06), p. 11–11. Available from Internet: <http://doi.acm.org/10.1145/1128923.1128925>.

THALMANN, D.; MUSSE, S. R. **Crowd simulation**. [S.l.]: Springer, 2007. I-XII, 1-242 p.

TORCHELSEN, R. P. et al. Real-time multi-agent path planning on arbitrary surfaces. In: **Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games**. New York, NY, USA: ACM, 2010. (I3D '10), p. 47–54. Available from Internet: <http://doi.acm.org/10.1145/1730804.1730813>.

TREUILLE, A.; COOPER, S.; POPOVIć, Z. Continuum crowds. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 25, p. 1160–1168, July 2006. Available from Internet: <http://doi.acm.org/10.1145/1141911.1142008>.

TREVISAN, M. et al. Exploratory navigation based on dynamic boundary value problems. **Journal of Intelligent and Robotic Systems**, v. 45, p. 101–114, 2006.

TSIOTRAS, P.; BAKOLAS, E. A Hierarchical On-Line Path-Planning Scheme Using Wavelets. In: **European Control Conference**. Kos, Greece: [s.n.], 2007. (ECC'07), p. 2807–2812.

UBISOFT. Assassin's creed. 2007.

WANG, T.-K.; DANG, Q.; PAN, P.-Y. Path planning approach in unknown environment. **International Journal of Automation and Computing**, Institute of Automation, Chinese Academy of Sciences, co-published with Springer-Verlag GmbH, v. 7, n. 3, p. 310–316–316, aug. 2010. Available from Internet: <http://dx.doi.org/10.1007/s11633-010-0508-6>.

WOLINSKI, D. et al. Parameter estimation and comparative evaluation of crowd simulations. **Comput. Graph. Forum**, v. 33, n. 2, p. 303–312, 2014. Available from Internet: <http://dx.doi.org/10.1111/cgf.12328>.

WU, C.-P.; LEE, T.-T.; TSAI, C.-R. Obstacle avoidance motion planning for mobile robots in a dynamic environment with moving obstacles. **Robotica**, Cambridge University Press, New York, NY, USA, v. 15, n. 5, p. 493–510, 1997.

YEH, H. et al. Composite agents. In: **Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008. (SCA '08), p. 39–47. Available from Internet: <http://portal.acm.org/citation.cfm?id=1632592.1632599>.

YERSIN, B. et al. Real-time crowd motion planning: Scalable avoidance and group behavior. **Vis. Comput.**, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 24, p. 859–870, September 2008. Available from Internet: <http://portal.acm.org/citation.cfm?id=1410249.1410252>.

YERSIN, B. et al. Crowd patches: populating large-scale virtual environments for real-time applications. In: **Proceedings of the 2009 symposium on Interactive 3D graphics and games**. New York, NY, USA: ACM, 2009. (I3D '09), p. 207–214. Available from Internet: <http://doi.acm.org/10.1145/1507149.1507184>.

ZHANG, Y.-P. et al. Resolving local minima problem of potential field. In: **International Conference on Virtual Reality Continuum and Its Applications in Industry**. New York, NY, USA: ACM, 2010. (VRCAI '10), p. 339–346. Available from Internet: <http://doi.acm.org/10.1145/1900179.1900250>.