UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA
CIÊNCIA DA COMPUTAÇÃO

JONAS HARTMANN

Real-Time Path Planning in Large Dynamic Environments based on a Voronoi Diagram

Projeto de Diplomação

Profa. Dra. Luciana Nedel
Advisor

Francisco de Moura Pinto
Coadvisor

Porto Alegre, December 2010

ACKNOWLEDGEMENTS

I wish to send the most warm thanks to my family whose support during my whole life was very important for me to achieve an adult life with great confidence, health and respect. I am also grateful for the teachers I had during my life. They taught me great values. I am also very thankful for Daniela's patience and support. She understood why my time was so short.

This work is the result of a continuous research of a strong team. For this I am thankful to Professor Luciana P. Nedel, whose vision and devotion instructed me how to successfully fulfill the tasks required to achieve this result, and Francisco M. Pinto for all the help, inspiration and criticisms, which improved my writing and knowledge. Last but not least, I wish to thank the Brazilian Federal Government for supporting a high quality university which opens opportunities for all kind of students. With the help of the Brazilian foundation Coordenação de Aperfeiçoamento de Pessoal de Ensino Superior (CAPES), I was able to participate in an exchange program that gave me educational experience and life long teachings, which facilitated this work's accomplishment. For this I thank CAPES and the communities of the Universidade Federal do Rio Grande do Sul, Instituto de Informática and Technische Universität Berlin.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

GVD	Generalized Voronoi Diagram
VD	Voronoi Diagram
UFRGS	Universidade Federal do Rio Grande do Sul
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Ensino Superior
RTS	Real Time Strategy
SDK	Software Development Kit
AI	Artificial Intelligence

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Path planning is the ability to find a path free of obstacles from an arbitrary initial position to a given final position. One constraint when finding this path, is the size of the environment in which the search is made. Large environments increase the computational time of algorithms to an unwanted level. In this work, we address the problem of path planning in large environments. Our approach is based on a recent implementation of a high performance algorithm for the creation of a dynamic voronoi diagram for complex sites. We use it as the underlying structure for path planning. Our approach is able to produce smooth paths with arbitrary clearance.More than this, it handles dynamic obstacles and allow space for steering behaviors. Our results show that it is possible to produce, in real-time, smooth paths with arbitrary clearance in large dynamic environments.

Keywords: UFRGS, path planning, smooth path, multi-agent, voronoi diagram, large scale environments, games.

Planejamento de Caminho em Grandes Ambientes em Tempo Real baseado no Diagrama de Voronoi Dinâmico

RESUMO

Planejamento de caminho é a habilidade de encontrar um caminho livre de obstáculos a partir de uma posição inicial arbitrária até uma posição final dada. Uma limitação no momento de procurar este caminho, é o tamanho do ambiente onde a busca é realizada. Grandes ambientes aumentam o tempo computacional de algoritmos para um nível indesejado. Neste trabalho, abordamos o problema do planejamento de caminho em grandes ambientes. A nossa abordagem é baseada em uma implementação recente de um algoritmo de alta performance para criação de um diagrama de voronoi dinâmico para sítios complexos. Usamos este como uma estrutura para o planejamento de caminho. A nossa abordagem é capaz de produzir caminhos suaves com afastamento arbitrário. Mais do que isto, ela lida com obstáculos dinâmicos e permite espaço para comportamentos que mudem a direção de movimento. Nossos resultados mostram que é possível produzir, em tempo real, caminhos suaves com afastamento arbitrário em grandes ambientes dinâmicos.

Palavras-chave: planejamento de caminho, caminhos suaves, multi-agentes, jogos, diagrama de voronoi, grandes ambientes, UFRGS.

1 INTRODUCTION

The games industry is one of the largest entertainment industries in the world. Games are interactive and therefore rely mostly in real-time solutions. One of the bottlenecks of games, are the artificial intelligence(AI) components. At first, games did not use many AI algorithms, because they required more processor speed and memory than it was available. With improvements in hardware, more and more AI solutions were able to be used. Therefore, the area received a special interest from the big industry.

Although the hardware and algorithms improved, the AI is still a problem when it comes to real-time solutions. Planning paths for virtual autonomous characters in real-time is still a challenge. We understand a path as going from one point to another without intersecting any obstacle in the way. There are several different approaches for virtual agents path planning (LAVALLE, 2006).

There is also a necessity to produce high quality paths, because players want the virtual characters to behave as real as possible. In this meaning, virtual human characters should walk like humans. Those paths are always calculated within a given environment. In this environment, different obstacles appear. A game environment can have static and dynamic obstacles. Static obstacles represent the environment part which does not change often, as for example virtual buildings. Dynamic obstacles, on the contrary, represent those obstacles which change their position or shape often, like the virtual characters themselves for instance.

There are several intrinsic difficulties to dynamic obstacles and ever growing environments. If a path planner is going to treat them, it should keep special attention to the desired frame-rate. Our goal is then to achieve high performance path planning in these large dynamic environments. For this, we decided to have a solid structure that would allow us to keep track of dynamic obstacles and represent the large environments.

In this work, we propose an approach to deal with the problems in path planning mentioned before. It deals with path planning in large scale environments filled with static and dynamic obstacles. More than this, natural-looking paths are produced in real time. To achieve this, we use a recently proposed dynamic voronoi diagram for complex sites (PINTO; FREITAS, 2010). Pinto's contribution is that the voronoi diagram is dynamic, because it performs local updates, and analytical, instead of being discrete as when the graphics hardware is used. We believe that in order to achieve real-time solutions for path planning, a robust underlying structure is needed. This structure should account for proximity queries and allow the implementation of reactive behaviors (REYNOLDS, 1999). The rest of this work is organized as follows.

In chapter 2 we start by showing some related work. We discuss voronoi diagram-based algorithms, illustrate the use of potential fields and review the latest approaches using navigation meshes. We also give a glimpse over comercially available path planning middleware software. After that, in chapter 3, we give a brief description of the dynamic voronoi algorithm used in this work. We present its general idea and show its properties of most importance when building our solution. On chapter 4, we explain our approach to path planning. We show how we deal with obstacle avoidance, path finding with arbitrary clearance, large environments, smooth paths and steering behaviors. At chapter 5, we show the results achieved and interpret them. Finally, we draw our conclusions and analyze further improvements and possibilities at chapter 6.

2 RELATED WORK

In this chapter we show related work in the area. Path planning is an active area of research for many years already, therefore it is not our intention to perform a complete bibliographic research. Thus, we will illustrate only the most important works that are closely related to ours.

The Voronoi Diagram is the conjunction of all the points that are maximally far from their surrounding sites. To accelerate the computation of generalized voronoi diagrams, the graphics hardware was used. Therefore, allowing the use of a voronoi diagram for path planning (HOFF III et al., 2000). But it was still a discrete aproximation of the diagram. Later, a better aproximation of the generalized voronoi diagram has been used to construct a roadmap. It was considered "computing the Voronoi Diagram in an exact manner infeasible in realistic environments where obstacles can have arbitrary shapes" (NIEUWENHUISEN; KAMPHUIS; OVERMARS, 2007). Higher order discrete voronoi diagrams calculated using graphics hardware were used to achieve navigation graphs for multi agents, (SUD et al., 2008). Still using a voronoi diagram approach, but aiming at producing smooth paths, a technique which uses composite bezier curves was introduced (HO; LIU, 2009). This latter approach does not deal with dynamic obstacles. If the reader would like to know more about the concepts behind voronoi diagrams and its applications, please refer to the following book (OKABE et al., 2000).

Another approach to solve the path planning problem is using the potential fields technique (HWANG; AHUJA, 1992). One of the biggest problems with potential fields is the local minimum problem. This problem appears frequently in mazes, but not only. Recent research was made to solve it (MABROUK; MCINNES, 2008; SILVEIRA et al., 2010). Silveira et al. showed that the paths produced by this technique can be very similiar to human paths. This kind of approach was also used for crowd simulations with collision free path planning (TREUILLE; COOPER; POPOVIĆ, 2006).

Another crowd simulation approach is the division of the environment in blocks called 'Crowd Patches'. Those patches are used to handle large crowd simulations. Each block contains intern pre-calculated paths and connections to other blocks. The global path is then a connection of many patches (YERSIN et al., 2009). The resulting paths are good for simulation purposes, but the solution is not able to handle goal-based interactivity. Therefore, it is not suitable for player controlled autonomous characters.

Navigation meshes refer to the use of any type of polygonal mesh to represent walkable areas in a given environment. A more specific navigation mesh used to perform navigational queries are the ones made with triangles. Such triangulations

allow a variety of important navigational queries to be made with a complexity related to the number of segments used to describe the world. A special triangulation called Local Clearance Triangulation (LCT) was used to efficiently compute locally shortest paths with arbitrary clearance. The Local Clearance name refers to the fact that the triangles constructed by the method are free of obstacles. The LCT requires a pre-computation of $O(n^2)$ to achieve high quality paths very efficiently. The LCTs are also not calculated for dynamic obstacles, therefore other techniques should be employed to treat them (KALLMANN, 2010).

The next paragraphs are dedicated to comercially available solutions for path planning.

The PathEngine is a comercial pathfinding library licensed for many games being currently developed. Their core is, in their words:

> "A powerful and well-defined motion space description (essentially: augmented navigation meshes for ground management + static and dynamic obstacle detail + automatic expansion by pathfinding agent shape).
>
> Paired pathfinding and collision, for efficient and effective reactive behaviours and fundamentally robust movement planning.
>
> Fast pathfinding over long distances, with no aliasing."

They achieve fast pathfinding over long distances by the cost of a preprocessing phase. Their preprocess is made per agent shape. Depending on the size and complexity of an environment, it can cost a lot of memory to save the preprocessment data. Because, for each agent size, they need to save the preprocessment data for the whole environment. Therefore, an application using their planner should take an extra consideration about their agent size variation so as to reduce the total number of preprocess computation and memory footprint.

Their main pathfinding queries are the shortest path from a known start position to a known end position and the shortest path away from a region around a specific position. The second can be used as basis for a "run away" behaviour.

The path returned by the shortest path queries does not look natural, having straight lines and sharp turns at corners. Therefore, they have another phase in which they smooth the path, generating curves at corners.

Their solution allows collision queries for different cases:

- Points collision;
- Lines collision;
- Directional movement;
- Overlapping agents.

The collision query for directional movement will return true if there will be a collision when moving in the given direction (PATHENGINE, 2010).

Havok^{TM}AI is an SDK with support for efficient path finding for dynamic game environments. Their focus is on automatic navigation mesh generation. They have optimized solutions for runtime queries, such as single and multiple goal pathfinding and proximity queries. Their solution is multithreaded without restriction on mesh size. They also provide a system called navigation mesh streaming, which allows the creation of different navigation meshes for different parts of the game world, for later

connecting them on-the-fly (HAVOK, 2010). There were not enough details about its algorithms.

We propose a new approach because we want to fill the gaps found in the related works. We want to be able to handle large environments together with dynamic obstacles. We also want to produce smooth paths that have an arbitrary clearance from obstacles. More than this, we want to do all of this in real-time. So, even though there are very good works which deal with one or two of this problems at a time, our goal is to be able to deal with all of them at once. Therefore, we will start with an overview about the underlying structure of our algorithm.

3 DYNAMIC VORONOI DIAGRAM

The Voronoi Diagram is the conjunction of all the points that are maximally far from their surrounding sites. A site is here defined as being a point, line segment or polygon. Most of the voronoi diagrams are defined only for points. For this reason, when refering to voronoi diagrams which are defined for any shape of site, the use of the word "generalized" is common. The Generalized Voronoi Diagram is also called a maximum-clearance roadmap (LAVALLE, 2006). Clearance means here the free space away from obstacles. Therefore, in a maximum-clearance roadmap, any point of the roadmap is as far away from any obstacle as it could be. In the rest of this work, we will call the Generalized Voronoi Diagram simply as Voronoi Diagram.

For the complete construction of the voronoi diagram, all sites are divided into subsites, which are vertices and segments. The voronoi diagram is then defined for the subsites, following these possibilities:

Segment-Segment: Voronoi Line Segment

Vertex-Vertex: Voronoi Line Segment

Segment-Vertex: Voronoi Parabole Segment

Connecting these possibilities, we are able to build the voronoi diagram for points, line segments and polygons (PINTO; FREITAS, 2010). In Figure 3.1, we see the voronoi diagram as a line between two segments (figure 3.1a), line between two points (figure 3.1b) and parabole between a point and a segment (figure 3.1c).

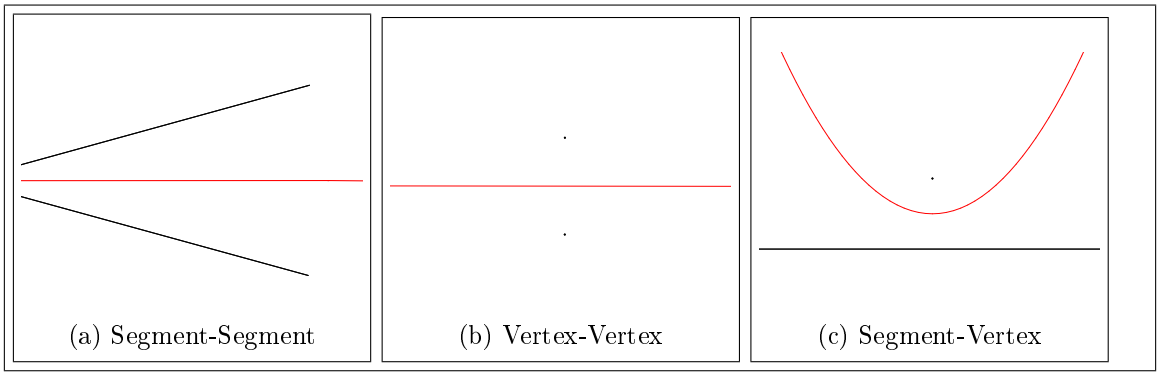


Figure 3.1: Voronoi diagram simplification. The red lines represent the voronoi segments. Black lines and points represent subsites. (a) Two segments define a voronoi line segment. (b) Two vertices also define a voronoi line segment. (c) A segment and a vertex define a voronoi parabole segment.

The main contribution of Pinto is the possibility of removal of complex sites (lines and polygons) and voronoi diagram update in interactive time, therefore called Dynamic Voronoi Diagram. The efficient insertion of sites is already a well known operation. With the insertion and removal defined, it is possible to move the sites, therefore allowing dynamic movement. More than this, with a real-time update, we can have the correct voronoi diagram even in the presence of those dynamic sites.

The dynamic voronoi diagram is analytically defined and its update is done locally. Therefore, it achieves high frame rate even if the complete environment has more than a million points. There are two phases in the diagram update: the update of the sites and the update of the diagram.

The first phase is the insertion and removal of sites. On the insertion, the inserted site just goes to the quadtree, which armazenates the sites with a geometrical distribution. On the removal, the site is removed from the quadtree and all the sites, which define an edge with the removed one, are notified - this other sites are called pair sites. Every edge of the voronoi diagram is defined by two sites which are pair of each other. A site has a structure associated with it which lists every edge of the diagram defined by it and also the pair sites with whom those edges are defined. This structure allows, in case of deletion, the removal of all invalid edges of the diagram and the elimination of the deleted site from the list of its pair sites. Both the insertion and the deletion make the diagram be incomplete and require it to be updated. Every inserted site or that had one of its pair sites removed are marked to be the starting point for the diagram update.

For every site marked in the previous process, a radial search in the quadtree starts at the node which contain the site (more than one node can contain the same site). The search goes through the imediate neighbors in the quadtree and expands itself gradually until all neighboring sites with whom the site in focus define valid voronoi edges are analyzed. The search stops when the diagram is complete in reference to the voronoi edges defined for the focused site. When all previously marked sites are submitted to this search process and edge creation, the diagram update will be completed.

The dynamic voronoi diagram has "an exact, continuous-space efficient solution [...] involving from points to polygonal sites" (PINTO; FREITAS, 2010). That is the reason why it can be used for representing virtual worlds, where a lot of different obstacle shapes are involved. Below is the illustration of some important properties for the use of the voronoi diagram as an underlying structure for path planning. It is not our purpose here to fully describe all the properties that a voronoi diagram has.

Property 1. *For every position C in the voronoi diagram, there is a circle tangent to at least two sites with the center exactly at C. This circle does not include any other site within its borders, and is therefore called Empty-Circle.*

Property 2. *A voronoi vertex is defined by at least three generator sites. The empty-circle, whose center is at the voronoi vertex, is tangent to all those sites.*

Property 3. *The union of all empty-circles defined by the diagram covers the complete environment.*

Property 4. *The voronoi diagram is connected as a graph, where voronoi vertices are nodes and voronoi segments are edges.*

This was just a simple overview about the dynamic voronoi diagram, with just the necessary for the understanding of this work. We refer the reader to the original article for a more extensive explanation (PINTO; FREITAS, 2010).

4 PATH PLANNING ALGORITHM

Path planning using a roadmap can be split into four phases:

1. World modeling
2. Roadmap construction
3. Path finding
4. Path smoothing

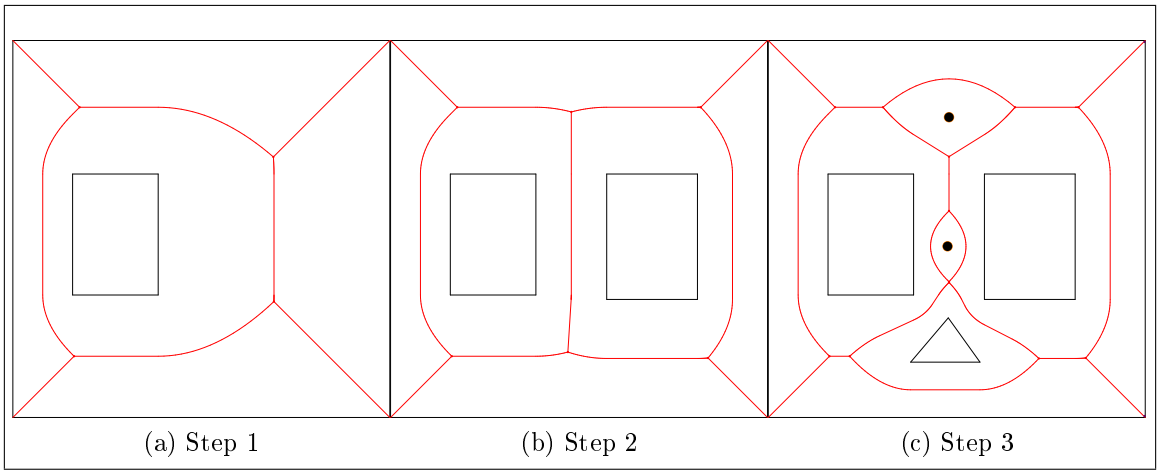
In the next sections we will show how our algorithm handles each phase.

4.1 World Modeling and Roadmap Construction

A world model is usually built once and then modified through the life of the application. In our solution we allow the world to be designed with points, lines and polygons. For instance, a building could be a polygon and a single lamppost could be a single point. The person who is modeling it, the designer, have the freedom to create any polygon in a planar environment. It is important to pay attention to specific details when modeling an agent in the environment. Those details are for example its size, or shape. In robotics, for example, if we have a two meters square robot, it would not be advisable to model it as a half meter line, otherwise we would lose the ability to predict when it would hit obstacles. Therefore, we decided to model our agent as having a center point and an arbitrary radius. Describing the size and shape of the agent with a circle has enough accuracy and allows fast collision checks.

As we create the world model by adding points, lines and polygons, the dynamic voronoi diagram is calculated at the same time. This works because we can update, at an interactive rate, the diagram with every new object insertion. Therefore, at the end of the modeling phase, we have already finished the second phase, the roadmap construction. This leads us also to the ability of adding content to the world model in real-time. We could use it to create exploration in unknown worlds, where the world would grow as we explore it. This mechanism can be seen in figure 4.1, where three moments in time are shown. It is important to notice that the agent is part of the diagram as an obstacle.

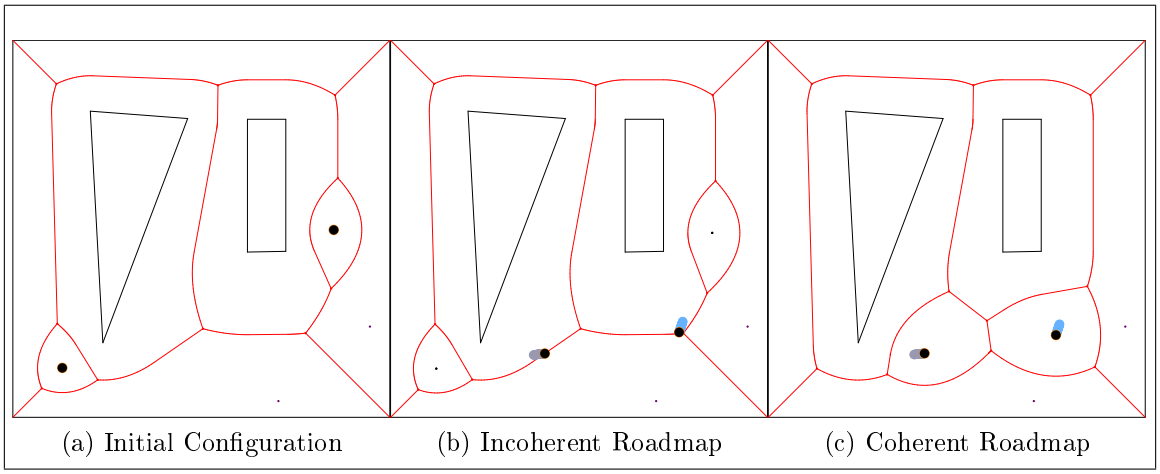
For static environments it is enough to build a roadmap just once. But for dynamic environments, this solution does not work. The roadmap should be constantly updated, as the environment changes. More than this, the update should be done as



(a) Step 1 (b) Step 2 (c) Step 3

Figure 4.1: Update of the dynamic voronoi diagram. The red lines are the voronoi diagram, the black lines represent the obstacles and the black circles are the agents. In the first step, we insert one obstacle and show how the dynamic voronoi diagram updates to accomodate it (a). Then, another obstacle is inserted (b). Finally, a triangular obstacle and two agents are inserted (c).

soon as the changes happen, so we do not have incoherent roadmaps, which represent an old moment in time (Figure 4.2). With this comes the problem of having a real-time update. This is specially the case when we have large environments filled with dynamic obstacles. The roadmap update in this case would probably take more time than we have available. To solve this problem, we use the dynamic voronoi diagram implementation to address the roadmap calculation and consequently updates, as implemented in (PINTO; FREITAS, 2010). It gives us means to always maintain a coherent maximum-clearance roadmap, even within large environments filled with dynamic obstacles. More than this, we have a continuous and exact representation of the world with it.



(a) Initial Configuration (b) Incoherent Roadmap (c) Coherent Roadmap

Figure 4.2: Incoherence. At the initial configuration, there are two agents with different goals (a). At a given time, the agents moved, but the roadmap representation was not updated (b). Therefore, the figure shows the worst case scenario, where an agent is blocking a path that should be free. The other figure shows the diagram produced when the update is made (c).

4.2 Path Finding

Path finding means, for us, solving the problem of finding a path from an arbitrary start position to a defined end position. To perform this with a roadmap, an algorithm should first connect the start and goal positions to it, and then later search for a path connecting both through the nodes and edges of the roadmap.

In our method, the algorithm firstly connects the start and goal positions to the dynamic voronoi diagram. After this, it uses the diagram as a graph to find the shortest path between the connections. We will start by explaining how to connect the goal and then we will explain how to connect the start position. We can see that the voronoi diagram performs just like a roadmap.

4.2.1 Connecting the start and goal positions

A valid goal position will always be situated inside a voronoi cell. The goal position will be connected with one of the voronoi segments of the cell. The shortest path from the connection point to the goal position is a straight line. Therefore, for us to choose the shortest connection, we need to find a point in the voronoi segment and make sure that the straight line connecting the point to the goal does not pass through any obstacle. To find a point in the diagram, which fulfills these requirements, we use the empty-circle property of voronoi diagrams, shown in section 3. A simplified algorithm for finding the best empty-circle is shown in figure 4.3. The center of the best empty-circle will be the connection point.

```
1 For every segment of the voronoi in the relevant neighborhood:
2  Find the empty-circle which best embeds the goal position
3 End For
4 Return the best empty-circle found
```

Figure 4.3: Choosing the best embedding of the goal position.

The relevant neighborhood in line 1 means that we search only in segments of the voronoi cell which contains the goal position. That is because the other segments are farther away and searching through them would be meaningless. On line 2, the circle which best embeds the goal position means that the goal is further from the edge of this circle than to others. This search for the best circle is an analytical inference, which means it is exact and has high performance. At line 4, the algorithm returns the best empty-circle found inside the loop. This is just a simplified explanation of a feature implemented by Pinto that is very important for this work. Figure 4.4 illustrates the choice for the best empty-circle.

We could use the same technique mentioned above to connect the start position. It would involve taking the agent out from the diagram, and then performing the same task as when connecting the goal. In the beginning of the project, we thought it was necessary to do this in order to find a path. Our algorithm first took the agent out of the voronoi diagram, connected it and then searched for a path through the voronoi vertices. After returning the path it would then search for the first empty-circle, in the direction of the path, tangent to the agent, which gives the agent a safe (free of obstacles) step destination towards the center of the circle. An explanation about why we decided to make the agent follow the empty-circles is given later at section 4.3. The older algorithm to find this circle was simple but involved a series

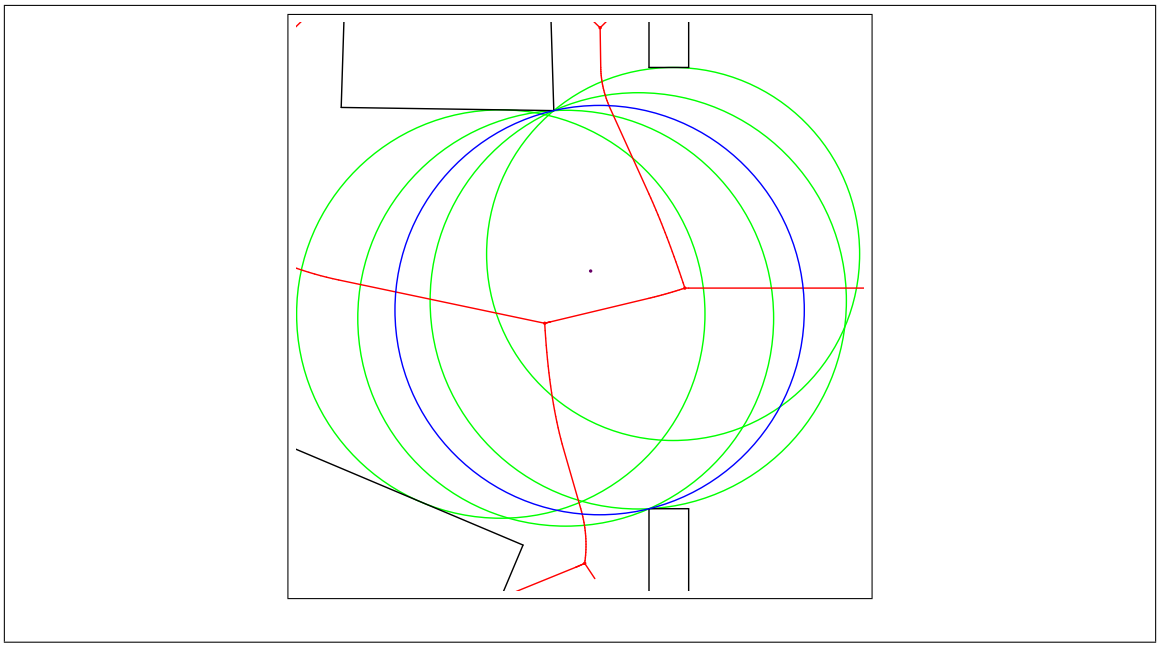


Figure 4.4: Best embedding empty-circle. The best empty-circle embedding the goal position is shown in blue. The goal position is shown as a purple point. The green circles also embed the goal position, but the goal is closer to their edges than it is to the edges of the blue circle.

of problematic tasks, because of their high risk of programming errors. But with the initial results, we were able to create a better technique to connect the position of the agent to the voronoi diagram. All empty-circles tangent to the agent have their centers exactly at the voronoi vertices of the diagram when the agent is still considered as an obstacle. The logic behind it is that every empty-circle of a voronoi vertex is tangent to at least three obstacles, otherwise it would not be a voronoi vertex. Therefore, all voronoi vertices which are directly defined by the agent have empty-circles tangent to it. We can see it clearer by the representation in Figure 4.5. It is good to notice that we consider the agent's position as being its center. Therefore, the circles are tangent to the center point and not the outter radius.

The new technique goes as follows. Leaving the agent as an obstacle, it generates a cell in the voronoi diagram with at least two voronoi vertices. Since the space between the agent and any of its surrounding voronoi vertices is guaranteed to be obstacle-free by the voronoi diagram definition, we can consider every one of the vertices of the agent cell as being a connection to the voronoi diagram. Therefore, the first step of the algorithm is to get all the voronoi vertices that are around the agent. This is achieved by getting all the voronoi vertices defined by the site which represents the agent. The next step is to add all of them as starting vertices to the graph search algorithm. This new technique has a major advantage over the older one, which is the reduction of one voronoi diagram update per agent step. To move the agent one step we can insert it at the new position just after removing it from the last position. There is no need for an update of the diagram between these two operations. For this technique to work fine, the costs of the starting vertices should be well calculated. We will discuss how we calculate them later, at section 4.2.2.

The search for the shortest path can start as soon as we have the start vertices and the end vertices. We already talked about the start vertices, but we only showed

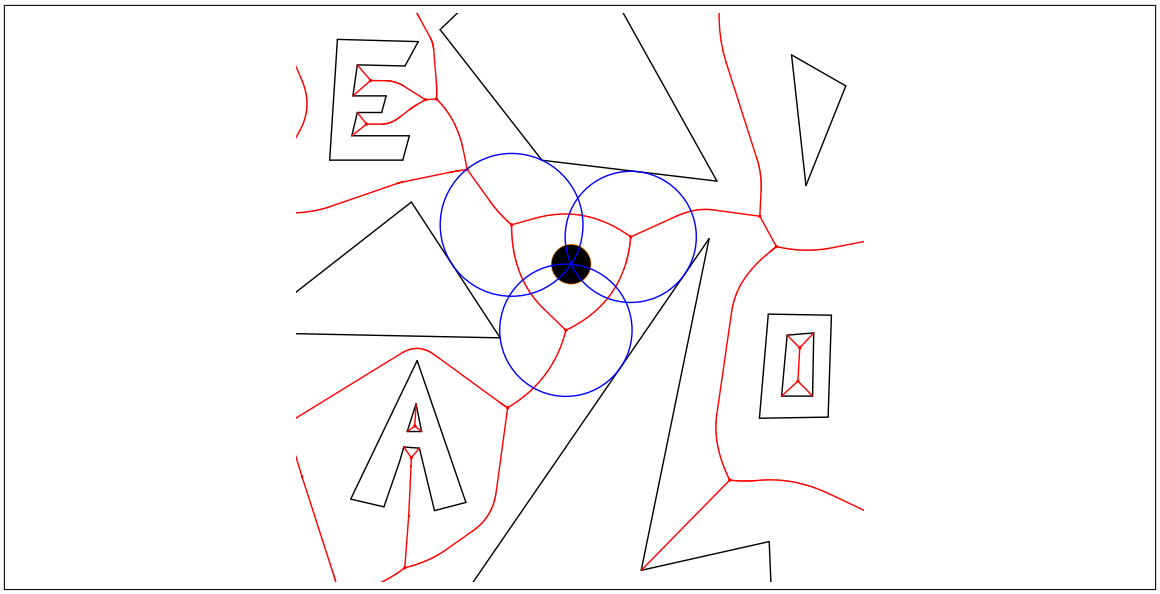


Figure 4.5: Empty-circles tangent to the agent with their centers exactly at the neighboring voronoi vertices are shown in blue. The black circle in the middle is the agent. The red lines represent the voronoi diagram. The black lines represent the borders of the obstacles.

how to find the goal connection and nothing about how to get the end vertices. That is because it is a simple task, the algorithm just needs to get both vertices of the segment on which we have the connection point.

The pathfinding algorithm we used will be seen in the next section.

4.2.2 Pathfinding with A*

The algorithm used to search the graph for the shortest path was the A*. It is a classical pathfinding algorithm. If the reader does not know the algorithm, a clear visual explanation of it can be found in (PATEL, 2010). The following optimizations were added to the original algorithm:

- The open nodes are armazenated in a multiset using an implementation from the C++ standard library
- There is the possibility to stop the execution after a given amount of iterations
- An edge can be considered as non-existent if the radius of its smallest empty-circle is bigger than the size of the agent. This will be discussed again in section 4.4.1

In our case, the use of an A* has some advantages. The first motivation is its simplicity and high performance. To achieve a safe navigation in dynamic environments, the path must be recalculated every frame. Therefore, it is mandatory the use of a simple but efficient algorithm. Another important ability the algorithm give us is the possibility of breaking the calculations at an arbitrary moment. It is something valuable when dealing with large environments, specially with many agents at the same time. In real time applications, we do not want our agents to be stalled until the algorithm returns the complete path. Most of the time it is enough to calculate

just a part of it and then making the first step. It is specially true when all the calculations are thrown away in the next step, as is our case.

With the voronoi diagram definition in a continuous space, the A* is exact and provides the shortest path from any vertex to another arbitrary vertex. That is because we have a continuous, uniform and isotropic space and therefore, the euclidian distance is the size of the shortest path possible.

The A* uses two different cost functions. The first one is the path cost since the start. The second is the estimate path cost until the end. It then expands nodes comparing the sum of both. Using the euclidian distance as the estimation for the second cost function we can assure the optimality of the path.

The cost of going from one voronoi vertex to another is the length of the segment connecting both.

As we said before, there are some details concerning the costs of the start and end vertices. The cost of an end vertex is the length from it until the goal connection point. The cost of a start vertex is the euclidian distance to the agent's position. We can see a representation of them in figure 4.6.

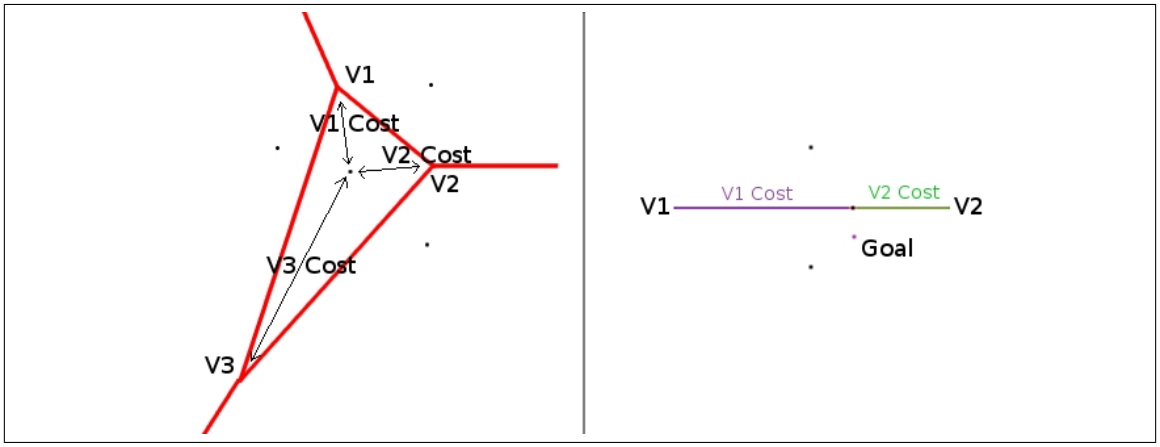


Figure 4.6: Representation of the start and end costs.

4.3 Smooth Paths

When roadmaps are used to find a path, the raw path following the roadmap vertices and edges does not look natural, with sharp curves and edges, specially the start and goal connections. Therefore, after calculating the path, another phase is necessary to smooth it.

In our solution, smooth paths are easily achieved, without this extra phase. We achieve it by directly using the information provided by the voronoi diagram. We decided to make the agent follow in the direction of the first voronoi vertex calculated by our planner. The reason behind it is that the empty-circle of the voronoi vertex is tangent to the agent. Therefore, the agent always moves towards the middle of an empty space, avoiding obstacles. More than this, while following the path, this voronoi vertex smoothly changes its position, respectively changing the empty-circle position, and guiding the agent towards the goal. We can see in figure 4.7 a smooth path followed by an agent. In the next section we will talk about steering behaviors which allow to perform a better navigation when confronted with more complex challenges.

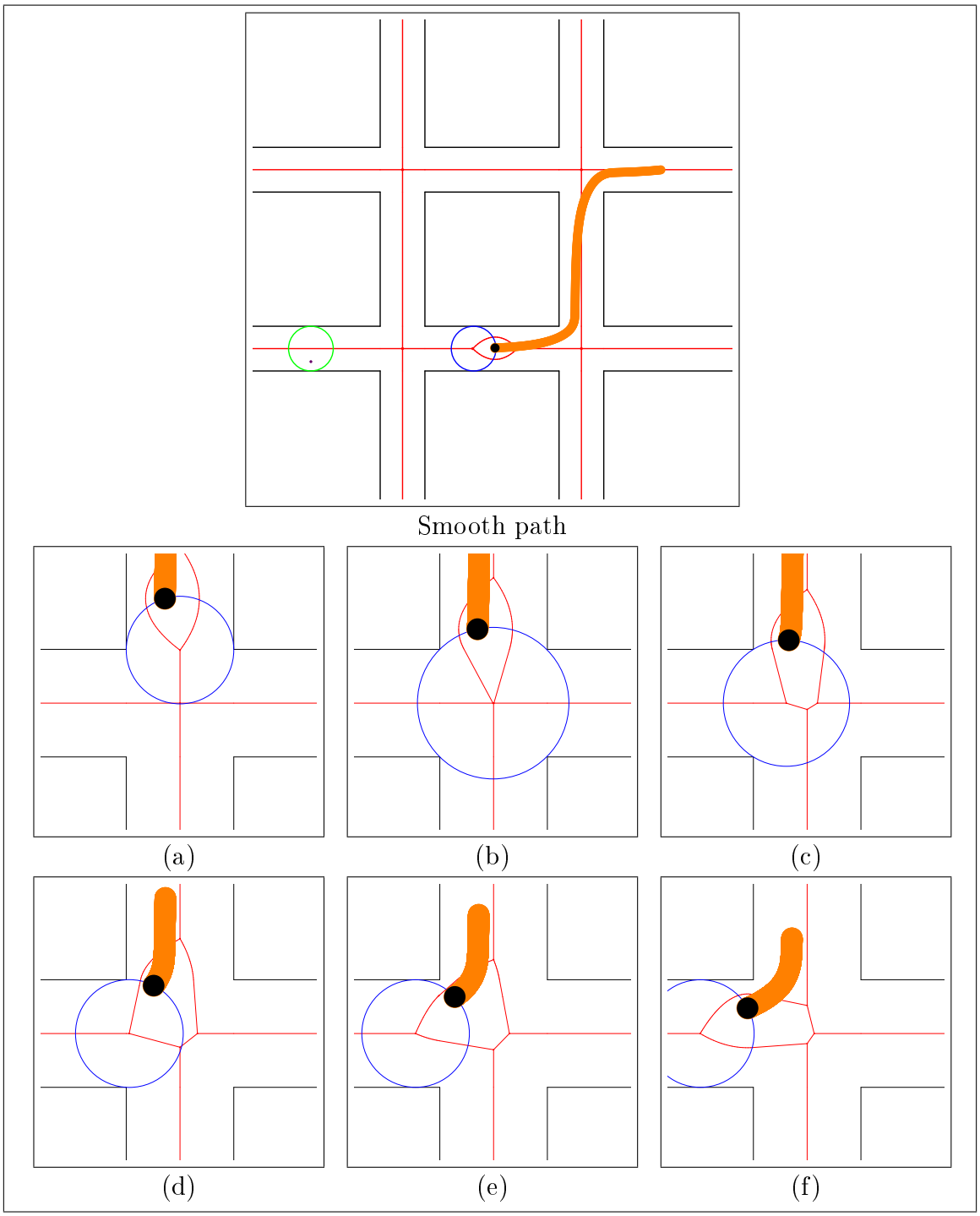


Figure 4.7: The path followed by the agent is shown in orange. The goal position is shown in purple with its empty-circle in green. The Voronoi Diagram is shown in red. The blue circle is the empty-circle of the leading voronoi vertex. The sequence shown from (a) to (f) is a zoom at the moment the agent made the second curve.

4.4 Features

In this section we show features that required small programming effort to be achieved.

4.4.1 Arbitrary Clearance

Games commonly involve more than one kind of agents. For example, soldiers and tanks in a RTS(Real-Time Strategy) game. They occupy different sizes in the world. Therefore, they should be represented with different sizes. In our implementation, we map the shape of the agent within a circle. Therefore, we can represent its size with that circle. Collision checks with circles are fast, it is just necessary to check the radius of both circles. But other shapes could also be handled, if different limitations for different directions were used.

Keeping it simple, smaller agents pass where bigger agents do not. This is a problem also often faced in robotics, where the location data of the robot is not always precisely equal to its place in real life. Therefore, it is advisable to find paths arbitrarily far from obstacles. To solve this in an efficient way, the path planning algorithm could treat it at the path finding level. We use the voronoi diagram information to create this kind of behavior. In possession of the size of the agent, the A* algorithm searches for a path that allows the agent to cross all segments on it. To know how much of free space there is when crossing a segment, we use its empty-circles. The size of the smallest empty-circle of a voronoi segment gives us the maximum size of the agent allowed to cross the complete segment. It is simple to find the smallest empty-circle of a voronoi segment (blue circle in figure 4.8). That is because it is analytically defined. The algorithm to find it was already implemented by (PINTO; FREITAS, 2010). With this information, the A* algorithm handles segments, which have smaller apertures than the size of the agent, as non-existents and only returns a path when the agent can cross all of its segments.

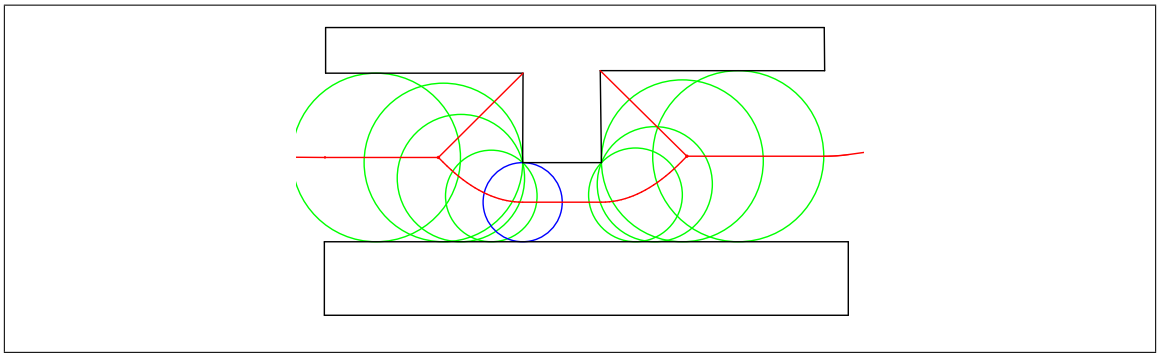


Figure 4.8: Smallest empty-circle. In this case, the smallest empty-circle found by a path traversing from left to right is the one in blue. The circles shown in green are other possibilities, but they have bigger radius.

4.4.2 Obstacle Avoidance

One of the most important reasons to have path planning is to find a path in a given environment which avoid obstacles. Using the voronoi diagram as a roadmap, we assure that our path will be clear. Because the voronoi diagram has vertices which are maximally distant from obstacles, as we can deduce from the first property shown

in section 3. We also have the information about all the empty-circles in the path, which create a free channel like the one in figure 4.9. So, with this free channel, it is possible to know exactly how far the obstacles are from any point, because it is defined in a continuous space. Therefore, it allows the agents to avoid obstacles in real-time.

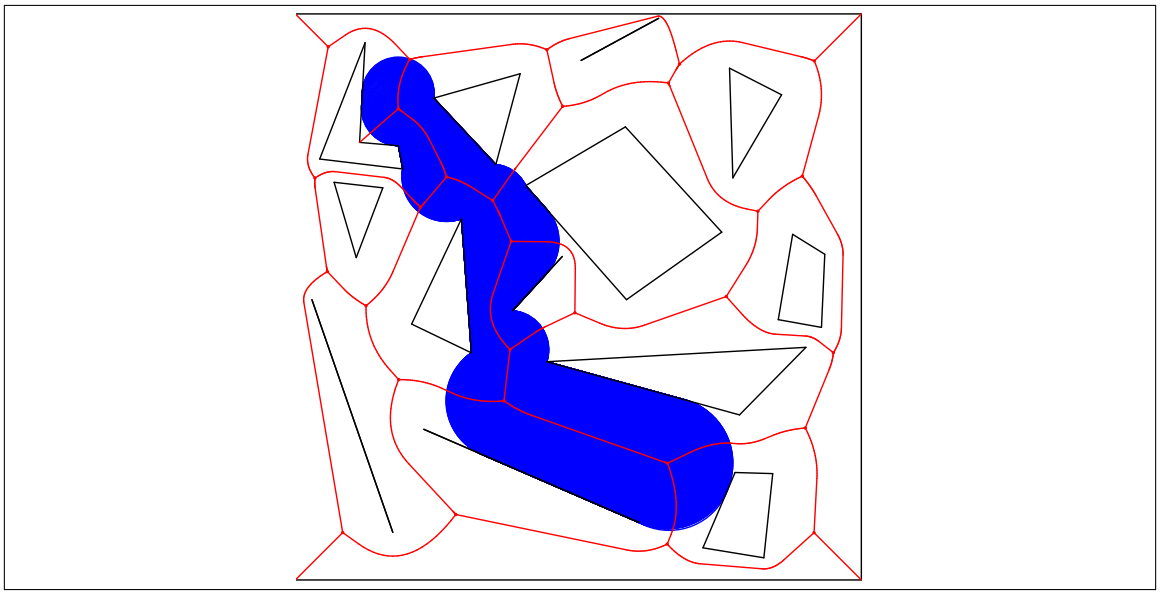


Figure 4.9: Voronoi free channel. It is made of the empty-circles of every point defined in the voronoi path.

4.4.3 Large Environments

Dealing with large environments is another concern for path planners. The bigger the environment, more data needs to be analysed. One of our contributions to solve this problem is to apply a fast and useful underlying structure. Instead of using a grid-based approach, we focus on a structure which gives us a lot of useful information and handles dynamic changes in the environment. This structure, the dynamic voronoi diagram, handles the updates locally.

For example, if the environment is a very large empty field, with only two houses on each side. The respective voronoi diagram will have a very small amount of vertices. Therefore, the path finding algorithm will be able to give quick results, as, regarding efficiency, it does not matter the length of the segments, only the number of voronoi vertices. We can see in figure 4.10 an example of such an environment.

With the same logic, a very complex environment will impose a higher difficulty. Nevertheless, techniques can be used to accelerate the pathfinding in those cases:

- **Break the A***: Return the best path calculated so far. The one that leads to the nearest found position to the goal. It can be done using a different range of metrics.
  - **Max iterations**: Stop when reached a maximum number of iterations. It is the same as stopping when expanded more than a certain amount of vertices.

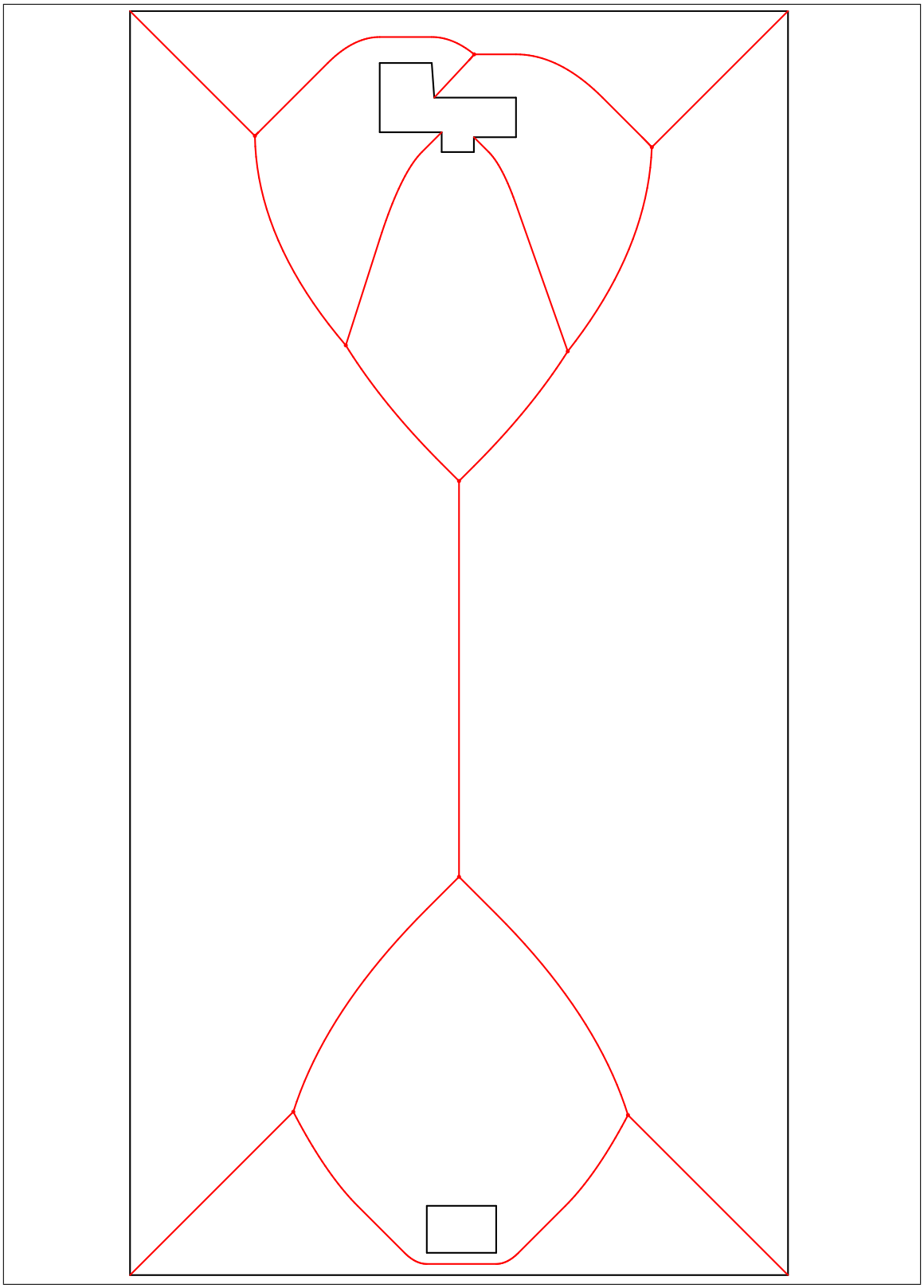


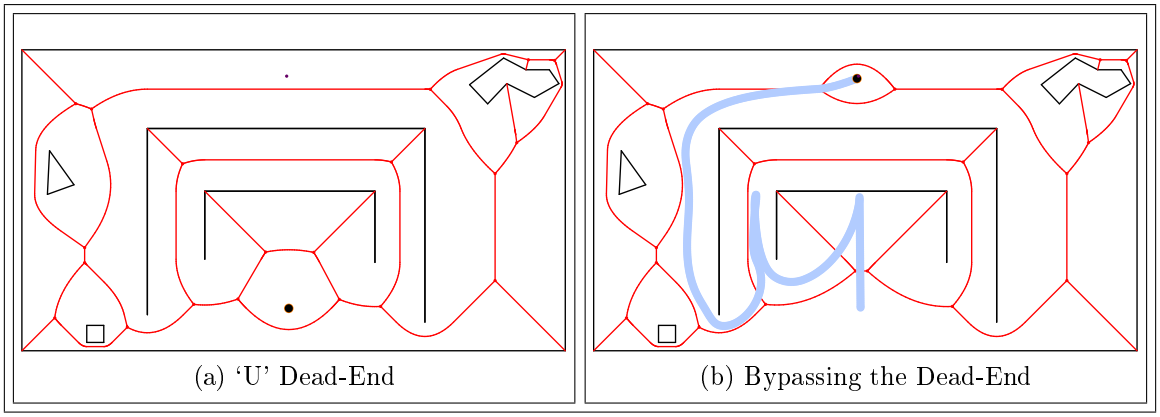
Figure 4.10: A large empty field.

  - **Time**: Stop when reached a time limit.

- **Block Streaming**: Stream blocks of the environment as the agent moves, therefore performing the path finding only in relevant areas.

When we think about games, we hardly have the case where the complete map of the environment is available for the path planner. It is usually the case that the map is being discovered as the agent travels. Therefore, the ability to add obstacles at run-time is of great value. Using our approach, we can achieve this, because the voronoi diagram updates at real-time. Therefore, obstacles can be added and we always keep a coherent maximal-clearance roadmap. This can also be seen as a block streaming technique, where the environment changes as the agent moves.

Breaking the A* before it finds a complete path brings another challenge, which is what we call "Dead-Ends". A dead-end is defined as a place where the straight forward path is cut by some obstacle. The classical example of it is the 'U' obstacle (figure 4.11a). The implemented technique to address this challenge is based on the perception of dead-ends. Therefore, we give it a technical definition based on the voronoi diagram. What happens is that when an agent reaches a dead-end, all voronoi vertices in its path have less than 3 incident edges. After the agent perceives a dead-end, it ignores its maximum number of iterations for computing a path through the A* algorithm until a voronoi vertex closer to the goal is found. We can see in Figure 4.11 an example where there are two obstacles in sequence with a high probability of creating dead-ends. Our agent was set with the maximum of 5 iterations for the A* algorithm, representing an almost blind agent. In Figure 4.11b we can see that our agent is able to perceive a dead-end and bypass it.



(a) 'U' Dead-End (b) Bypassing the Dead-End

Figure 4.11: Dead-End bypassing.

4.5 Behaviors

Having more than one agent in the same environment is common for games, but this is a problem for path planners. That is because they all need to find free paths and avoid hitting each other. There are some behaviors we added to our agent entity in order to overcome this and other problems. They will be explained in the sequence.

4.5.1 Empty-Circle

The first and most important behavior is the action of moving the agent in the direction of the center of an empty-circle. With this behavior, the agent moves to the interior of a free space, meaning it will get far from obstacles. The combination of path following with this behavior is perfect. That is because following the first voronoi vertex returned by the path finding algorithm is the same thing as going towards the center of an empty-circle tangent to the agent. This is derived from a property of voronoi vertices, shown at section 3. The sequence at Figure 4.7 uses this behavior. When the path of two or more agents cross, the size of the empty-circle will be updated respectively to those agents. Therefore, when all agents move to the interior of their empty-circles, they automatically avoid each other.

4.5.2 Space Negotiation

This behavior intends to keep agents away from hitting each other. It also works as a static obstacle avoidance behavior. In cases where the agent finds no path, this behavior is extremely important to keep away from dynamic obstacles that may want to pass over the position of the agent. It is a force that pushs the agent to the contrary direction of an incoming obstacle when the distance is smaller than a given threshold. If the agent is stopped and another agent pass very close to it, this force will move the stopped agent away from the incoming one. The incoming agent will also change its velocity respectively to the force which acts on it. The query for nearest obstacles was already implemented by (PINTO; FREITAS, 2010).

4.5.3 Goal Focus

There are cases where we want a more aggressive agent trying to reach the goal. For instance, the situation with pedestrians crossing a road. At a given moment, their goal is to reach the other side of the street. Then, the pedestrian semaphore turns to red. What is expected is that the pedestrians walk until the border of crossing and wait until the semaphore turns to green. In this example, when the semaphore closes, there is theoretically no path anymore to the other side of the street. Thus, the pedestrian, our agent, would stop wherever they are, without this behavior. With it, they walk until the border of the street, in direction of their goal. What keeps them from crossing to the other side of the street is the last behavior described. In the example given, the street would have been modeled as an obstacle.

5 RESULTS

In this chapter we show the results of our approach.

5.1 Test Environment Details

For simulation purposes we implemented the algorithm on top of the dynamic voronoi implementation, in C++ with OpenGL and GLUT to build the user interface. The results shown here were produced with an Intel Core 2 Duo CPU PC with Windows 7 as operating system, Nvidia GeForce GT 120M as graphics card and 3Gb of usable memory(RAM). Only one processor was used. Our simulation testbed is limited from -1 to 1 in x and y. Since we use double precision, we can represent pratically any environment by scaling it accordingly. In the simulations we represent 1 meter as the value 0.001. Therefore, 10 meters is 0.01 and so on.

5.2 Simulation 1

This simulation is to test the limits of our algorithm with respect to the number of agents.

- **Environment:** No obstacles, except the other agents;
- **Agents:** 20, 40, 60, 100, 200;
- **Goals:** Arbitrarily chosen;
- **StepSize:** Between 0.5m and 1.5m.

We can see in figure 5.1a the beginning of the simulation with 20 agents. The red lines represent the voronoi diagram. There are 10 black points on each line representing the agents. And 10 purple points per line representing the goals. We can see in figure 5.1b how the voronoi diagram dynamically adapts itself to the dynamic agents. Figure 5.1c shows the end of the simulation. The colorful lines represent the path each agent traveled. In figure 5.2a we can see the same simulation but now with 200 agents.

The tables shown in this section use the following structure:

Number of Agents. This is simply the number of agents of the simulation.

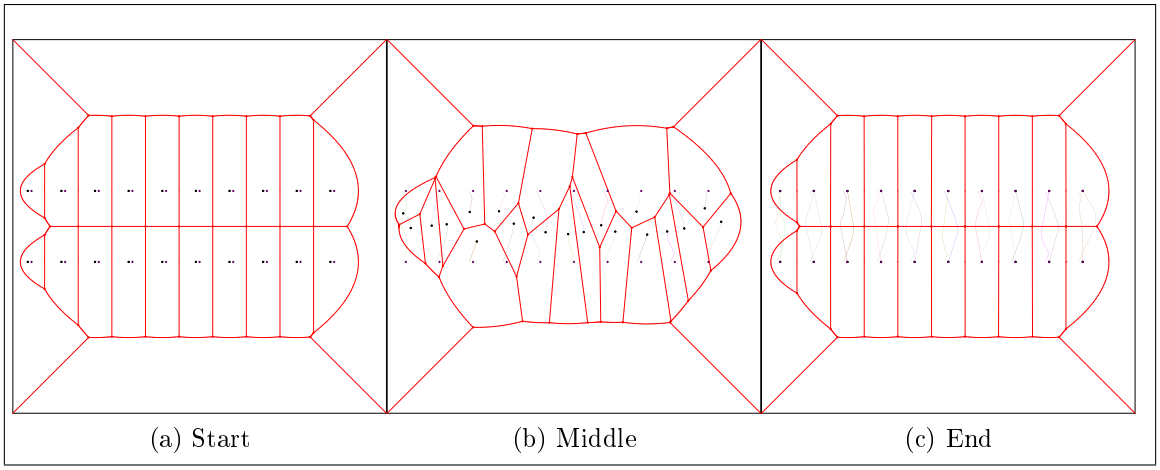
Max Number of Voronoi Segments. This is the maximum number of voronoi segments in the complete environment achieved during the simulation.

Mean Step Time Per Agent(s). This is the mean time, in seconds, that one agent takes to move one step. The mean is made over the data from the complete simulation.

Mean Iterations. This is the mean number of iterations taken to find a path for each agent in the simulation.

Maximum Length(m). This is the maximum size of a path, considering all the paths calculated in the simulation.

We can see on table 5.1 that the maximum number of voronoi segments increased according to the number of agents. This is because every agent is considered a site in the voronoi diagram. The number of iterations needed by the A* algorithm to find a path was influenced by the number of voronoi segments. Both of them contributed to slightly increase the mean step time. No optimizations of the A* were used in this simulation.



(a) Start (b) Middle (c) End

Figure 5.1: Simulation 1: 20 Agents

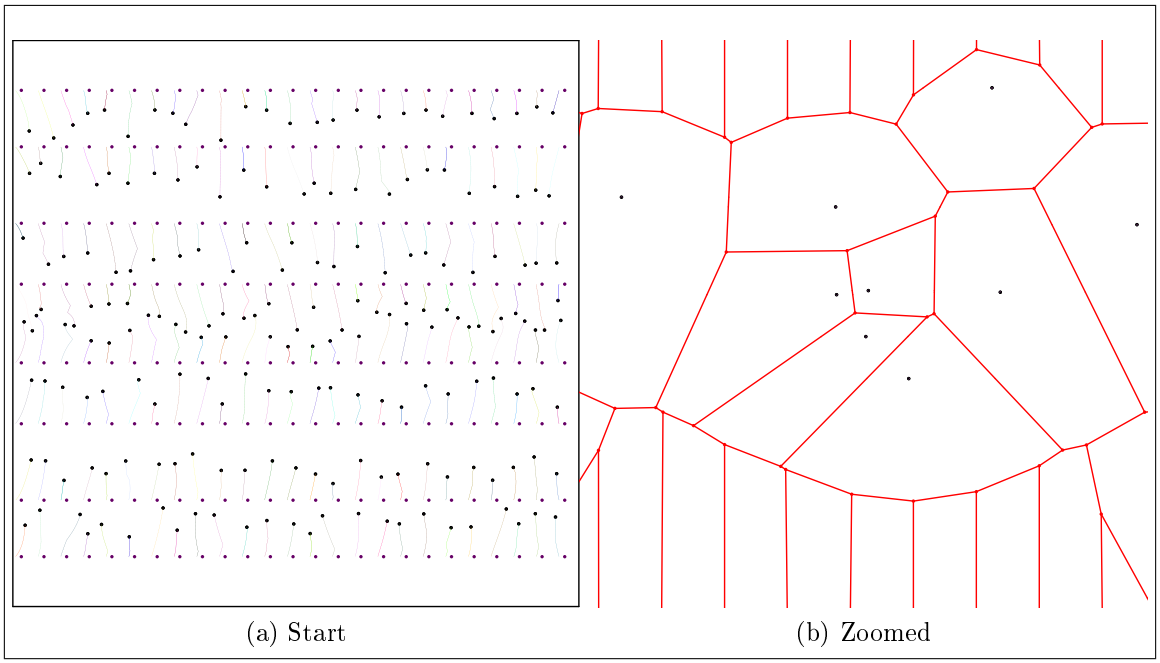
Table 5.1: Simulation 1

Number of Agents	20	40	60	100	200
Max Number of Voronoi Segments	212	430	682	1,280	2,031
Mean Step Time Per Agent (s)	0.0002	0.0003	0.0003	0.0004	0.0008
Mean Iterations	24.98	49.52	62.94	90.05	99.77
Maximum Length (m)	796.61	766.15	772.22	765.77	1,930.95

5.3 Simulation 2

The objective of this simulation is to show how the algorithm performs when confronted with an environment full of obstacles.

- **Environment:** 9,604 blocks of 15m x 15m separated by 5m
- **Agents:** 10, 20, 50



(a) Start (b) Zoomed

Figure 5.2: Simulation: 200 Agents. (a) The beginning of the simulation. (b) Zoom showing the voronoi diagram created by the agents (black points) at a specific moment.

- **Goals:** Randomly chosen around the center block
- **StepSize:** Between 0.5m and 1.5m

We can see in figure 5.3 two moments in the simulation. Both are a zoom at the city center. We can see a top view of the full city in figure 5.4. As shown in table 5.2, there are more than 100 thousand voronoi segments. Nevertheless the algorithm works at a high frame rate, with the mean step time for each agent below 1ms. That happens because all the voronoi diagram updates are done locally and only if something changed. In the next simulation we will see how it performs when there are many obstacles between the agent and its goal.

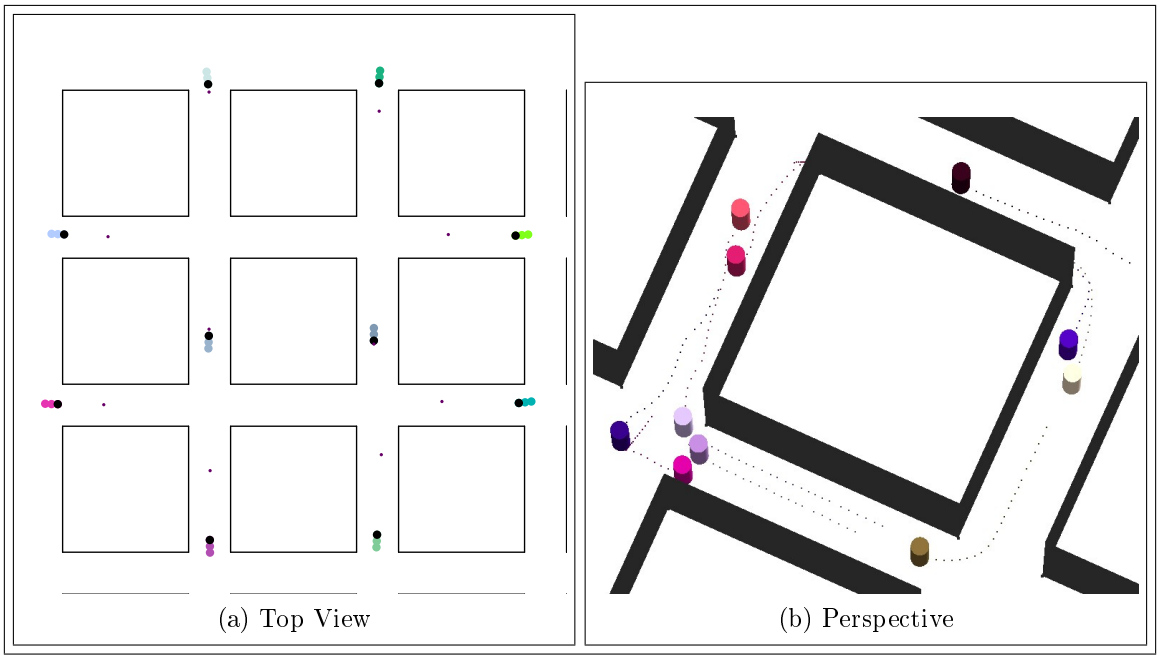
Table 5.2: Random Paths

Number of Agents	10	20	50
Max Number of Voronoi Segments	105,960	106,040	106,338
Mean Step Time (s)	0.0005	0.0005	0.0007
Mean Iterations	11	3.93	38.71
Maximum Length (m)	3.1	6.04	88.08

5.4 Simulation 3

This simulation is to show the performance when crossing the whole city environment.

- **Environment:** 9604 blocks of 15m x 15m separated by 5m



(a) Top View (b) Perspective

Figure 5.3: Random paths

- **Agents:** 1
- **Goals:** At the other side of the city diagonal
- **StepSize:** 1m

As we can see by the table 5.3, the complete city has more than 100 thousand voronoi segments. The first path calculated had almost 4km and needed more than 48 thousand iterations. Even with this high amount of work, the step time was kept between 30ms and 40ms. If we do not have more than one agent, than this is a reasonable time. Otherwise, it is possible to limit the number of iterations, therefore achieving faster results. We can see it on the next simulation.

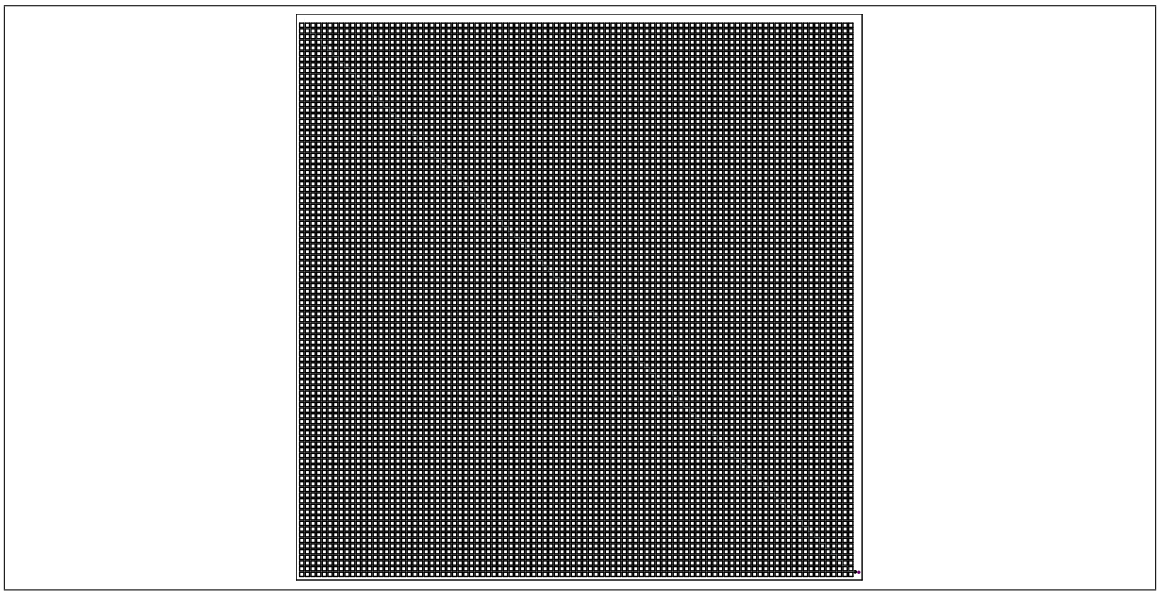


Figure 5.4: Path traversing a city with 9,604 blocks

Table 5.3: Large Environment

Number of Agents	1
Max Number of Voronoi Segments	105,877
Mean Step Time (s)	0.036
Mean Iterations	48,376.64
Maximum Length (m)	3,927.82

5.5 Simulation 4

At this simulation we mix the previous ones, performing simulations across the whole environment.

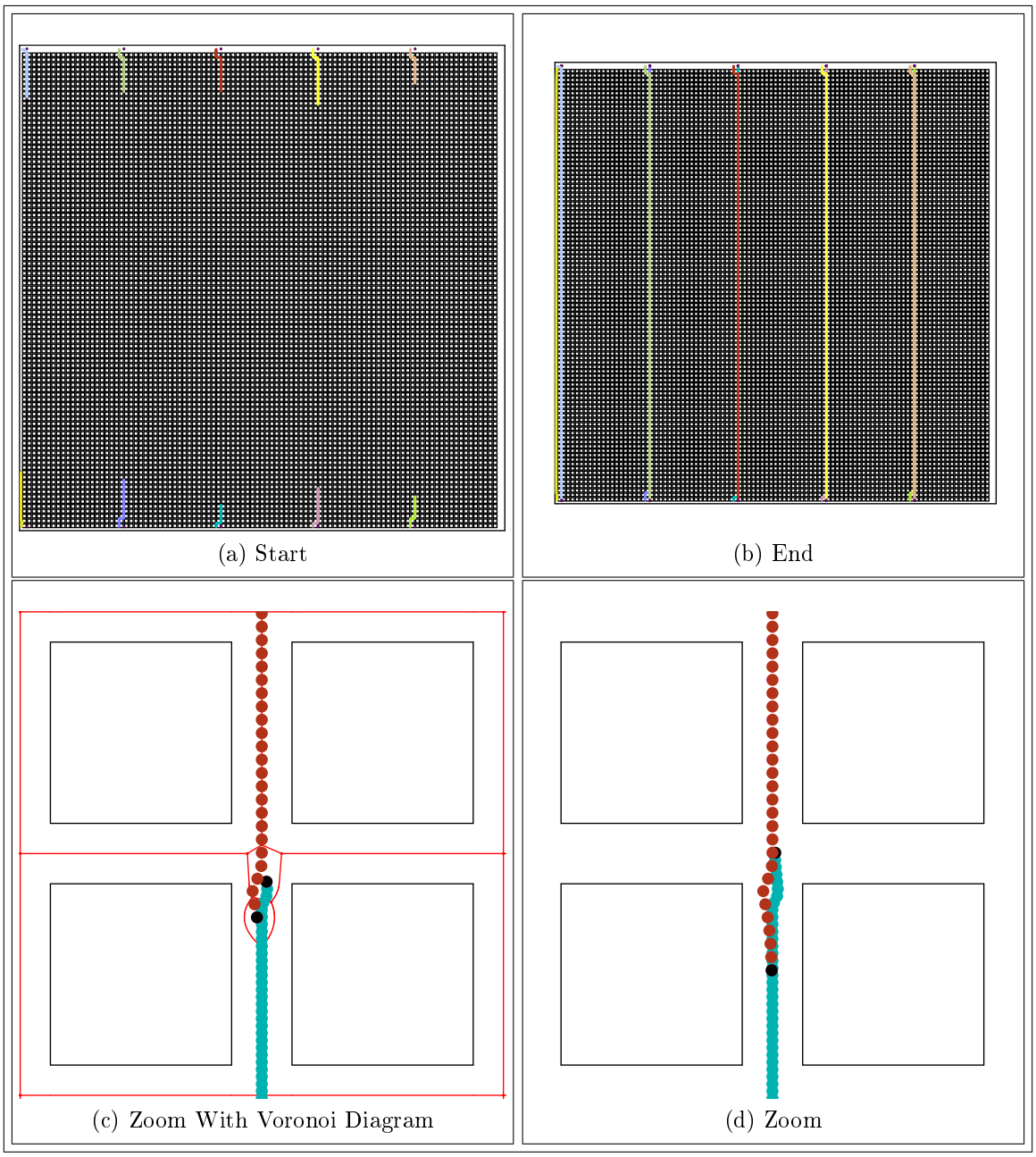
- **Environment:** 9,604 blocks of 15m x 15m separated by 5m
- **Agents:** 10, 40
- **Goals:** Arbitrarily chosen
- **StepSize:** Between 0.5m and 1.5m
- **Maximum Iterations:** 1,000

The simulation represented by table 5.4 is very similar to the ones from table 5.1. Except that here the environment is full of obstacles, represented by more than a thousand voronoi segments. This answers the higher amount of iterations needed to find a path. In this simulation the agents were relatively close to each other, as we can see by looking at the maximum path length. Therefore, even though the maximum number of iterations was kept on 1,000, they did not reach it.

The next simulation, represented in table 5.4, increases the distance between the agents. They now have to cross the entire city, as we can see by the maximum path length, almost 2km. We have seen before that without the limitation on the number of iterations the mean step time was more than 30ms with just one agent. But here, with the limitation of 1,000 iterations, we can see that it is possible to achieve a faster step time, even in the presence of more agents. As we can see in Figure 5.5b, the resulting paths are still the shortest paths. We can also see, at Figure 5.5c, that they smoothly avoid each other because their voronoi vertices guide them in a free direction. In the sequence, they return to their original path, Figure 5.5d.

Table 5.4: Simulation in a Large Environment: 40 Agents

Number of Agents	40	10
Max Number of Voronoi Segments	106,177	106,009
Mean Step Time (s)	0.0003	0.0005
Mean Iterations	314.49	334.62
Maximum Length (m)	392.92	1,988.33



(a) Start

(b) End

(c) Zoom With Voronoi Diagram

(d) Zoom

Figure 5.5: Simulation in a large environment

5.6 Comparisons

In this section we will compare our work to others.

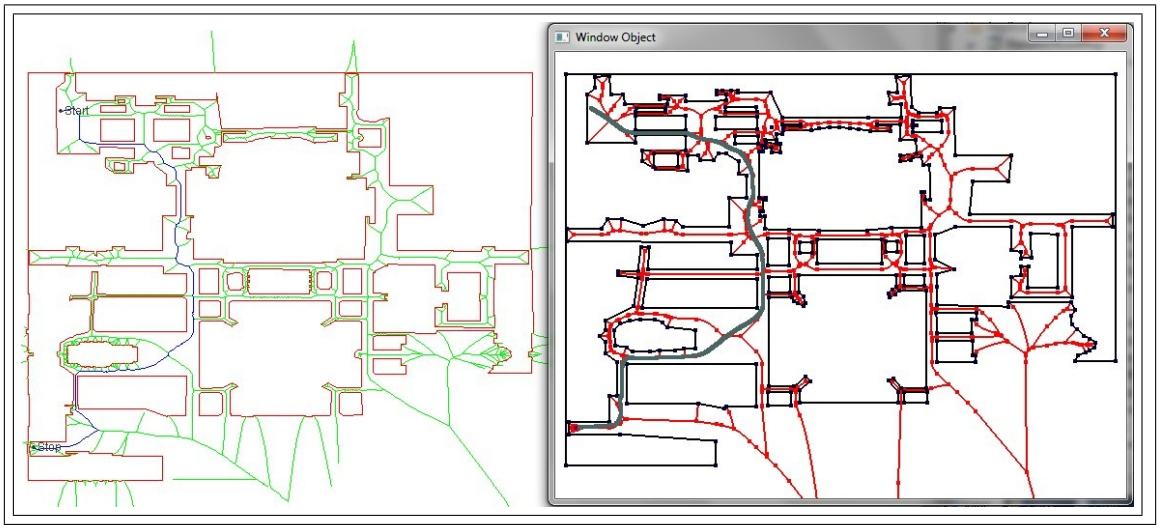
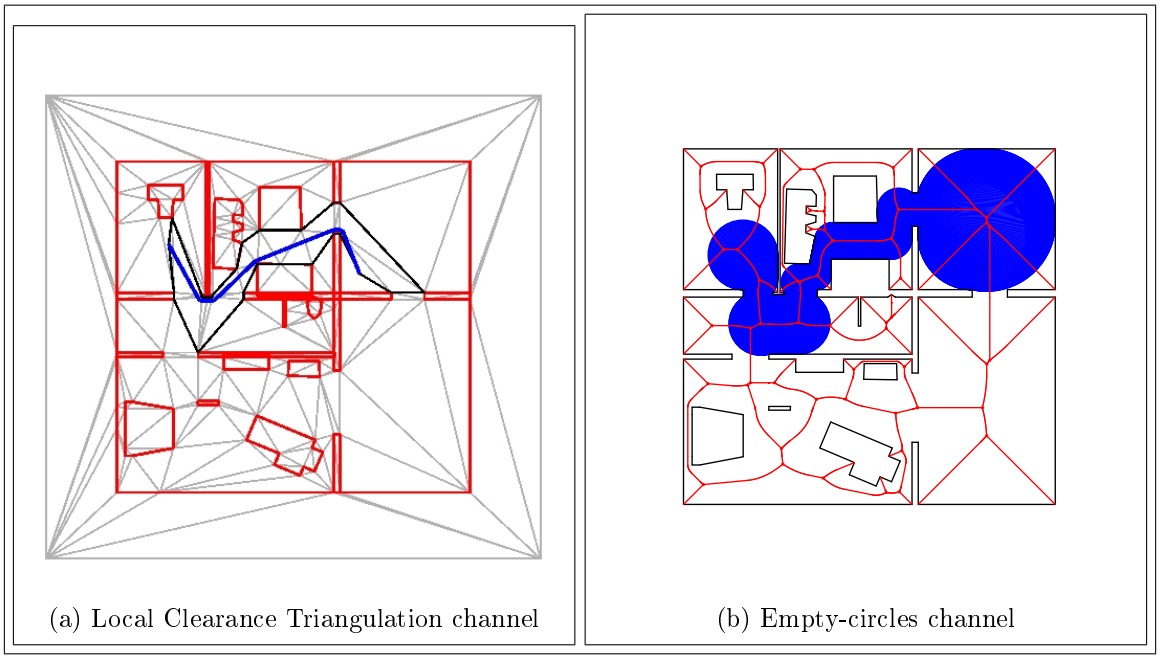


Figure 5.6: Comparation with discrete-points approximation of the voronoi diagram

Figure 5.6 shows a representation of the northern half of the Morningside Campus of the University of Columbia, New York. At the left, there is an approximation of the voronoi diagram using discrete points (BLAER, 2001). The campus map contains 4,407 points in their dicrete-point approximation. At the right, we can see the same campus map as represented by our application. The map was hand-made in less than 30 minutes with lines and polygons. It shows how easier it is to model the world in our approach. There is also a representation of a path calculated in each one of the applications. Their voronoi diagram calculation takes time and is not suitable for dynamic environments.



(a) Local Clearance Triangulation channel

(b) Empty-circles channel

Figure 5.7: Comparation between circular and triangular channels. (a) The black lines represent the channel. (b) The channel is represented by the circular blue area.

In this comparation we show that our approach is able to produce channels like the ones created with triangular meshes. The difference is that, in our case, the circles represent more of the free space than the triangles. We can see the result of both approaches in figure 5.7. The empty-circles are drawn picking point samples from the path that are very small apart. Joining them together we can build a channel which indicates the possible free space an agent has when following the calculated path.

6 CONCLUSIONS AND FUTURE WORK

In this work we presented an approach to perform path planning in large dynamic environments. The proposed algorithm makes use of a high performance underlying structure, the dynamic voronoi diagram. With its calculation at interactive rates, we were able to build an optimal maximal-clearance roadmap. We used then the properties of the voronoi diagram to calculate smooth paths with arbitrary clearance by calculating a path which takes into account the empty-circles. We showed that our algorithm is able to perform real-time smooth path planning in the face of large environments filled with static and dynamic obstacles for more than one agent.

There are some improvements that can be made in the algorithm. As for example the addition of a block streaming technique to contain the path search to relevant areas. Different heuristics to control the movement of the agent could also be added. For instance making the agent walk randomly towards a place inside the empty-circle on the direction of the path. Therefore creating unique movements while still keeping the obstacle avoidance. We are also planning to provide our source code on the internet, but first we should make it more robust to user input.

REFERENCES

BLAER, P. S. Robot Path Planning Using Generalized Voronoi Diagrams. Available in: <http://www.cs.columbia.edu/ pblaer/projects/path_planner/>. Acessed in: dec. 2010.

HAVOK. Havok AI Brief, June 2010. Available in: <http://www.havok.com/index.php?page=havok-ai>. Acessed in: nov. 2010.

HO, Y.-J.; LIU, J.-S. Collision-free curvature-bounded smooth path planning using composite Bezier curve based on Voronoi diagram. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTATIONAL INTELLIGENCE IN ROBOTICS AND AUTOMATION, 8., Piscataway, NJ, USA. **Proceedings...** IEEE Press, 2009. p.463–468. (CIRA'09).

HOFF III, K. E. et al. Fast computation of generalized Voronoi diagrams using graphics hardware. In: COMPUTATIONAL GEOMETRY, New York, NY, USA. **Proceedings...** ACM, 2000. p.375–376. (SCG '00).

HWANG, Y.; AHUJA, N. A Potential Field Approach to Path Planning. In: IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION. **Proceedings...** [S.l.: s.n.], 1992. v.8, p.23–32.

KALLMANN, M. Navigation Queries from Triangular Meshes. In: THIRD INTERNATIONAL CONFERENCE ON MOTION IN GAMES (MIG). **Proceedings...** [S.l.: s.n.], 2010.

LAVALLE, S. M. **Planning Algorithms**. [S.l.]: Cambridge University Press, 2006.

MABROUK, M. H.; MCINNES, C. R. Solving the potential field local minimum problem using internal agent states. **Robot. Auton. Syst.**, Amsterdam, The Netherlands, v.56, p.1050–1060, December 2008.

NIEUWENHUISEN, D.; KAMPHUIS, A.; OVERMARS, M. H. High quality navigation in computer games. **Sci. Comput. Program.**, Amsterdam, The Netherlands, The Netherlands, v.67, p.91–104, June 2007.

OKABE, A. et al. **Spatial Tessellations**: concepts and applications of voronoi diagrams. 2.ed. [S.l.]: John Wiley&Sons, 2000.

PATEL, A. Amit's A* Pages, October 2010. Available in: <http://theory.stanford.edu/ amitp/GameProgramming/>. Acessed in: dec. 2010.

PATHENGINE. PathEngine SDK Documentation, 2010. Available in: <http://www.pathengine.com/Contents/page.php>. Acessed in: dec. 2010.

PINTO, F. M.; FREITAS, C. M. D. S. Dynamic Voronoi Diagram of Complex Sites. In: SIBGRAPI - CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES, 2010, GRAMADO, RS, BRASIL. WORKSHOPS OF SIBGRAPI - SPECIAL SESSION ON WORKS IN PROGRESS, 23., Porto Alegre, RS. **Proceedings...** Sociedade Brasileira de Computação, 2010. p.1–8.

REYNOLDS, C. W. Steering Behaviors For Autonomous Characters. In: GAME DEVELOPERS CONFERENCE 1999 HELD IN SAN JOSE, CALIFORNIA, San Francisco, California. **Proceedings...** Miller Freeman Game Group, 1999. p.763–782.

SILVEIRA, R. et al. Path-Planning for RTS Games Based on Potential Fields. In: BOULIC, R.; CHRYSANTHOU, Y.; KOMURA, T. (Ed.). **Motion in Games**. [S.l.]: Springer Berlin / Heidelberg, 2010. p.410–421. (Lecture Notes in Computer Science, v.6459). 10.1007/978-3-642-16958-8_38.

SUD, A. et al. Real-Time Path Planning in Dynamic Virtual Environments Using Multiagent Navigation Graphs. **IEEE Transactions on Visualization and Computer Graphics**, Piscataway, NJ, USA, v.14, p.526–538, May 2008.

TREUILLE, A.; COOPER, S.; POPOVIĆ, Z. Continuum crowds. **ACM Trans. Graph.**, New York, NY, USA, v.25, p.1160–1168, July 2006.

YERSIN, B. et al. Crowd patches: populating large-scale virtual environments for real-time applications. In: INTERACTIVE 3D GRAPHICS AND GAMES, 2009., New York, NY, USA. **Proceedings...** ACM, 2009. p.207–214. (I3D '09).