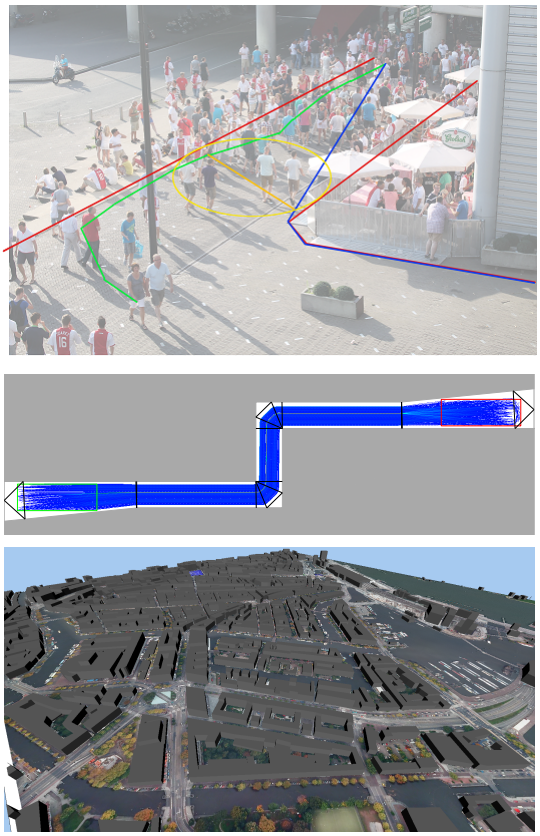
Creating Dynamic and Density Dependent Indicative Routes for Crowd Simulation

Martijn Bloemheuvel

September 1, 2014



Department of computer science
Utrecht University, The Netherlands
Supervisor: R. Geraerts

Universiteit Utrecht

Abstract

Crowd simulation has emerged as an important research field within computer science. Simulations for new commercial complexes and large events are increasingly becoming the standard. Additionally we are able to show a growing range of realistic specialized crowd behaviors. The latest generation of research has provided us with powerful approaches for exceedingly larger crowds.

Within the field of crowd simulation we see a strong distinction between global and local methods. Unfortunately, combining these micro and macro methods can be difficult and can lead to problems when the results of these methods conflict. In this thesis we will define an approach that attempts to unify two existing global and local approaches by functioning on a meso level between the macro and micro planning. Agents remain individual entities with personal goals and settings.

We will show that currently global and local methods can heavily counteract each other and lead to undesirable behavior. By storing global paths for individual agents that are dynamic, instead of static paths from one specific point location to the next, we achieve increased flexibility. We introduce the concept of paths that are planned from the edge of one cell to the edge of the next cell.

To protect the quality and benefits of the original methods, we will show that the resulting paths for individual agents not under the influence of other agents will remain the same. We will implement this aproach to obtain an implementation that unifies global and local pathing.

Experiments show that the paths reached with our approach mitigate undesirable and unrealistic results like large clusters within densely agent-filled regions making it a valuable contribution to current research. The concepts can be applied to a range of pathplanning solution that depend on a combination of separate global and local approaches.

This project has been made possible through a collaboration between Utrecht University and INCONTROL Simulation Software. The algorithms have been developed for Pedestrian Dynamics, which is a pedestrian flow simulator by INCONTROL.

Contents

1 Introduction

In Section 1.1 we discuss what crowd simulation embodies. Next, in Section 1.2 we touch on path planning and some of the factors that should be considered. Readers familiar with the concepts of crowd simulation and path planning can start from Section 1.3 where we briefly summarize the contributions of this thesis. Lastly, in Section 1.4 we explain the overall structure of this thesis.

1.1 Introduction to crowd simulation

1.1.1 Motivation for crowd simulation

Crowd simulation is a much discussed and continuously growing research field (Zhou et al., 2010). In addition, the rise of more powerful hardware allows for increasingly complex approaches to be implemented and utilized in practice. This is accompanied by an increasing awareness of the importance of crowd simulation in many scenarios.

Scenarios for crowd simulation range from commonplace commercial models of your local shopping-mall and optimization of your shopping experience, its application for entertainment purposes with the creation of large virtual crowds for movies, and for disaster avoidance like modeling crowd flows at large events to avoid disasters due to overcrowding.

Simulations of major events and within the planning of new construction are essential in this modern day and age. More than just seeing the characters move around within crowds, we can retrieve important information from these models such as the expected crowd density at critical locations, or how we expect crowd flows to change under the influence of different events.

(a) A crowd gathering outside an Amsterdam train station.

(b) A crowd outside the Amsterdam Arena during a sporting event.

Figure 1

1.1.2 Process of crowd simulation

The intent of crowd simulation is to simulate the movement of many *characters* within an environment. These autonomous characters or *agents* imitate various human behaviors that are based on various factors such as crowd densities or even the perceived personalities of the characters.

The issue of *navigation* is an important part of the field of crowd simulation. Navigation refers to planning smooth and realistic paths within the environment from a start to a goal while respecting obstacles, other characters and various

other conditions. In practice we see that the realistic behavior of an individual or the real time simulation efficiency of extremely large crowds are prioritized to different extents depending on the application. Where some methods aim to plan short simple paths for large amounts of characters, others consider a wide variety of conditions to achieve very natural-looking paths.

A crowd can be defined as *a large group of characters in the same physical environment* (Musse & Thalmann, 1997), or in the case of crowd simulation, a virtual crowd in a virtual environment. As such, we can define crowd simulation as stated by van Goethem (2012); *Crowd simulation entails realistically mimicking, in a virtual way, the behaviour of a crowd of entities in a diverse set of situations. The behaviour of the entities should be consistent with observed behaviour in real-life.*

1.2 Path-planning

The general path-planning problem is an integral part of crowd simulation. It relates very closely to navigation. Path-planning addresses the problem of finding a path from a character's position to its goal through an environment. There is a wide range of path-planning methods which support various different scenarios, complexities and efficiencies. We will give an overview of these methods in Sections 2.1 and 2.2.

The goal of a path-planning method is to find an optimal route; however, the optimal route is often situational. What is considered optimal varies according to the application, environment and desired behaviors. In robotics, for instance, the focus is on the shortest or most energy efficient collision-free paths for robots. However, when these paths are applied to characters, the result is often not a realistic path for a person. In crowds we vary our paths based on a wide range of different concepts. Hence, crowd simulation requires more of path-planning methods than just a path, since more criteria come into play.

As an example of how we vary our paths in crowds, it has been shown that we attempt to maintain personal space (Helbing & Molnar, 1995), and this tells us that the local crowd density is an important influence in the kind of path we take.

Another example is that we have a preference for the type of terrain we traverse. Pedestrians, for instance, prefer sidewalks over the middle of the road yet will still cross the road if need be. Jaklin et al. (2013) provides a method accounting for these kind of preferences.

One last example is the process known as *laning* that occurs in dense crowds. In very busy places, people tend to follow the person in front of them which we see simulated by the formulation of streams according to van Goethem (2012).

1.3 Research goals

In this thesis we seek to address undesirable complications in local behavior by utilizing dynamic route selection for Indicative Route Method (IRM) based path planning (Karamouzas et al., 2009) rather than strict routes which can become irrelevant due to local influences.

Indicative Route Method-based implementations offer a global route while a corridor surrounding the route constrains the character. Corridors were introduced in (Geraerts & Overmars, 2007a). Corridors are collision-free for the static obstacles. This follows from corridors being the union of the set of circles with their center points lying along the medial axis and their radius defined by the distance to the nearest obstacle points from this center point. As such, by never leaving the corridor it is ensured that the character does not collide with static obstacles.

By using the corridor to pick attraction points on the indicative route, the character is pulled forward. The Explicit Corridor Map (ECM) (Geraerts, 2010) offers a structure to quickly obtain these corridors.

Of course this still leaves dynamic obstacles and collision avoidance with other characters to be addressed. Current implementations of Explicit Corridor Map-based methods and Indicative Route Method-based approaches center on global pathing and rely on social forces to resolve many local problems.

For further details of ECM and IRM see Sections 3.2 and 3.3. For now, we will state that though path selection is beyond the scope of the ECM/IRM methods, generally it is done utilizing A* (see Section 2.1) on the ECM and then creating a triangulation-based shortest path within the corridor (Geraerts & Overmars, 2007a) (see Section 3.4). This works well for creating an indicative route that leads to smooth paths for individual characters and small groups of characters in general cases.

Unfortunately, relying fully on local methods to resolve local issues while still pulling a character in a given direction can produce issues of its own. The methods used to handle local conditions can at times heavily counteract the direction from the attraction point. This can lead to jittery paths, or forcing characters back towards an indicative route that might not be so relevant anymore, or lead to big clusters of characters that are all being pulled towards overlapping shortest paths.

We can elaborate on the problem of shortest paths creating clusters. It has been shown people favor energy-efficient paths (Guy et al., 2010). However, shortest paths often have a lot of overlap because in most corners the shortest path will lie on the inside of the corner. For one character this is not an issue and the path followed is realistic and natural. The problem becomes apparent when a large number of characters all follow their overlapping shortest paths, thus creating large clusters of characters shoving each other aside.

Using these shortest paths as indicative routes for IRM methods leads to attraction points on this shortest path. Characters are then dependent on the local methods to avoid collisions. Unfortunately, in a high-density situation, characters will continue to get attraction points that pull them together where social forces try to keep them apart.

This indicates the need to have attraction points on paths that do not lie on the shortest path but on paths that are to some extent influenced by local

conditions. As recomputing these paths constantly is expensive, dynamic paths would be desirable.

By making it possible to define paths more generally and giving consideration to the position of a character within a segment, we allow characters to change the path IRM draws upon for attraction points, based, for instance, on density information. This means we need a method that functions on a meso level of path planning which functions between the ECM (macro) and social-forces methods (micro).

Deciding to adapt a path based on local conditions is nothing new within the realm of path planning, many methods have tried to tackle this problem. Local methods even focus solely on local conditions as mentioned in Section 2.2.

In applying ECM and IRM methods, it is common practice to replan a global path based on local conditions; however, it is difficult to know before attempting to compute an alternative path whether viable alternative paths are available, and moreover computing alternative paths at real-time can be costly.

Additionally, global routes generally do not consider local positioning within a segment, only a retraction onto the underlying graph. For example with ECM and IRM methods, the medial axis is used as a graph and the closest points on the medial axis from the character and goal respectively, are connected. It can happen that a different path is shorter based purely on a character's position relative to the graph. If a character is on the left hand side of a corridor, odds are that passing an obstacle on the left is shorter even if the medial axis that passes the obstacle is slightly longer on the left side.

This thesis seeks to explore the possibility of planning global routes while still taking into account the position of the character within that segment by giving meaning to the position of a character within a segment.

We hope to provide paths that automatically adapt based on local displacement, due, for example, to having avoided other characters.

Lastly, we hope to provide paths that maintain, more strictly, the offset of a character from the medial axis at higher densities.

1.4 Document structure

The rest of this document is structured as follows.

Section 2 provides background information on related methods in crowd simulation. Readers with a background in path planning may consider skipping this section. Section 3 gives an overview of core crowd simulation concepts such as the medial axis, ECM, crowd density and IRM that this thesis relies on heavily. Readers familiar with ECM and IRM may consider skipping this section and referring to them when needed. Additionally, in Section 4 we will discuss some observations regarding the relevance of the shortest paths for pedestrians in crowds.

- In Section 2, we give an overview of related works. It offers an overview of crowd simulation methods starting with global methods and scaling down to micro methods, followed by a brief look at crowds.

- Section 3 highlights the research most relevant to our thesis with an overview of the ECM and IRM, and is particularly crucial to those not familiar with either method.

- Section 4 discusses the observations on crowd behavior that motivated this thesis.

Following, in Section 5, 6 and 7 we present a way to obtain dynamic paths and how to utilize them.

- Section 5 discusses various ways of defining alternative paths within the ECM for IRM and the supporting structures employed. It evaluates different alternatives for local position dependent pathing. To achieve this we will introduce a different way of using the information in the ECM.

- Section 6 offers two alternative approaches to create local position dependent paths. It shows how to use a subdivision as explained in Section 5 and obtain indicative routes for IRM. One alternative focuses on high quality curved paths, where the other alternative focuses on practical efficiency at a loss of quality.

- In Section 7 we investigate how best to set attraction points on the dynamic paths offered in Section 6. We present two different approaches, one with a heavy focus on efficiency and one with a focus on quality. For practical applications, an implementation that draws on both these approaches should be preferred.

In Sections 8, 9 and 10 we look at the results of our research.

- Section 8 we outline the experiments done. We take a look at some runtime measurements to generate cells in preprocessing for a large scenarios. Additionally we present heatmaps created with characters moving through a selection of scenarios at varying crowd densities to highlight the effects of our method.

- Section 9 presents the conclusions of our research and experiments. Here we will examine the contributions and benefits of our method while also noting the difficulties and hurdles.

- Section 10 examines what room there is for future works and their potential direction.

Lastly, in the addenda we look at high-level routing and how the approach in this thesis can be used to select additional high-level routing points which could improve global planning. Also, we propose a format for storing the contributions of Section 5.

2 Related Works

In this section, we give an overview of the research done on the topic of crowd simulation that served as the foundation for this thesis. We start by addressing global planning methods, before shifting our perspective to local methods. Finally we discuss crowds in relation to crowd simulation.

For those interested in a more extensive introduction to general path-planning, LaValle (2006) is recommended reading. Zhou et al. (2010) provides further background on crowd simulation.

2.1 Global planning

Global path planning targets the problem of constructing a global route from a start to a goal within an environment. Global methods function on a grid, graph or other abstract representation of the environment, often creating low resolution paths for the sake of efficiency.

2.1.1 Representations of the environment

A *grid* (as shown in Figure 2a) is one of the simplest forms of a spatial subdivision. It is used in many games and applications and has been studied and evaluated thoroughly (Sturtevant, 2012).

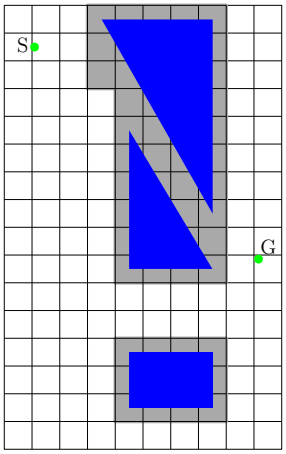
A grid can be described as a large number of cells which are each connected to their neighbor. Grid cells that cover obstacles are made inaccessible. The graph that follows allows us to quickly plan paths through the free space.

An obvious drawback to this approach is that it is unclear how to handle partially covered cells as shown in Figure 2b. On the one hand, excluding these partially occupied cells from the graph can mean not finding many paths that are actually valid paths. On the other hand, including them can lead to finding paths that cross obstacles and as such are not valid.

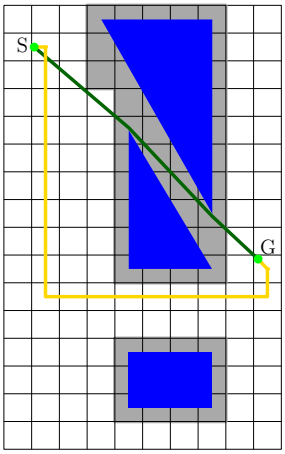
The quality of a grid generally improves with a higher resolution, although this unfortunately increases the storage requirements. More importantly still, a bigger graph drastically increases the time required to find a path. This is why some applications use multiple layers of path-planning on different resolution graphs. First, a general path is found. Following, this path is refined on higher resolution graphs.

Another drawback to this approach is that the resulting paths make strong 90 degree turns. This is because the grid only allows characters to move up, down, right or left. As a consequence this means that a path where we travel fully horizontally and then vertically is equally long as a path where we alternate horizontal and vertical steps. More advanced solutions exist that allow diagonal movement from cell to cell as well, but to some extent the problem remains and resulting paths are not realistic for human motion.

Many methods rely on smoothing to achieve more realistic paths, such as Botea et al. (2004). As smoothing operations are computationally expensive however, this can be a tough sell in practice.



(a) Basic grid with obstacles in blue, disallowed cells colored. Start and Goal marked by S and G.



(b) Yellow: A possible "shortest" path planned on the grid, Green: actual shortest path.

Figure 2: Grid based path planning.

Roadmaps are more advanced graphs that capture the connectivity of the free space. A roadmap can be described as a union of curves that can connect any start point in the free space to any goal point in the free space. This requires that the roadmap can be reached from the start point (accessibility), and that the goal point can be reached from some point on the roadmap (departability). And lastly it requires that for each point on the roadmap, there exists a path that is part of the roadmap to every other point in the roadmap (connectivity).

Good implementations of roadmaps are much sparser than graphs resulting from grids. Where a grid does not consider the environment in which it sets its vertices, roadmaps vary based on the environment, often requiring far fewer vertices to achieve full connectivity throughout the free space. For more information on roadmaps, the book Choset et al. (2005) is recommended.

A popular implementation of roadmaps is Probabilistic Roadmaps. Probabilistic Roadmaps are based on sampling the free space and using a local planner to connect a sample to other nearby samples. The goal and start points are added in, and, assuming the connectivity constraint holds for the resulting graph, a path is found. The connectivity constraint only holds if the graph is dense enough, meaning enough samples took place. It has been shown that Probabilistic Roadmaps are probabilistically complete (Kavraki et al., 1996). As long as the method runs long enough, eventually completeness will be achieved. This results from the probability that a path will not be found if one exists approaching zero as the number of sample points increases. An analysis of Probabilistic Roadmaps planners can be found in (Geraerts & Overmars, 2007b).

2.1.2 Finding paths on spatial representations

Once a graph is obtained, there remains the question of how to find the path in the graph. The most popular methods for finding such paths are Dijkstra's algorithm (Dijkstra, 1959) and the A* search algorithm (Hart et al., 1968). The

A* search algorithm is an extension of Dijkstra's algorithm that uses heuristics to achieve better runtime performance. There exist an immense number of variations of the A* search algorithm and a considerable number of different heuristics.

Dijkstra's algorithm takes the closest unvisited vertex in the graph. It then calculates the distance from this vertex to all neighboring vertices not yet visited and updates the distance at a neighboring vertex if it is shorter than any distances previously recorded at this neighboring vertex. Once the goal is found the algorithm can terminate with the shortest distance path having been found.

The A* search algorithm considers the "best" unvisited node, rather than just the closest unvisited node. Which node is considered the best depends on the heuristic because the A* search algorithm takes the cost of reaching the node and the estimated cost resulting from the heuristic. The requirement for a heuristic to be admissible is that it must not overestimate the distance to the goal.

Whereas Dijkstra's algorithm expands broadly, in the case of the A* search algorithm good heuristic functions lead to vertices that are more likely to be part of the path being explored first. This, in turn, often leads to paths to a specific goal being found faster. Submitting a heuristic function that always returns 0 leads the A* search algorithm to the same radial exploration pattern as Dijkstra's algorithm.

2.2 Local pathing

Finding global paths, as discussed in the previous section is one part of the path planning problem. However, these global paths do not consider dynamic obstacles, or collisions with other characters. This means that in crowd simulation with large numbers of characters, just finding global paths is obviously not enough. To avoid dynamic obstacles and other characters various local pathing methods exist which we will discuss here.

2.2.1 Force-based models

Perhaps the most widely known local path planning method was first described in Reynolds (1987). The boids model outlines a method in which many entities maintain a flock while avoiding collisions within the flock. Much like a flock of birds, which is the example used in the paper. The boid flock is a generalization of particle systems which *"are used to represent dynamic 'fuzzy objects' having irregular and complex shapes"* as stated by (Reynolds, 1987). It is based on three rules, listed in decreasing order of importance;

- Collision avoidance.
- Velocity matching.
- Flock centering.

These rules are very intuitive, collision avoidance speaks for itself, namely avoiding collisions with other flockmates. Velocity matching refers to attempting to match the heading and speed of nearby flockmates. Lastly, flock centering refers to staying close to nearby flockmates.

In another paper, the same author brought us (Reynolds, 1999) which presents a method for autonomous characters to navigate around an environment in what the paper describes as a *"life-like and improvisational manner"*. This method is a good example of how a multilevel hierarchy can be used to address path planning. It discusses a three-level behavioral hierarchy.

- Action selection. This refers to constructing a strategy reliant on goals and planning.
- Steering. This dissects the strategy into simple instructions.
- Locomotion. This converts the instructions into motion.

Steering behaviors are extensively covered in the paper. To give a good idea of what steering entails, we note that several steering behaviors are described in the paper, including seeking, fleeing, pursuit, obstacle avoidance and path following.

Another model for handling the local pathing problem is the *social forces model* as introduced by Helbing and Molnar (1995). Rather than being exerted on the agent by the environment, these forces are an expression of individual agents' motivations to perform certain actions, hence social forces. Social forces, referred to therein as *social fields*, were first introduced by Lewin (1951) in reference to behavioral changes. The model offered by Helbing and Molnar (1995) presents 3 terms to describe pedestrian behavior subject to social forces.

- Acceleration towards the desired velocity of motion.
- Keeping distance from other pedestrians and borders to avoid collisions.
- Attractive effects (friends, window displays, etc).

A major downside to this method is that agents behave reactively. This is addressed in Karamouzas et al. (2009) where the idea of social forces is extended with a predictive collision-avoidance force. This leads to characters adapting their course much sooner, thus ensuring smoother collision-avoidance.

2.2.2 Flow

A very different way to handle the pathing problem is through modeling flows, an approach that has emerged from the fluid dynamics community. Chenney (2004) serves as a good introduction to this kind of path planning. It introduces *flow tiles* (shown in Figure 3 where each tile represent a velocity field. This method can be used to model crowds but is generic enough to also be used to model rivers or fog. The big downside to this approach is that the speed and direction of the agents is completely controlled by the tiles. Accordingly, it is not possible to consider aspects like goals or other attributes that give characters individuality. Additionally, because this method is widely applicable and not just targeted at crowds, it does not address many aspects of crowd behavior.

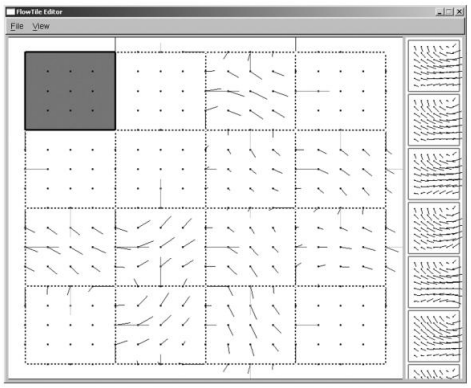


Figure 3: Flow tiles shown in an editor. Chenney (2004)

Treuille et al. (2006) takes a flow-based approach as well. Their method targets large groups with common goals, and is targeted at crowds specifically. They calculate a potential field, which can be used for all members of a group sharing the same speed field, discomfort field and goal rather than having to calculate it for every individual character. The potential field indicates the direction for all of these agents at each point of the environment. This also highlights the downsides of this method. If you have individual characteristics and goals for each character, you have to calculate a potential field for each character which makes the method rather inefficient. Additionally, as there is no coordination between groups of characters with different potential fields the results are suboptimal.

2.2.3 Velocity Obstacles

Another way of handling local path planning has its origin in robotics. The velocity obstacle (Fiorini & Shillert, 1998) of a virtual agent is the set of all velocities for the virtual agent that will result in a collision between the agent and a specific moving obstacle in the near future. Velocity obstacles can be used to efficiently move a character through an environment with moving obstacles. Unfortunately it is less well suited to handle environments with many characters because not only will characters change their velocity to avoid other characters that are often also changing their velocity, but also each character will take full responsibility for avoiding collisions. This means that when a future collision seems imminent, one character will move out of the way fully while the other character makes no effort.

Reciprocal Velocity Obstacles (RVO), (van den Berg, Lin, & Manocha, 2008) and (van den Berg, Patil, et al., 2008), attempt to address this problem by sharing the responsibility of avoiding collisions with other characters instead of both characters taking full responsibility. This means, if both characters pick a velocity that is not part of the RVO, we generally see collision-free paths. When characters pick a velocity very far from their last velocity, however, it can happen that both characters dodge to the same side, thus producing undesirable behaviors.

2.3 Crowds

There has been a lot of research on the subject of crowds. Specifically crowd behavior in extreme scenarios is a popular subject because of its uses in modeling disaster and evacuation scenarios as described by Helbing et al. (2000), Helbing et al. (2007) and Moussaïd et al. (2011)). Research into this subject is vitally important as it can be used to predict crowd disasters and help authorities manage escape routes in panic situations to reduce casualties. This has prompted several evaluations of crowd disasters (Curtis et al. (2011), Still (2000)).

Far more common, of course, are crowds in which densities do not reach these extreme conditions. Busy shopping streets, central transportation hubs like train stations and sporting events are just a few commonplace examples of large crowds with high densities that more often than not remain mostly incident free. Modeling these scenarios is very important as well. As soon as the first designs are modeled, it becomes important to study how many people a location can support.

Utrecht Central station supported an average of 163877 passengers a day in 2011 (Verkeer en Vervoer Utrecht, 2012) and this number rapidly increases every year (Prorail, 2009). The station has been undergoing near constant renovations for years to support increasingly large crowds. It is essential, therefore, to know how many visitors the station can support now and in the future to meet this increasing demand.

How pedestrians find their way has been studied by Kneidl and Borrmann (2011) among others. To provide realistic simulations for crowds, we need to consider behaviors specific to crowds. An example of this is crowd turbulence, (Yu & Johansson, 2007), which extends the repulsive force term of the social force model (Helbing & Molnar, 1995) to reproduce crowd turbulence. Helbing and Molnar (1995) itself already facilitates another such crowd simulation in the form of lanes. Evaluating these simulations can be done by comparing them to data from real crowds (Lerner et al., 2009).

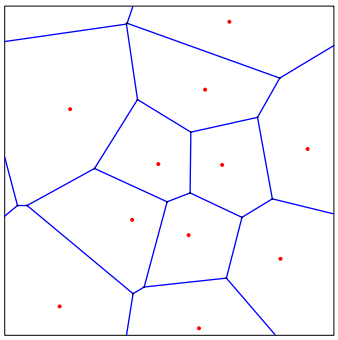


Figure 4: A set of sites in red and the resulting Voronoi diagram in blue. Generated using Explicit Corridor Map Generator v5.2 ©. R. Geraerts & W. van Toll. Utrecht University, Department of Information and Computing Sciences.

3 Fundamental prior research

This section presents an overview of the methods specifically extended and utilized heavily in this thesis. Additionally, it introduces terminology from these works used repeatedly in this thesis.

Section 3.1 discusses basic geometrical concepts such as the medial axis, generalized Voronoi diagram and Delaunay triangulations. de Berg et al. (2008) is an excellent reference for these geometrical concepts and especially the algorithms that use them. Section 3.2 presents an overview of the Explicit Corridor Map (ECM) structure and Section 3.3 an overview of the Indicative Route Method. Lastly we discuss the use of funnels to quickly compute shortest paths within slabs, or in our case, corridors.

3.1 Geometrical concepts

3.1.1 Voronoi Diagram

The Voronoi diagram (shown in Figure 4) is a very versatile geometric structure. It provides a subdivision of the total area under consideration. For a site p that is part of a set of sites P, a Voronoi cell associated with p consists of all points in the space closest to site p.

This means that the edge bordering two Voronoi cells lies exactly between the two closest sites of P, this is called a Voronoi edge. At points where three or more Voronoi edges intersect, three or more sites of P are equidistant from and closest to this point. These intersection points are called Voronoi vertices.

Voronoi diagrams commonly support different distance metrics, such as Manhattan distance, but the present thesis focuses only on Voronoi diagrams based on Euclidean distance.

A popular way of computing the Voronoi diagram for a set of points is to use Fortune's algorithm (Fortune, 1986). Fortune's algorithm is a plane sweep algorithm that considers new site events and disappearing arc events. It traces over the space using the sweep line while maintaining a beach line. Every site above the sweep line defines a parabola, and the beach line is formed by the lowest point of the union of all these parabolas. We can conclude then that a new arc can only appear on the beach line if the sweep line passes a new site;

this is a new site event. Meanwhile an arc disappearing as part of the beach line leads to a disappearing arc event. For a background on general plane sweep algorithms and a more detailed explanation of Fortune's algorithm, see de Berg et al. (2008).

There are many variations of Voronoi diagrams. They exist for higher dimensional spaces, higher orders and different input types. In the field of path planning, we are specifically interested in the use of Voronoi diagram to describe environments with polygonal obstacles. Voronoi diagrams for more complex inputs are commonly called Generalized Voronoi Diagrams (GVD). We can use a GVD as a roadmap (Section 2.1.1) as shown by Geraerts and Overmars (2008). Also, the paper Geraerts and Overmars (2008) points out that roadmaps based on Voronoi edges and vertices have the added benefit of maximizing the clearance. Because the Voronoi edges and vertices are equidistant from the nearest obstacle points, moving them always brings us closer to one of the obstacles which means less clearance.

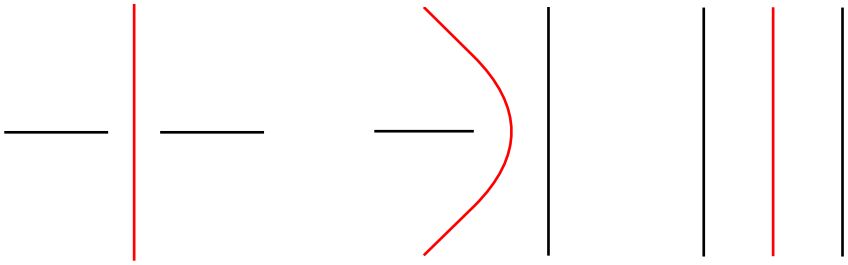
Supporting concave polygons requires splitting them up into convex polygons. This can produce the undesirable side-effect of Voronoi edges ending on convex corners of the original polygon if that is where the polygon happens to have been split. Once we have convex polygons we can treat them as chains of line segments. Of course, we need to ignore any parts of the Voronoi diagram that lie within any obstacles since such segments do not contribute to our roadmap as they are not even in the free space. If we proceed to remove the undesirable edges from splitting the polygons into convex polygons, we end up with a structure known as the medial axis (Section 3.1.3).

It should be noted that the term Generalized Voronoi Diagrams has been used for several different approaches. The present thesis concerns itself only with Voronoi edges and vertices that lie on the medial axis. Specifically, we are after the constrained Voronoi diagram (Aurenhammer & Klein, 1996), which respects the constraints enforced by the edges of the obstacles.

Using the GVD to get a Voronoi diagram suited for line segments leaves us with different types of segments that form the Voronoi edges. A segment of a GVD Voronoi edge is either a line segment or a parabola. A segment of the Voronoi edge is defined by the closest obstacle points for that segment. Such points can be of the following types (each displayed in Figure 5):

- The closest points can be on two line segment end-points forming a line segment. (Figure 5a)
- The closest points can be on two line segment forming a parabola and a line segment. (Figure 5b)
- The closest points can be traveling along two line segments forming a line segment. (Figure 5c)

A Voronoi edge can consist of several different segment types. At the point along the Voronoi edge where a segment switches from one type to another, we can place a node to signify this change. This means we can represent a Voronoi edge as a list of nodes with types.



(a) Voronoi segment defined by two end points of line segments.

(b) Voronoi segment defined by two line segments.

(c) Voronoi segment defined by two parallel line segments.

Figure 5: Voronoi edges in red with the obstacle line segments that define them in black for each type of Voronoi edge.

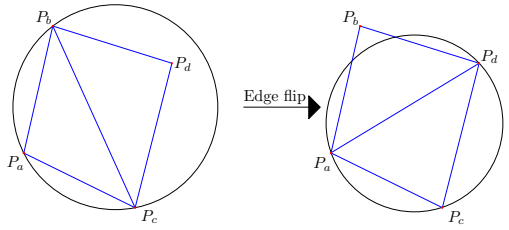


Figure 6: Flipping edges to reestablish the empty circle property.

3.1.2 Delaunay graph

The Delaunay graph is a dual graph of the Voronoi diagram. Each Voronoi vertex leads to a face in the Delaunay graph, and for every 2 adjacent Voronoi cells there is an edge in the Delaunay graph. The vertices of the Delaunay graph are the original sites used to create the Voronoi diagram. A simplified explanation of how a Delaunay triangulation can be generated is as follows; explain how to compute the Voronoi graph, then use duality to get the Delaunay graph, and then finally triangulate the faces that are not triangles.

Faces in the Delaunay graph end up with more than three vertices when a Voronoi vertex has more than 3 edges. This means that four sites must lie on a circle centered at the Voronoi vertex with no other sites being inside the circle. This also leads us to the empty circle property. The empty circle property tells us that *"Any circle through three points forming a Delaunay triangle does not contain any other points"* (Chena et al., 2010) and *"two points p_i, $p_j \in P$ form an edge of the Delaunay graph of P if and only if there is a closed circle disc C that contains p_i and p_j on its boundary and does not contain any other point of P"* (de Berg et al., 2008).

Another way of computing the Delaunay triangulation is through randomized incremental construction. By inserting the points one by one and fixing the triangulation for each step, we end up with the Delaunay triangulation for the full set of sites. To fix the triangulation after inserting a new site we need to reestablish the empty circle property. We do this by checking the new edges and flipping the edge as shown in Figure 6. For pseudo code of a randomized incremental construction algorithm that computes the Delaunay triangulation

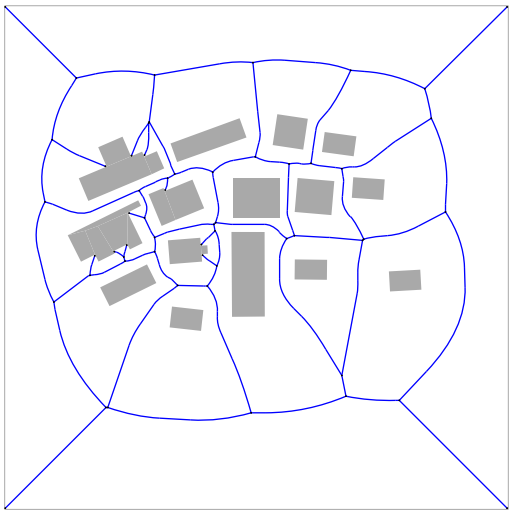


Figure 7: Medial axis generated using Explicit Corridor Map Generator v5.2, ©R. Geraerts & W. van Toll, Utrecht University, Department of Information and Computing Sciences.

see (de Berg et al., 2008). For this thesis the empty circle property specifically will be important.

Since we are utilizing constrained Voronoi diagram, we also need a constrained Delaunay graph (Aurenhammer & Klein, 1996). The constrained Delaunay graph includes the edges from the obstacles in the environment. These edges cannot be flipped and, as such, the constrained Delaunay graph does not always satisfy the empty circle property although it is as close as we can get under the constraint of the obstacle edges. Aurenhammer and Klein (1996) refers to the triangulation from this constrained Delaunay graph as *as much 'Delaunay' as possible.*

3.1.3 Medial Axis

In general terms, the medial axis of an environment is the set of points that has more than one closest point on the obstacle boundaries. Introduced in literature in 1967, Blum (1967) used the medial axis in combination with a time based medial axis function (and space velocity) to define shapes. For our purposes, the definition by Preparata (1977) is perhaps more convenient; *"The medial axis M(G) of an arbitrary simple polygon G is defined as the set of points of the plane internal to G which have more than one closest point on the boundary of G."* We can apply this to bounded environments by taking the boundaries as polygon G and considering the obstacles as holes in the polygon. Of course the definition is for a simple polygon, we have to exclude all the parts of the medial axis that fall within the obstacles to make this work.

Figure 7 shows a medial axis generated on a bounded environment with simple obstacles.

Exact methods for computing the medial axis can be challenging to implement robustly. Fortunately for the purposes of this thesis an approximate method will

suffice as long as we select a high enough level of precision. Though we have to be careful not to encounter degeneracies, an approximation will also provide us with higher efficiency.

3.2 Explicit Corridor Maps

The Corridor Map Method (CMM) was introduced in Geraerts and Overmars (2007a) as a technique to quickly generate corridors for path planning purposes. The method defines a corridor $C = (B[_t], R[_t])$ as the union of the set of balls with radii $R[_t]$ whose center points lie along its backbone path $B[_t]$. The backbone path here refers to the medial axis. The radii are defined by the clearance, the distance from the closest obstacle point to the point of the backbone path. This leaves us with a subdivision that defines the relevant free space along a path that can be planned on the GVD. As long as a character stays within the corridor, we know that it will not collide with any static obstacles which makes it a useful global path planning tool.

Explicit Corridor Maps (ECM) are an extension of the structure used in the CMM. The Implicit Corridor Map used in the CMM was found to be an inefficient way of storing the information of the free space. It was realized that as polygonal obstacles were being used, only the nodes that define the type of the Voronoi edge and the closest obstacle points at that node were of concern. This means that we could store the same information much more efficiently.

- The Voronoi vertices forming the roadmap that store a list of Voronoi edges originating from the vertex.
- The Voronoi edges storing a list of nodes that define the edge.
- The nodes that store the closest obstacle points and the Voronoi edge segment type until the next node. (Section 3.1.1)

This effectively defines the ECM and from it we can obtain all the information we need to generate corridors for paths planned on the graph swiftly.

Recently, research has been done into combining Explicit Corridor Maps (see Section 3.2 with exact methods for computing the GVD (Cibotaru, 2013). In the present thesis, however, we use the same approximation algorithm that most other Explicit Corridor Map-based research utilizes (to name just a few (Geraerts & Overmars, 2008), (Geraerts, 2010), (van Toll, 2011) and (van Goethem, 2012)).

3.3 Indicative Route Method

In Section 2.1, we discuss global path planning and the acquiring of global paths. Once a global path is found, we still need a method to utilize the global path. The Indicative Route Method (IRM) is such a method. IRM (Karamouzas et al., 2009) uses a so-called *indicative route* as a global path and a surrounding corridor to handle local problems. It has the benefit of leading to smooth paths because of how *attraction points* are selected. At each step a character selects an attraction point on the indicative route towards which an steering force is placed on the character. Additionally a repulsive force is introduced to keep characters in the corridor. Social forces (Helbing & Molnar, 1995) can be combined with these two forces to produce a complete path planning solution.

Attraction points are selected by drawing a circle along the backbone path (medial axis) and picking the point furthest along the path. The center of the circle is based on the retraction point on the medial axis and the radius is dependent on the local clearance. Retraction-based (ÓDúnlaing et al., 1983) planning centers on retracting a configuration to a more desirable region of the free space. In terms of the IRM, this is retracting the position of the character to the nearest point on the backbone path.

The selection of an indicative route is, strictly speaking, beyond the scope of the IRM. It supports any path for which we can have a valid corridor to match. Indicative route selection is of course crucial to the results of the IRM. The shortest paths are commonly calculated within the corridor, but in this thesis we show that this is not always a good choice and to provide an alternative way of getting indicative routes better suited for crowds.

3.4 Funnel-based shortest paths

Historically, finding a path within a simple polygon has been studied extensively. Guibas et al. (1987) gives us several algorithms related to this, including an algorithm to calculate the shortest path to all vertices of the simple polygon. What we are interested in is a specific algorithm that can be utilized to plan shortest paths within corridors specifically from our begin and end points. Guibas et al. (1987) shows it is possible to compute the shortest path within a simple polygon in linear time. What we use specifically is an extension of this algorithm that allows us to support circular arcs as required by the corridors (Demyen & Buro, 2006).

Applying the algorithm on a corridor is quite straight forward. Starting at the beginning of the corridor, we walk along its left and right borders until we reach the goal. We maintain a funnel as we progress along the borders. Initially, this funnel originates at the start point and runs to the left and right border points. At every step, we check the next left and right points along the border of the corridor.

- If a new point falls inside the current funnel, we narrow the funnel using the new point. That is to say, if the new point is a left border point that is to the right of the left point of the funnel and still within the funnel, the new point is the new left point of the funnel.
- If the new point lies outside the funnel on the same side however, we do not update the funnel with the new point. That is to say, if the new point is a left border point that lies to the left of the left point of the funnel, it is disregarded.
- If the new point is across the far side of the funnel, we store the point of the funnel that was crossed as part of the path and start the funnel again from that point. This means that if the new point is a left border point that lies to the right of the right point of the funnel we store the right point of the funnel as part of the path.

This is enough to find us the furthest visible point. Whenever we find a new point along the path, we will restart the funnel from that point. Using a queue, we can find the appropriate points without added expense to the runtime cost.

The key element here is that we find the furthest visible point of the path until the goal is visible.

Note that this is only a simplification of the algorithm presented in Demyen and Buro (2006), which is considerably more complex. However to read this thesis, understanding this simplified explanation is sufficient. We do recommend reading Demyen and Buro (2006) should you want a more in depth understanding of Funnel-based shortest paths.

4 Observations regarding using the shortest path in crowds

In the realm of path planning there is a wide range of methods to determine different paths. Shortest paths are popular and many methods are based on shortest paths, because people tend to minimize spent energy. However that does not always make them the right choice for our focus. What we show in this section is that in certain crowds, the shortest path actually makes a poor choice for an indicative route for the IRM (see Section 3.3 for more information on the IRM and its workings).

4.1 Observation

The Indicative Route Method (IRM) supports any path for which a matching corridor can be found. Most commonly we see the IRM being applied to shortest paths. This makes sense because it has been shown that people tend to follow shortest paths. There is a branch of path planning devoted to optimization approaches based on Integer Linear Programming, van den Akker et al. (2010) being one of many examples.

At the same time, however, many papers and methods focus on the replanning of paths based on density, not just because such paths are more efficient but primarily because it is natural for people to prefer less crowded areas (van Toll, 2011). This indicates that shortest paths are not necessarily the best focus where crowds are involved at a global level.

Additionally, in crowds we see the occurence of behaviors such as *laning* (see Section 2.3), which have nothing to do with shortest paths. This implies that, at a local level, shortest paths are not always the answer either.

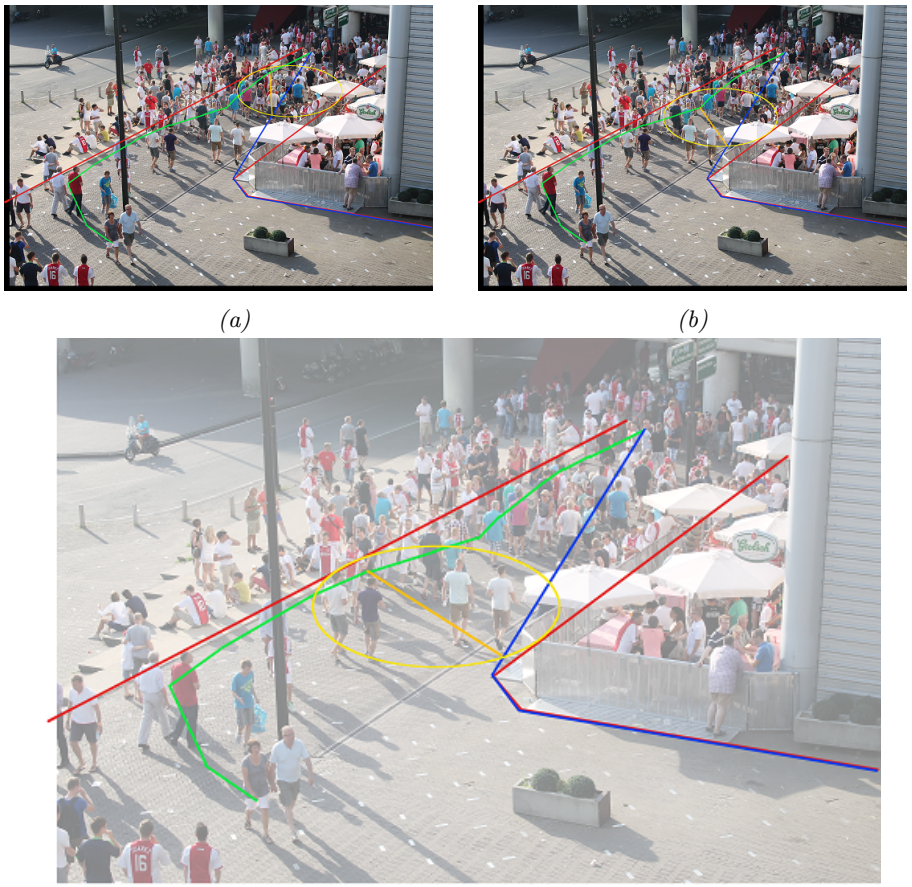
Although it is very tempting to argue that local and global path planning should be distinctly separated, when we have methods like IRM providing attraction forces it is intuitive to reason that IRM should function on locally relevant paths.

To show that in crowded regions people do not rely much on shortest paths, we took a series of pictures outside the Amsterdam Arena as people gathered there for a soccer match. In Figure 8 we highlight the path of one individual as she progresses through a crowded region. It is important to note here that the individual has several opportunities to change her position relevant to the borders of the region.

The relevant area is bounded by stairs on one side and a café on the other. Although it is possible for pedestrians to walk up and down the stairs, so many people are seated on them that few people would consider it a viable option. This leaves a passage way with a constant stream of people flowing through it.

Figure 9 shows the most probable shortest path. Starting with Figure 8a we see what happens when we try to place attraction points at several points along the path. Figure 9c shows an enlargement.

Figure 8: Pictures taken at the Amsterdam Arena. In every frame a single individual is selected and that individuals path is traced. Red marks the relevant boundaries, green marks the path step by step.



(a) *(b)*

(c)

Figure 9: In these pictures taken at the Amsterdam Arena, the shortest path is displayed in blue. We show the attraction point as it would have been selected had the individual been a character in a virtual simulation. The yellow circle is based on the retraction and clearance. The orange line shows how the attraction point would be calculated from the current position to the intersection between the retraction circle and the shortest path calculated at the start.

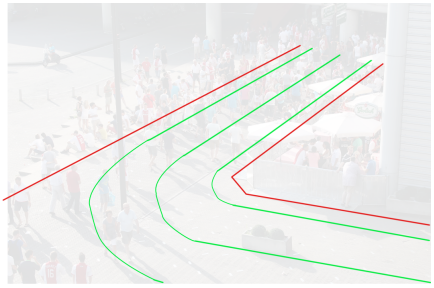


Figure 10: The same picture taken at the Amsterdam Arena. This time highlighting expected paths according to our base principles shown in green.

4.2 Conclusions based on observation

Figure 9c clearly shows just how far the attraction point lies from the actual path this individual takes.

This observation is consistent for the majority of individuals observed. Very few people forcefully stick to a shortest path. In a crowd only those with extreme urgency show the behavior of attempting to follow shortest paths.

Utilizing shortest paths for IRM in these crowded regions would mean that characters in many situations constantly have a force pulling them away significantly from where they would move in the real world. Accordingly, it can be argued that we should not simply use shortest paths for IRM within crowds. A local method based on viewfields that obtains laning behavior by shifting the attractive force resulting from IRM is provided by van Goethem (2012), but will still crowd the insides of corners more than desired. In this thesis, we propose finding more relevant indicative routes and relying more on IRM rather than trying to adjust the resulting forces as a solution for maneuvering characters through crowds.

Instead we see that the real world path stays relatively consistent with the borders of the region. The path does not run parallel with the border, however, since the direction must sometimes be changed slightly in order to avoid collisions. By and large, we see that even if there is room to cut corners, people only do so to a very limited extent.

We therefore base our thesis on two main principles to achieve realistic paths in crowds.

- IRM should generate attraction points from locally relevant paths.
- Characters within densely crowded regions should attempt to maintain their offset to the region borders as they progress through the region.

In Figure 10 we highlight how we expect the paths to progress without collisions according to our principles. The middle path is very much like the medial axis (see Section 3.1.3), with the other two paths maintaining their relative position between the medial axis and the border.

Based on our principles and the expected paths delineated in Figure 10, we can model simple regions, as shown in Figure 11. Achieving routes like these is the objective of this thesis.

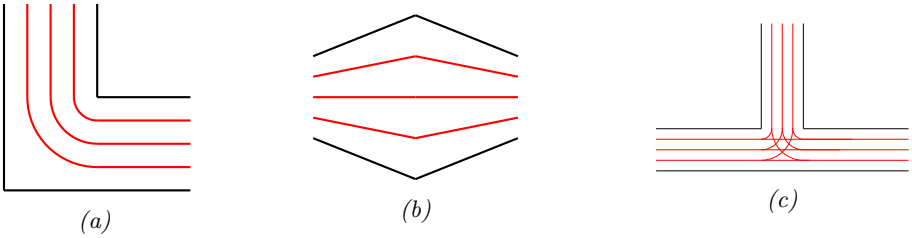


Figure 11: Modeled paths based on our principles in simple regions. Black marks the borders, the paths are shown in red.

5 Dynamic IRM paths

In this section we evaluate different approaches to obtain sets of alternative paths that match our observations from Section 4 regarding pathing in crowded regions. We show these sets of paths can be created using a support structure based on the Explicit Corridor Map (ECM). We call these alternative indicative routes *lanes*. The purpose of these lanes is to represent paths along edges of the ECM and to utilize them to achieve more realistic pathing. In Section 6 we show two alternatives for actually creating the lanes and in Section 7 we show two methods to obtain attraction points on these lanes.

Where the Indicative Route Method (IRM) to date is used to work on strictly preplanned routes and relies solely on local pathing methods like social forces to resolve local issues, our lanes are dynamic and sensitive to local conditions. This means we still rely on social forces or other local methods to deal with collisions, but our lanes change based on the effects of the local methods. Where traditionally preplanned paths forced characters back to their old indicative route or required full replanning, dynamic lanes change based on the local position of the character automatically. We then compute the actual lanes over small segments of the ECM only when needed by a character. Therefor our lanes are a good way of introducing a meso level of planning, between the social forces (micro) and global planning (macro) levels.

5.1 Criteria and objectives for lanes

In Section 4 we list observations and describe what kind of paths are desirable as indicative routes (See Section 3.3). There are several variations on how to create these kind of paths which roughly match our observations. They vary both in how their edge segments are defined and in which points they connect. Therefor we note three criteria crucial to the considerations in this thesis here.

- Completeness.
- Matching observed behavior.
- Efficient computation.

Completeness guarantees that anywhere in the free space, applicable lanes can be found. This ensures the definition is robust and widely applicable. To fulfill this requirement, we have to be able to handle difficult cases. Examples of difficult cases are forks, intersections and sharp corners. Some approaches can not provide useful lanes covering intersections and would require characters to make their way across without guidance. At intersections or forks, lanes for each of the possible paths should be available.

We will use the terminology *"a set of lanes"* to describe lanes running parallel along a given section. Figure 11c in Section 4.2 shows an example of how such sets can overlap. To have lanes for all possible global paths, at every n-degree vertex in the Voronoi graph G, we need $\sum_{i=1}^{n-1} i$ sets of lanes.

Matching our observations is critical because of our intent to run IRM attraction points for the characters along the lanes. If the resulting lanes are not

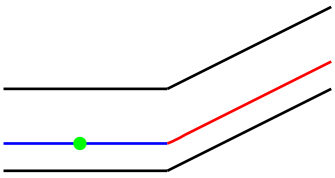


Figure 12: Figure showing a simple example of a derived lane segment resulting from the current segment. A character (green) traveling along a lane segment (blue) and the upcoming segment (red) derived from the previous lane segment and the bordering edges.

representative of paths characters would take according to our observations, the resulting paths when we use lanes as indicative routes will not be realistic.

Though paths handed to IRM are not followed strictly, nor was that ever the intention, and social forces have a significant influence, a quality path given to IRM should yield a good resulting path. The motivation for this thesis was to provide more relevant paths for IRM to run on. Paths that are sensitive to local conditions. To reach realistic results these paths have to match our observations.

Especially in regions where the clearance changes drastically and corners, it is important to relate the lanes to the observed behavior. Examples of such regions are shown in Figures 11a and 11b in Section 4.2.

Efficient computation is vital to the applicability of the method. Lanes must be efficient to compute, store and derive attraction points from. This last requirement is lessened by the fact that current implementations of ECM/IRM depend heavily on offline computation already, so adding some cost to the preprocessing is acceptable. The real-time performance is much more crucial so it is essential for it to be efficient to get a path from the stored information. It should be noted that efficient computation should not come at a significant cost to the quality of the paths.

To evaluate the efficient computation criteria, we look at how easy it is for a character to find lane sections extending from the current lane section it is on. This means that we benefit heavily from lane sections having exact connectivity and being easy to derive from the information stored. An example of this is shown in Figure 12.

5.2 Alternative lanes on the ECM

There are several candidates to base our lanes on. In this section, we will look at some of these alternatives and reason which is best suited for our purposes based on the criteria in Section 5.1.

5.2.1 Medial axis parallel lanes

We can obtain lanes by running them parallel to the medial axis. For every point p of the medial axis M we can calculate the normal n at every point p of that section of M. We then calculate the distance d between a character and the closest point on M. We then move every point p in the direction of the normal n with the distance d. This gives us lanes parallel to the medial axis.

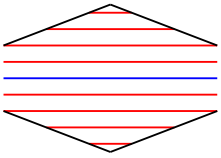


Figure 13: Figure showing a set of lanes (red) running parallel to the medial axis (blue).

The benefit of this approach is that we have all the information readily available in the ECM. One of the downsides however is that this gives a non-continuous curve. We do have to do stitching and trimming at points between segments of the medial axis. On the side where the angle between the normals is less than 180 degrees, we will see overlap and on the other side we will see gaps. As said we can resolve this with stitching and trimming to obtain proper connectivity and completeness, but at the cost of additional processing. Held (1998) shows how we can accomplish this.

Unfortunately there is an inherent problem with this set of lanes when the clearance increases or decreases highly along a section of the medial axis. Figure 13 shows this issue. In these regions, lanes that run parallel to the medial axis will not cover the whole space well. Lanes running parallel to the medial axis will be created and cease to exist as the clearance varies. If we simply move the character inward when its lane ceases to exist, the reaction to the region narrowing will be far too late and many characters will be found against the boundaries. Characters walk straight relative to the medial axis, until they can not anymore.

The behavior we want to achieve is for characters to slowly move towards the medial axis as the clearance shrinks and slowly outwards as the clearance increases. The desired behavior is shown in 11b in Section 4.2.

5.2.2 Shrinking corridor border lanes

Another way of obtaining lanes would be to use the borders of corridors. Instead of using an offset from the medial axis, this would effectively mean taking an offset from one of the borders. We have the information on the local borders available and the ECM can quickly generate corridors. By shrinking the corridor at different lengths and using each border, we can create sets of lanes. Shrinking corridors is an operation generally used to ensure clearance, the radius of the character dictates how much the corridor is shrunk. Here we shrink corridors until a character is on one of the borders and use that border as an indicative route.

This solution ensures connectivity of the lane segments and unlike segments parallel to the medial axis they adjust according to the clearance. Once you have a corridor, it would be simple to compute where the next attraction point should lie.

There are two major disadvantages to this method. The first is a result from social forces influencing a character's movement. When a character deviates from the lane, we have to shrink (or enlarge) the corridor again. We could compensate for the computational costs of shrinking the corridor by only shrinking it where needed to avoid shrinking the entire corridor every time. Al-

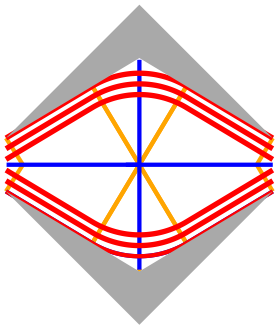


Figure 14: This figure shows a set of lanes (red) resulting from shrinking corridors. The medial axis is shown in blue and lines to the closest points in orange.

ternatively, rather than using corridors explicitely, we could calculate the edges as need it on the ECM. Proper storage and only recomputing as needed and as far as needed will alleviate this concern. The second and more crucial disadvantage is that these lanes only keep a given distance from one of the borders. This means that characters following these lanes do not seek to keep their relative position within the region, but their relative position from one side. Where Section 5.2.1 presents an alternative in which characters do not spread out or group up sufficiently when clearances vary, this method results in spreads that are too extreme. Characters will avoid the medial axis drastically with increasing clearance. Figure 14 shows this behavior which again is very different from the desired behavior shown in Figure 11b in Section 4.2.

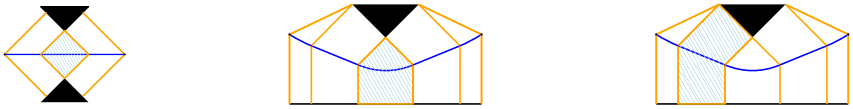
5.2.3 ECM density cell lanes

A different way of generating lanes would be segmenting the free space and creating sets of lanes for each segment. As long as connectivity is ensured from one segment to the next, this achieves completeness. A common way of segmenting the space when using ECM is a segmentation based on *ECM density cells*. Once we have our cell division we create lanes from each open end of the cell to the others. There is one degenerate case in that we also need lanes starting at end vertices to all open edges.

The ECM is based on a Generalized Voronoi Diagram (GVD) as explained in Section 3.2. The Voronoi vertices are connected by edges. The edges consist of a series of nodes with closest point information for the obstacles that define that subsection of the edge on each side.

To handle and store density information on the ECM, implementations of crowd simulation methods such as Pedestrian Dynamics (by INCONTROL Simulation Solutions) use *ECM density cells*. A cell is defined by the area between the closest points on a given subedge and the nodes themselves. Though cells in the ECM have not been defined in literature as such, we also see them utilized in point location and research on handling dynamic updates in ECM implementations.

Here, we present a definition of an ECM density cell in the ECM: *An ECM density cell $C(n_i, n_{i+1})$ in the ECM is collision-free polygon of which the border is defined by the position of an ECM node n_i and its left and right closest obstacle*



(a) (b) (c)

Figure 15: Figures showing various shapes of density cells.

points cpl_i *and* cpr_i*, and the position of ECM node* n_{i+1} *and its left and right closest point* cpl_{i+1} *and* cpr_{i+1}. Closest obstacle points are stored on the ECM nodes. At Voronoi vertices, more than two closest points are available, but only two that relate to each incoming or outgoing edge.

There is several resulting cells to consider:

- A quadrilateral; The shape of ECM density cell C will result in a quadrilateral when both closest obstacle points from both nodes n_i and n_{i+1}, cpl_i and cpl_{i+1}, and cpr_i and cpr_{i+1}, share the same location, or if the two closest obstacle points from nodes n_i or n_{i+1}, cpl_i and cpr_i, or cpl_{i+1} and cpr_{i+1} share the same location. This can happen for instance because obstacles are often split into convex segments or at the boundary. An example of a quadrilateral cell is shown in Figure 15a.

- A pentagon; If one side of the closest obstacle points for nodes n_i and n_{i+1} match (cpr_i and cpr_{i+1}, or cpl_i and cpl_{i+1}), the shape of ECM density cell C is a pentagon. Figure 15b shows an example of a pentagonal cell.

- A hexagon; If none of the closest obstacle points for the nodes n_i and n_{i+1} share the same location, ECM density cell C takes the shape of a hexagon. Figure 15c shows an example of a hexagonal cell.

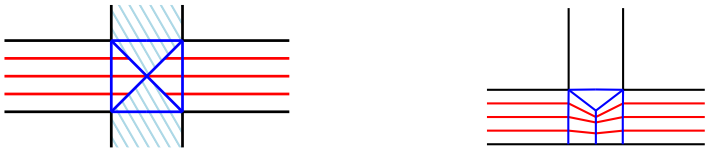
We are guaranteed these polygons are convex because the node would be invalidated if the node does not lie to the outside or on the edge between the closest points. Note that because we are not considering the medial axis segment, a node could lie on the line segment between its closest points which can also reduce the degree of the polygons.

Using these cells, we can segment the free space and scale down the problem to creating lanes from one end of the cell to the other. As all the open cell edges are connected, connecting the segments for a single Voronoi edge is straightforward.

Problems arise with lanes around vertices. At intersections, such as the one shown in Figure 16a, the ECM density cells we want to connect to form a path are only connected at the vertex, not the open edges. A character will want to move from the left cell to the right cell, but there is no connection between the two aside of the vertex.

Figure 16b shows that where paths split, we can not properly connect the lanes between the bordering cells because they are only bordering on half the open edge. The left cell and right cell at the intersection only border below the intersection point, above the intersection point they both border with the top cell at the intersection which is not part of a path from left to right.

These problems occur as soon as there is more than one direction, so at every Voronoi vertex. We can not connect the cells directly seeming cells along



(a) This figure shows an area marked in light blue that is not part of the region that is considered part of the path. This is because those cells are created by nodes of edges that split off at the intersection. The interuption of the red line segments along the path show how this results in incomplete lanes because the cells on the left and right of the vertex only border eachother directly at the vertex.

(b) This figure shows a set of lanes around a T-split and highlights that the resulting lanes show an awkward dent from the top. This is because the two cells only border on the lower end. The upper cell is not part of the path because it belongs to a different edge.

Figure 16: These figures shows traditional ECM density cells as used for handling density and point location. The ECM density cells are formed by the blue open edges and black obstacle edges. The red line segments show potential paths.

the path are only connected partially or even only at the vertex. Because there are no usable lanes around these points, characters would be required at intersections to look ahead until they find lanes useful for their specific path again without any guidance. This approach does not provide completeness. Seeming occasionally in environments series of vertices can be directly connected, this can lead to situations where there is no lane visible from the character's current position.

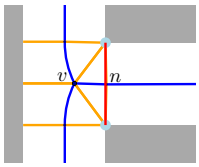
Additionally, this also shows that if were looking at density along an edge using these density cells, we will miss areas we would reasonably want to consider. Figure 16a highlights the missed areas part of which should have clearly been considered for accurate results. Therefor this approach will not satisfy the completeness criteria.

To find paths on ECM density cells we can use methods similar to those that are shown in Section 6. Here we can state that for an ECM density cell C we only need paths from any open edge e_i of C to any other open edges e_j of C. Because of this we can opt for curved paths which are very precise or in straight segment paths to gain some computational benefit in practice but at a loss of accuracy.

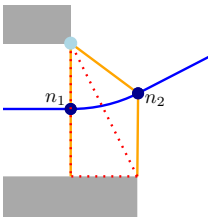
5.2.4 Closest Points Cell lanes

The ECM density cells addressed in Section 5.2.3 posed us with connectivity problems. If cells are not correctly connected through an overlapping edge, it is not possible to offer fitting lanes. To address this problem we can utilize a different cell division that does allow us to provide lanes around vertices properly. One such candidate, which we introduce here, is the Closest Points Cell division. We explain the exact construction of Closest Points Cells from the ECM in Section 5.3.

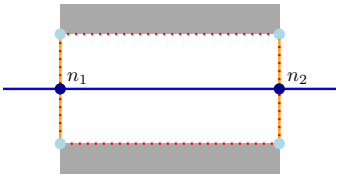
As we will show, all the information required for this cell division is present in the ECM. It is important to note that this means that the CPC division does



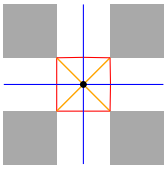
(a) A CP Cell shaped as a line. Because the node at vertex v and node n share two closest points (shown in light blue), a cell of the line segment between the closest points is the result. This line cell is shown in red.



(b) A CP Cell shaped as a triangle shown with the dotted red outline. Because the node n_1 and node n_2 share one closest point (shown in light blue), a triangle cell between three closest points is the result.



(c) A quadrilateral CP Cell shown with the dotted red outline. Because nodes n_1 and n_2 do not share any closest points (which are shown in light blue), a quadrilateral cell between the four closest points is the result.



(d) A quadrilateral CP Cell shown in red. This cell is the result of a Voronoi vertex with four closest points.

Figure 17: Figures showing various shapes of CP cells.

not necessarily have to be computed explicitly to utilize this division.

For now we will offer the following definition of the CPC: *The CPC division connects the closest obstacle points of the GVD. Every node (event point) n of the GVD becomes an edge $e \in E$ in the CPC. Every subedge s and vertex v of the GVD becomes a face of a cell C in the CPC.*

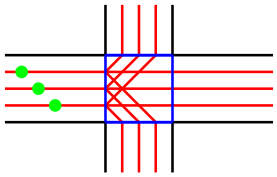
This leads us to the following two types for a CP Cell:

1. The cell $C(n_i, n_{i+1})$ created by two nodes n_i, n_{i+1} neighboring on an edge of the ECM is the area defined by the two closest points of n_i and the two closest points of n_{i+1}.

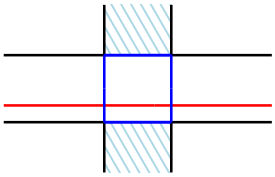
2. The cell $C(v_i)$ created by a vertex v_i of the ECM is the area defined by all closest points of vertex v_i in counterclockwise order.

The properties of the closest points in the ECM guarantee convexity. The CP Cells $C(n_i, n_{i+1})$ of nodes n_i, n_{i+1} can achieve 3 different geometrical types of cells.

- A line; If both of the closest points at node n_1 match the closest points at the node n_2, we end up with a line, in practice we can often filter these out by checking for this condition as they do not add anything for our purpose.



(a) Three sets of lanes (red), one for each character (green) moving from the left in each direction.
Open cell borders shown in blue.



(b) A lane that could be used by characters shown in red.
Open cell borders shown in blue.
Highlighted are the density areas that will not be considered when moving along the lane.

Figure 18: Closest Point Cells used as introduced.

- A triangle; If only one of the closest points at node n_1 matches one of the closest points at node n_2. Additionally, when the two closest points at n_1 or n_2 match, which could be the case with non-convex obstacles, we also get a triangle cell.
- A quadrilateral; When none of the closest points at node n_1 match the closest points at node n_2 we get a quadrilateral. Two opposing edges represent the obstacle borders and the line segment between two closest points of a node connect them. Because no obstacle border can have a point closer to the node than the closest points we are ensured this quadrilateral is a convex cell.

For a CP Cell $C(v_i)$ around a Voronoi vertex v_i we can state that because the closest points have an equal distance to the node, we end up with a convex polygon where the number of nodes is equal to the degree of vertex v_i. An example of such a CP Cell resulting from a Voronoi vertex can be found in Figure 17d.

Given this cell division we are still left with the issue of deriving a lane for a character in a cell. An open edge e in a CP Cell C is an the edge not formed by an obstacle boundaries. For a CP Cell $C(n_i, n_{i+1})$, these are the edges e formed by the edge between closest points cpl_i and cpr_i of node n_i and closest points cpl_{i+1} and cpr_{i+1} of node n_{i+1} responsible for forming the cell. For a CP Cell $C(v_i)$ these are all edges of $C(v_i)$ following from clockwise or counterclockwise neighboring closest points.

The goal is to create lanes from any open edge of a cell C to all other open edges of C. Two methods for doing so are shown in Section 6.

The approach outlined here handles intersections and difficult cases that caused problems for the subdivision in Section 5.2.3 much better, as shown in Figure 18a. At Voronoi vertices we have $\sum_{i=1}^{n-1} i$ overlapping sets of lanes where n is the number of open edges of the CP Cell. The lanes all directly connect the two open edges that overlap with neighboring cells unlike the traditional density cells. Figure 18b shows that the CP Cells much better represent the relevant space than traditional density cells (Figure 16a). In Section 5.4 we show cases where this approach does have some weaknesses, but the lanes are still valid and it is universally applicable.

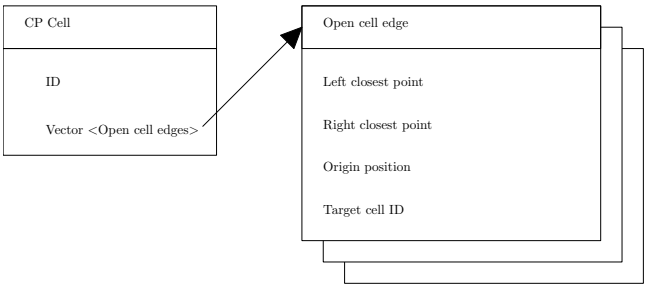


Figure 19: An object diagram for minimal data of a CP Cell. A CP Cell has an ID and a Vector with Open cell edges. An Open cell edge has the positions of its two closest points, the position of the node or vertex from which the edge originated from, and the ID of the Cell on the other side of the cell edge.

5.3 Constructing the CPC division from the Explicit Corridor Map

CP Cells contain a list of the open cell edges and an id. Additionally this is a good place to store, for instance, density information. The open cell edges contain the two closest points that define the line segment, the position of the node or vertex that the closest points belong to (its origin position), and the id of the target cell we end up in if we traverse this open cell edge. This means the structure for a CPC division as used in this thesis is as shown in Figure 19.

CP cell:

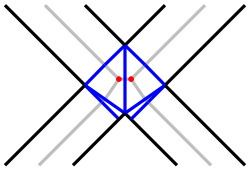
- ID.
- Vector of open cell edges.

Open cell edges:

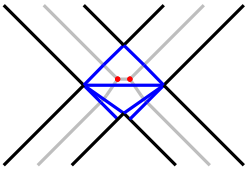
- Two closest points defining the edge.
- Position of the node or vertex the closest points belong to.
- ID of the target cell if edge is traversed.

Converting the ECM to the CPC division can be achieved in O(n) time where n is the number of nodes in the ECM. This with the understanding that at a vertex v in the ECM, there exist d nodes, where d is the degree of vertex v. We can accomplish this by starting at the root vertex of the ECM while it is represented as a directed graph, and traversing the edges of the graph from the ECM, creating the cells as we go. It should be noted that all the appropriate information is directly available, and as such, converting the ECM explicitly to a CPC structure is not required if you do not also intend to use it for other purposes such as storing density and other region specific data.

While traversing the directed graph, we need to handle two cases. A case for the start or end of an edge and a case for cells based on a pair of neighboring nodes. Because a vertex is reached multiple times, namely for each incoming and outgoing edge, we ensure it is only handled as a special case once by labeling a vertex as handled.



(a) A CP Cell division with the open cell edges in blue. The Voronoi vertices shown in red and GVD in gray.



(b) A valid possible generic cell division with the open cell edges in blue. The Voronoi vertices shown in red and GVD in gray.

Figure 20: Comparing a random cell division based on the closest points to the CP Cell division.

Whenever we reach a vertex: If a cell has not yet been made for this vertex, create the cell with a new *ID*. Add the open cell edge to the new cell based on the *closest points* and the *vertex location*. The *target* should be the previous cell. Add the open cell edge to the previous cell we handled based on the *closest point information* and the *location of the last node or vertex* handled. The *target* should be the new vertex cell.

If the vertex was previously handled, only add the open cell edge to the cell based on the vertex and the last cell in the same manner.

When we hit a node (except the first node of the edge): Create a cell with a new *ID*. Add the open cell edge based on the *closest point information* of the last previous node of the edge and its *location*. Make the *target* the previous cell handled (in case this is the second node of the edge, this target is a vertex cell). On the previous cell add the same open cell edge, only let the *target* be the new cell.

We did not address the deprecated case of neighboring nodes with identical closest points creating a cell. It is trivial to see those cells can easily be skipped here, considering cells consisting of just a line do not cover any regions.

We can complete this process in the promised $O(n)$ runtime bound. After the CP Cells are all created, it is recommended to sort all the open CPC edges at vertices in clockwise order. This makes operations such as finding the next left or right open edge more efficient.

5.4 Benefits and downsides to using Closest Points Cell division for deriving lanes

We have shown why a cell division is useful in Section 5.2 by offering it as a means of handling difficult cases such as forks and intersections. However aside from showing that cells as they were being used on ECMs are not suited for our purpose, we have yet to show why these CP Cells are better than any other division. We will address this by comparing closest points cells to a generic triangulation. A generic triangulation on the closest points constrained by the obstacle edges ensures the same purpose of separating the regions that are relevant to forks and intersections.

5.4.1 Relation between the Closest Points Cell division and ECM

The biggest issue with using a generic triangulation or any generic subdivision is that we lose the relation to the GVD upon which the ECM is based on. We are still subdividing over closest points, but if two Voronoi vertices are near one-another without any additional nodes in between the two, there is a chance the resulting triangulation will not tell us anything about how the vertices are connected. This means to plan global paths we would have to construct a new graph and many of the methods available on the ECM will not function anymore. Figure 20 shows how a generic cell division based on the closest points can lack a relationship to parts of the GVD.

5.4.2 More division than is needed for convexity

In some scenarios, CP Cells might divide the space more than is required for the characters to traverse the space. If a section widens and then tightens again, we can end up with two cells where visibility can still be ensured.

However, we need to take into account that we use this changing width and expect different character movement in each case. When the walkable area widens, we expect characters to move apart and utilize the available space more. When the walkable area narrows, we expect characters to merge closer together.

This is also why CP Cells are intuitive and show why it is natural for characters to progress from open edge to open edge just as characters move from subedge to subedge within other ECM based methods.

When less cells are desirable even at the cost of showing the behaviors this approach was designed for, cells that share visibility can be merged. However it is difficult to merge cells and ensure convexity within the new cell in all cases. Convexity is desirable because it ensures our paths will not intersect any obstacles even though they are generated from only two points on the respective open borders of a cell.

5.4.3 True closest obstacle point

A disadvantage of using a subdivision is that we no longer have information about the true closest point of a character within the cell. When near an open edge, it is common for the closest obstacle point to lie on the edge of an obstacle not involved in the creation of the current cell.

This can lead to severe problems if we do not enforce some strict boundary, characters could get pushed out of the cell and into nearby obstacles. Of course the cell boundaries themselves could be used, but it is not unlikely for a character to want to move outside of the cell because of local conditions when possible.

To handle this, we could opt to check all the nearby obstacles, but because there could be a series of thin cells we would often have to check a considerable number of obstacles at real time and it is not clear how best to find these obstacles in the first place. Fortunately we can apply a principle that corridors (Geraerts & Overmars, 2007a) use to obtain more allowed walkable area.

A character might want to enter a cell because of local conditions that is not part of the path we planned. This happens only when we find cells with open edges not part of the path. This can happen in two cases.

- A character in the first or last cell of a path.

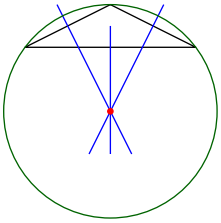


Figure 21: This figure shows an example of the circumcenter of a triangle lying outside of the triangle itself. The triangle is shown in gray, with its perpendicular bisectors in blue. The circle through the three points of the triangle is shown in green and the circumcenter in red.

- A character in a vertex based cell.

Knowing when this can occur, to show how to obtain more walkable area, take a triangle CP Cell created from a vertex.

We know the closest points forming the cell are all equal distance from the vertex, making the vertex the circumcenter of the triangle. From this we can deduce that the vertex is also the center of the circumcircle of the triangle.

Because we know no obstacle point can be closer or as close to the vertex as its closest points, it is safe to use this circle as an extreme boundary when traversing a CP Cell from a vertex. Because additional closest points from a vertex of a higher degree will lay on the same circle, it is safe to extend this to all CP Cells created from vertices.

With similar reasoning we can argue that there can not be any obstacle points closer to a node than its closest points, so a circle with its origin in the node, through the closest points should be safe for a character. As long as we respect these outer boundaries, we are ensured characters will not collide with the static obstacles.

This approach is not optimal, in the sense that we still could be cutting off a considerable amount of space that, in theory, the character could want to end up in because of local conditions regardless of its path. However it gives us access to the exact same regions ECM corridors cover in these scenarios now because the ECM corridor would be limited by the exact same circle.

We can also deduce one thing from this that may be unintuitive and should be noted. One of the geometric properties of the circumcenter of a triangle is that it does not have to lie inside the triangle itsself as shown in Figure 21. This also means there is no guarantee that the nodes and vertices that form a CP Cell are inside the cell themselves. Though this has had no adverse effects, it is very much unlike the cells that we see used in ECM traditionally. In traditional density cells, the node is part of the border of the cell following the definition in Section 5.2.3.

5.4.4 Lack of symmetry at intersections

One of the main issues with using the Closest Points Cell division is that some intersections can generate lanes that are not handled symmetrically. In practice this means that in worst case, lanes only cover half the intersection depending

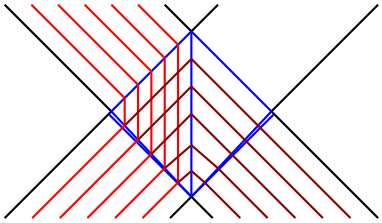


Figure 22: This figure shows how sets of lanes on a not perfectly square intersection look different depending on the edges. The lanes are shown for a character coming from the bottom left. If the character goes to the upper left, lanes cover about half the intersection. If the character goes to the lower right, lanes cover the full intersection.

on the path as shown in Figure 22. Because the ECM is often generated using an approximation method, we can see this occur in a lesser extent fairly regularly in practice. Figure 22 shows how though the lanes are still usable and fit our criteria, sometimes we make more use of the available space than others.

As noted this same problem exists with the areas that corridors cover in this cases as well, but it is less apparent in there because corridors work with circular arcs. In theory we can do this as well. If we allow lanes to leave the cells across open edges, we can cover most of the relevant area. This works just as explained in Section 5.4.3 which addresses allowing characters to temporarily cross over open cell borders due to local conditions. Unfortunately for many implementations this will lead to expensive operations when deriving lanes in these segments at real-time. This makes it debatable whether this fix should be applied in practice and shows that there is still room for improvement.

5.4.5 Constrained Delaunay triangulation

As an alternative to using the CP Cell division, we could opt to transform the cells into a triangulation, where a constrained Delaunay triangulation turns out to be desirable, simply by inserting appropriate edges. However restricting the cells to a triangulation can have an adverse effects when considering Sections 5.4.2 and 5.4.4.

Though we showed in Sections 5.4.2 that the CP Cell division was not stricter than needed, the same arguments can not be applied when using a Constrained Delaunay triangulation. To justify splitting up many cells, additional requirements or benefits would need to be present.

The primary issue with using the CP Cell division as shown in Section 5.4.4 is even more prevalent in a Constrained Delaunay triangulation. It is impossible to obtain a meaningful triangulation around high degree vertex cells that handles each path evenly without additional computations as previously shown. The Constrained Delaunay triangulation only enhances the downsides to this approach. So in most applications, this is not recommended.

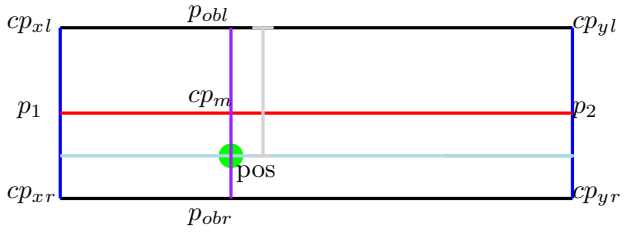


Figure 23: This figure shows the calculation of the middle line m (shown in red) from two open edges (shown in blue). Additionally it shows the points and line segments used to calculate the offset ratio, such as the line segment from the left hand obstacle edge through the character's position in purple. Lastly it shows the resulting lane in light blue.

6 Creating paths on cells

In this section, we offer two variations of lanes that can be created using the Closest Points Cell division as discussed in Section 5.2.4. The focal point will be determining an offset accurately representing a character's position within a cell. Using this offset, in Section 7 different attraction points are determined. The two variations offer different types of paths and performance, differences which most clearly can be shown around Voronoi vertices.

6.1 Straight lanes

For a set of lanes to be useful, they need to connect an open cell edge e_x to another open cell edge e_y of the same cell C. The complete set of lanes should be placed within the walkable area. And there should be a set of lanes for every combination of two open edges belonging to a cell C.

The most straightforward way of connecting two open cell edges is as shown in Figure 23. First we compute the offset within a cell between 0 and 1. With that offset, we can computing those points on open cell edge e_x and e_y of cell C. We then connect these points with a straight line segment.

Convexity of each cell guarantees that are lanes are fully within the walkable area, seeming we connect two points that are part of the polygon. The straight line segment ensures we can efficiently compute attraction points or process them otherwise.

To calculate an offset, the first step to accomplish is finding the appropriate middle line m of a cell. We find m by taking the two open edges e_x and e_y we want to connect, and selecting the points p_1 and p_2 on the open edges that are equal distance from the closest points. We connect p_1 and p_2 with a line segment to form the cells middle line m.

Once we have the middle line m we take the character's position pos and calculate the closest point cp_m on m from pos. We then find the point on the obstacle edges p_{obl} and p_{obr} that intersect with the line through cp_m and pos. We proceed to calculate the ratio of the distance between the p_{obl} and pos and the distance between p_{obl} and the p_{obr} to obtain an offset describing the position of a character in a cell relative to the boundaries.

Offset $o = \frac{|pos - p_{obl}|}{|p_{obr} - p_{obl}|}$

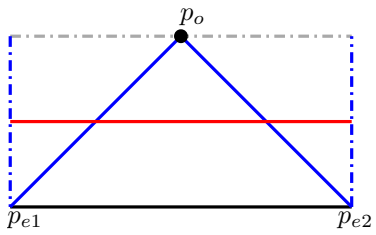


Figure 24: This figure shows a how we find a straight middle line in red. The obstacles are a point obstacle (p_o) and line segment obstacle (p_{e1} to p_{e2}) both shown in black cell. The open cell edges are shown in blue.
By mirroring the line segment obstacle through point p_o which is shown with the gray dotted line, we can now calculated an offset ratio as described previously. This figure also shows that the middle line segment within the cell lies exactly between the point obstacle and a point on the line segment obstacle.

With this offset we can take the open edges of any cell and connect these to form the lanes we want. This leaves us with a straightforward and fairly efficient 5 step plan to calculate the lane segment as it is shown in Figure 23.

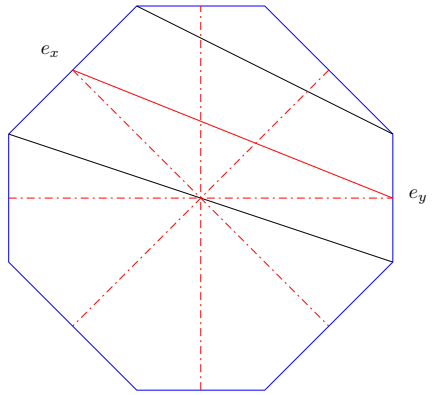
1. Get middle line of cell for relevant open edges.
2. Find closest point on middle line for character.
3. Find intersections between lines of obstacle edges and line through the closest point on middle line through the character.
4. Calculate the relative distance between the character and the left hand obstacle edge intersection point, to the distance between the right hand and left hand obstacle edge intersection points.
5. Use this offset ratio on the two open edges to get two points and connect the points to form a lane.

It should be noted that there is two degenerate cases that can occur. First the obstacle edge on one side can be a point. We can address this by simply making the line based on the obstacle edge a line parallel to the middle line, through the obstacle point. This works because we are effectively looking for the middle points on the open edges anyway. This is shown in Figure 24.

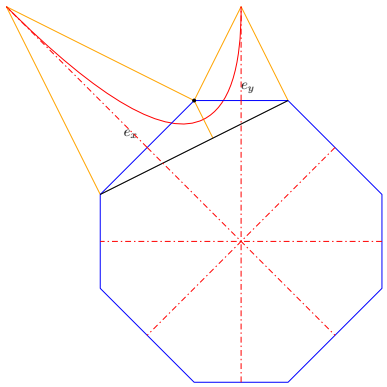
Second, if the character lies exactly on the middle line, the line from the middle line through the character is poorly defined. If this is the case, the offset ratio is 0.5 and the middle line itself forms the lane.

Once we have an offset ratio for the character in a segment, you can apply that same offset ratio to the next open edges that the character is planned to path along to get the next lane segment and so on to derive the lane as far as you need it along the path.

It is important to note a straight line can not represent the medial axis segment between a line and a point. When considering a point obstacle border and a line segment border, the middle line of the cell as presented in this segment



(a) A CP Cell around a vertex with the ECM medial axis shown with dotted red lines. The open cell edges are shown in blue. Open cell edges e_x and e_y are selected, with the relevant borders following from that shown in black and the medial axis between the two in red.



(b) A CP Cell around a vertex with the ECM medial axis shown with dotted red lines. The open cell edges are shown in blue. Open cell edges e_x and e_y are selected, with the relevant borders following from that shown in black and the medial axis between the two in red. To show where the curve for the medial axis comes from, closest point lines from the ECM are shown in yellow.

Figure 25: These Figures show vertex CP cells, highlighting the different types of middle lines within vertex cells.

does not represent the point exactly between the two closest points on the borders of a cell. What it represents within a cell is the middle between the point obstacle border and a point on the line segment border.

Though the method outlined in this section is efficient, it should be apparent that it has its flaws. They stem from the fact that line segments are used for lanes and the offset is calculated from a straight line.

When considering a cell with a point obstacle border and line segment obstacle border, the side of the point obstacle border has a much smaller area. A consequence from this is that in practice many characters will end up with seemingly large offset ratios simply because there is more space for the characters there.

Additionally, as shown in Section 4, people do not follow straight line paths during corners, but make smooth turns. The effect of using line segment lanes is of course lessened by the fact that we run attraction points along the lanes to attract the characters as in traditional IRM. These methods were designed in part to achieve smooth paths off straight line segment indicative routes.

6.2 Curved lanes

Rather than opting for line segment lanes, another option is to calculate paths and offsets off curves that split the cells with a middle line based on the closest points on the two borders. For this, we calculate a medial axis for each

combination of open cell edge e_x and e_y of cell C. This is needed because this medial axis between two cell borders is different from the medial axis of the ECM around vertices as shown in Figure 25. Figure 25b shows how the curved lane approach differs from the straight lane approach.

The borders for this local medial axis m consist of two types. A point and a line segment, when open cell edge e_x and e_y have a closest obstacle point in common, or line segments when they do not.

1. The line segments can either be closed cell borders as is the case in at least one edge of any cell C generated by two nodes.

2. The line segment can be an open cell edge which is not part of the path at a cell C generated by a three or higher degree vertex.

3. The line segment can be a line segment through the interior of the cell C, which happens if C is generated by a four or higher degree vertex and there are two or more open cell edges on that side of the cell border between the two open cell edges that are part of the path.

Now that we have defined the borders between which we want to calculate the medial axis, we can use traditional methods to compute them where needed. They can of course be computed on the fly or during preprocessing and stored, depending on the needs of the application. When the borders are both obstacle edges, the medial axis as used in the ECM can simply be copied.

The resulting lanes form curves much like the observed paths shown in Section 4. This means that even if we compute an attraction point only one stepsize ahead of the character, characters will still show smooth paths while following the computed lane very closely.

However as Figure 25b shows, the medial axis intersects the open cell edges rather unevenly. This would mean that if we computed an offset in the same way as we did for straight line lanes, we would get very uneven results. In practice it would almost seem like the side with the point obstacle border would be handled at a higher resolution.

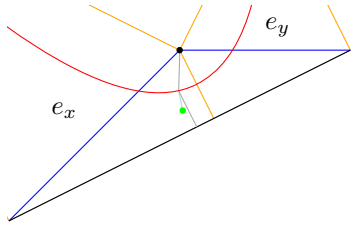
We can deal with this by slightly changing how we calculate the offset. We use the local medial axis m, and calculate a relative offset o based on the distance from the character's position to the closest point on the medial axis relative to the clearance at that point of the medial axis. Making a path with this offset o along the medial axis will connect the ECM density cell edges we are attempting to form a path between. Figure 26 shows this in practice.

6.3 Dynamic property

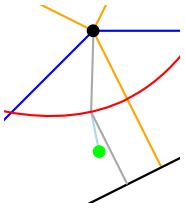
The offset, though different in both implementations, is what gives us the sought after dynamic property. As soon as the relative position of a character within the segment changes, the lane adapts.

These changes can be a result of social forces or simply cutting corners when characters can look far enough ahead to set their attraction point far away.

If there is no influence of other characters, and we set the attraction point to be the furthest visible point on the lane, the character is guaranteed to automatically take the shortest path. This is ideal, because the shortest path is a good



(a) A partial CP Cell around a vertex as shown fully in Figure 25b. The open cell edges are shown in blue. Open cell edges e_x and e_y are selected. The relevant borders following from those cell edges are shown in black and the medial axis between these borders is shown in red. The distance between the character is shown in blue and the clearance in gray.



(b) The same partial CP Cell but enlarged. The open cell edges are shown in blue. Open cell edges e_x and e_y are selected, with the relevant borders following from that shown in black and the medial axis between the two in red. The distance between the character is shown in blue and the clearance in gray.

Figure 26: This Figure shows how to calculate the offset ratio for vertex CP Cells.

indicative route when we are looking at a single character in an environment. Only when crowd density becomes higher do we want characters to follow their lanes more strictly.

7 Selecting attraction points on dynamic paths

In this section we will offer two alternative ways of finding *attraction points* for the Indicative Route Method (IRM). These provide benefits when lanes are used as indicative routes. Though it is unfortunate that we lose some of the ease with which attraction points are calculated traditionally (Karamouzas et al., 2009), these alternative approaches allow us to obtain different paths depending on density.

We achieve this by making the distance a character can look ahead to find an attraction point sensitive to the density. When density is not a factor, it is desirable that characters follow the shortest path. If we find the furthest point of the lane that is visible from our current position, we inevitably end up on the shortest path as shown in Section 3.4.

In dense situations, we want them to follow the lanes more strictly. By not looking ahead as far, we automatically follow the lanes more closely. First, Section 7.1 provides a way of finding valid attraction points to follow lanes without the ability to adapt the distance as an intermediate step. Following, Section 7.2 shows how to compute attraction points on lanes that can be capped by distance and if not capped will lead to the shortest path. Lastly, Section 7.3 lists conditions under which we can keep attraction points for a character rather than recomputing it every step.

7.1 Utilizing circles to locate attraction points on Closest Points Cells

Karamouzas et al. (2009) computes attraction points by retracting the position of a character on to the medial axis and defining a circle around that point based on the clearance as explained in Section 3.3. In Section 6, we have indicative routes planned on Closest Points Cells. Here we addapt IRM slightly to adapt to this.

The most important property that we have to ensure still holds, is that the selected attraction point is visible and will not cause the character to collide with static obstacles along the way. Because CP Cells are convex, we know that any point in the cell that the character is located in, can be reached. Of course, once we approach the edge of a cell, that is no longer useful information seeming the attraction points would be so close to the character that it would have to slow down until it ends up on the edge.

The cell edge as introduced in Section 5.4 provides additional information. We have the position of the node or vertex to which the closest points belong too. As this point is part of the medial axis, no obstacle can be closer to this point than the closest points. This means that we can use this circle to find areas outside of the cell that are safe to select attraction points within.

Figure 27a gives an example of this. The character is in the left cell, but because of the open circle property, more than just area contained by the cell edges is guaranteed to be free of obstacles.

We can use this same principle when looking at neighboring cells. If a character is inside the circle defined by a cell edge of a given CP Cell, the region of that cell, as well as the regions defined by the circles of any other cell edges of that cell are valid regions to find an attraction point in.

(a) Here we highlight the area resulting from the left cell that is guaranteed to be visible from the character's position.

(b) Here we highlight the area resulting from the right cell that is guaranteed to be visible from the character's position.

Figure 27: Here we highlight the regions of a CP Cell division. For a certain cell and character position we mark the area that is guaranteed to be visible in light blue. The character's position is shown in red. Open cell edges are shown in blue with the circles defined by the nodes of the open edges and its clearance shown in gray.

Figure 27b shows an example of this. The character is not in the cell on the right, however it is inside the circle which has its center at the ECM node and its range defined by the closest points. Because of this, we know that every point within the cell is visible from the character's position, even if it is not in the cell itself.

To achieve smooth paths, it is crucial to run an attraction point along the path. What we can do here is draw a circle around the character's position with a radius depending on how far we want the character to normally look ahead when selecting its attraction point. We can then look at the upcoming lane segments until we find an intersection with the circle or we find that the path has left the safe region. Whichever point is found will be the attraction point.

Because lanes are based on a character's offset, a circle surrounding the character is guaranteed to find an intersection unless the goal is inside the circle. We do not have to be concerned that the lane somehow lies completely outside the circle and no points can be found.

This method is simple and efficient. Unfortunately, it does not look far enough ahead in many cases. We have noted that it is desirable for us to end up at the shortest path if there is no influence from crowd density. To achieve that we often have to look further ahead then this method will allow because we are limiting the attraction point selection to an allowed region.

7.2 Utilizing funnels to obtain attraction points

To achieve that characters follow the shortest path, even if we are not running an attraction point along the shortest path, attraction points should be based on the furthest visible point of the indicative route. The indicative route in our case is the dynamic lane. Guibas et al. (1987) and Demyen and Buro (2006) show how to find a shortest path in a triangulation. We can utilize this to find the shortest path in a corridor as described in Section 3.4. The same approach can be applied to the Closest Points Cell division.

Because we are looking for the attraction point, we are only interested in finding the first point along the shortest path. We have no need for the complete shortest path.

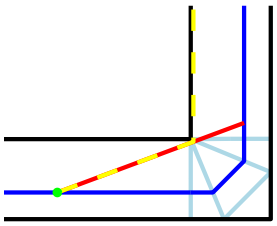


Figure 28: This figure shows the relation between the furthest visible point along the lane and the first point along the shortest path on a Closest Points Cell division. The CP Cells consists of the black obstacle borders and light blue open edges. The character is shown in green and the lane extending from it in the darker blue. The red line shows the line segment between the character and the furthest visible point on the lane. The yellow stripped line segments signify a shortest path. Depending on where the goal lies outside of the image, only the upper segment of the shortest path will rotate.

Figure 28 shows how the furthest visible point along the lane and the first point along the shortest path relate. The first point along the shortest path is the inside corner. Until that point, the shortest path and the line segment between the character's position and the furthest visible point on the lane lies exactly on the shortest path. This means that we can derive the furthest visible point on the lane with the same shortest path algorithm used in Section 3.4 as provided by Demyen and Buro (2006).

1. *Initiate a funnel starting at the character's position, with its ends at the closest points of the first cell edge of the path.*
2. *Step forward to the next cell and consider the left and right closest points of the next cell edge.*
3. *If replacing the closest point of the funnel with the same side closest point of the next cell edge narrows the funnel, use it.*
4. *If the funnel closes, take the closest point on the opposite side of the funnel as the first point on the shortest path.*
5. *Find the intersection between the lane and the ray extending from the first point on the shortest path in the opposite direction of the character's position.*

Following this procedure on our dynamic lanes will lead to the character following the shortest path once again. However, we do not want to just follow the shortest path. In crowded regions we want to follow our lanes strictly. Therefore we introduce another termination condition, a maximum distance D.

At each step, we look at the number of characters in that cell and scale it by the total area. From this we calculate the density and assign a density score, where a high density is scored highly. We multiply the length of the lane segment crossing the cell by our density score, then add that to the results of the previous cells. If this total length exceeds D, we find the point at which D is exceeded and attract to that point.

This leads to lanes being followed strictly at high densities and the shortest path being approached at low densities.

7.3 Conditions for caching attraction points

Recalculating an attraction point every timestep can be costly. Especially if we have to iterate far to obtain it. We can opt to reuse an attraction point if certain conditions hold. To ensure an attraction point is still acceptable, we need a guarantee that the attraction point is still visible.

Section 7.2 shows how to calculate attraction points and what information we compute. We utilize a funnel to test the visibility of the lane. Because of the geometric properties of a funnel, we know that from any point within the funnel is visible from any other point in the funnel. So it would be safe to reuse the attraction point if the character moved into the funnel used to calculate the attraction point.

Even if we do not want to reuse the same attraction point, we can reuse some of the information as long as a character stays within the funnel at the point when the attraction point was selected last time. By starting at the last attraction point and the funnel at the point where the maximum distance was passed, we can restart the the algorithm from there. Seeming the character is within the funnel, we know the last closest points are still visible. If we then set the funnels origin to be the character's new position, we once again have a valid funnel.

Of course, the conditions in this section do guarantee that the attraction point that is cached is still the best choice, only that it is still visible and as such usable. If we want characters to follow the shortest path, it will still be best to compete a completely new attraction point every time. As the character moves forward, new points further along the path may become visible at which point they make for better attraction points.

Also, conditions like density can changing rapidly. Applying these optimizations will simply ensure that no severe complications arise and using them sparingly will provide a welcome relief from the computational burden.

8 Experiments and results

To evaluate the effectiveness of cell edge based routes for the Indicative Route Method (IRM) (Karamouzas et al., 2009) as outlined in this thesis and its implementation, we conducted a series of experiments. The purpose of these simulations is to show the difference between a standard Explicit Corridor Map (ECM) (Geraerts, 2010) package using shortest paths (within the corridor) for IRM and an implementation that uses the cell based routes for IRM.

Additionally, for the cell edge based routes implementation, the simulations were ran at different density thresholds to illustrate the range of path planning behavior possible with this approach. This density threshold D directly defines the distance characters are allowed to look ahead to allow us to control the behavior of the characters. This density threshold should not be confused with the maximum distance from Section 7.2 as there are subtle differences.

When referring to the standard ECM package in this section, we are referring to ECM Crowd Simulation v2.2 for the simulation and Explicit Corridor Map Generator v5.2 for the ECM generation *(©R. Geraerts & W. van Toll, Utrecht University, Department of Information and Computing Sciences)*.

ECM Crowd Simulation v2.2 uses a collision avoidance method based on Moussaïd et al. (2011). In short, this means locally for each character a direction and speed is selected within a cone based on a cost function.

When referring to the cell edge based package in this section, we are referring to a package based on Density-Based ECMM Simulator for the simulation and ECM v3.43 for the ECM generation, *(©R. Geraerts & W. van Toll, Utrecht University, Department of Information and Computing Sciences)*. This package was modified to offer cell edge based routes to IRM.

It is important to note that while equipped with the same local collision avoidance method based on Moussaïd et al. (2011) as the standard ECM package, during our experimentation the local method was disabled. This was done to show that the behaviors shown are the result of the approach in this thesis and not the local method.

This is not meant to imply that the approach outlined in this thesis is a replacement for local collision avoidance methods. Rather, it should be used in conjunction with a local collision avoidance method for best results.

The experiments were performed on a PC with an Intel Xeon E5420 processor at 2.5Ghz, 4GB of RAM, with an NVIDIA Quadro FX 1700 video card. During the experiments the PC ran Windows 7 Enterprise (x64). All experiments were ran at a 600x600 resolution.

8.1 Simulations

The simulations on these scenes feature 500 characters. The characters start and goal positions are randomly generated beforehand, and stay identical across each simulation per scene.

To control the behavior of the cell edge based package, we adjust the density threshold D as described in Section 7.2. A very low density threshold means

Figure 29: This figure shows the scale used for the heatmaps in this section, from left (0) to right (max).

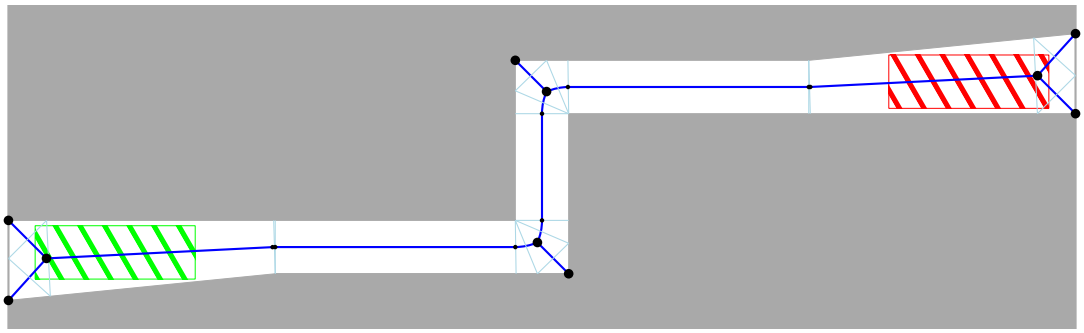


Figure 30: This shows the environment of Scene 1. The medial axis is shown in blue, with vertices (larger) and event points (smaller) marked in black. Cell edges are marked in light blue. Lastly, the start region is marked in green and the goal region is marked in red.

that characters will follow the lanes as described in Section 6 very strictly. On the other hand a very high density threshold will result in the characters looking as far ahead as possible. Without a local collision avoidance scheme, this will result in the characters following the shortest path along the route.

The scenarios in this section have a bounding box of 200 by 200, meaning a density threshold of $D >= 282.843$ will allow characters to look across the environment from any position, though a far lower D will suffice in our scenes because of the obstacles.

To obtain heatmaps, we stored the position of each character along its path. We then created a 1600x1600 grid. For each step of its path, we recorded the character's position within the grid. We then use a linear scale between 0 and the max recorded to obtain the heatmap as shown in Figure 29.

As the max value is different per image, these are given along with a table of the values for comparative purposes between images. The values are so drastically different between images that using the same scale for each image would result in many images lacking sufficient detail. Lastly, note that to exclude the space covered by obstacles from the tables, an estimate of the area of the free space in each scenario was used.

8.1.1 Scene 1 "Zigzag"

This scenario was selected to serve as the most straightforward way of displaying the difference in the resulting paths between the standard ECM package and the cell edge based package. The obstacles were built using 11 triangles. The resulting ECM structure consists of 9 vertices, 26 event points (including those at vertices, which means they can have the same position if they relate to different edges). The cell division consists of 21 cells.

Figure 31a shows the expected behavior for an extremely high density scenario. Characters very tightly follow the cell edge based paths. Table 1a confirms a relatively very low amount of unused space.

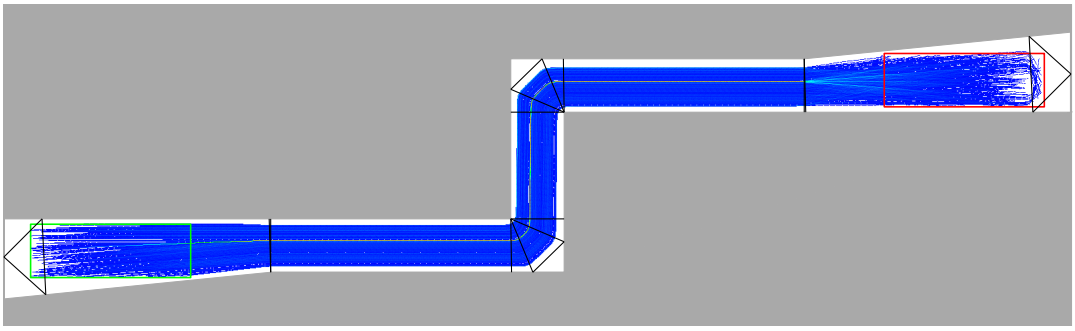
Along with the low amount of unused space, Table 1a also shows very low ranges. This confirms that in this scenario, the cell edge based package stops the crowd from packing up densely and works as intended.

The local avoidance shows a much higher average path length in Table 1d. It is important to note that this is a consequence of the local avoidance. Though following the shortest path, it often occurs that characters can not move along this shortest path. Even if characters can move towards the shortest path, often they can not move the full length of their step-size. The cell edge based package with local avoidance helps relief this but to a lesser degree still suffers an increase in path length.

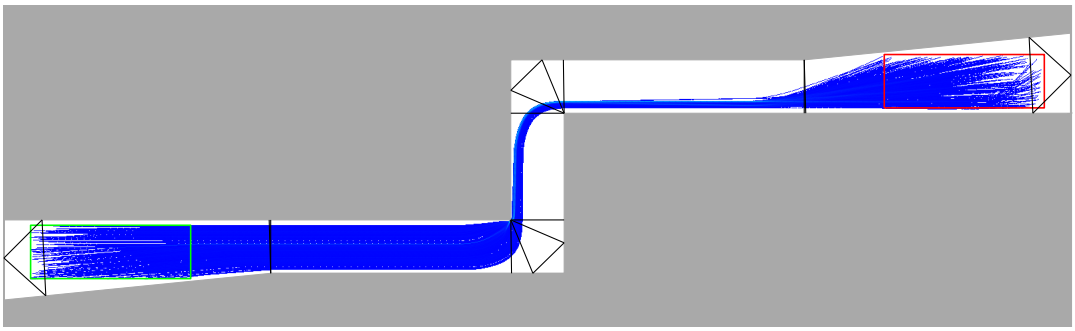
Figure 31b shows an interesting side effect of the cell edge based package. Though at every corner the characters attempt to approach the shortest path, only when the corner is within the characters' density threshold do they have this behavior. In straight sections, the characters are not allowed to look far enough ahead to start moving towards the shortest path. A sharp corner in one direction, followed by a sharp corner in the other direction results in the characters snaking around the shortest path, only catching up to it when in range of the corner.

Because characters start approaching the shortest path much later at *D=12* than at *D=283*, Table 1b shows a few anomalous spikes. Their occurrence is low and a local avoidance method smoothens these spikes out significantly.

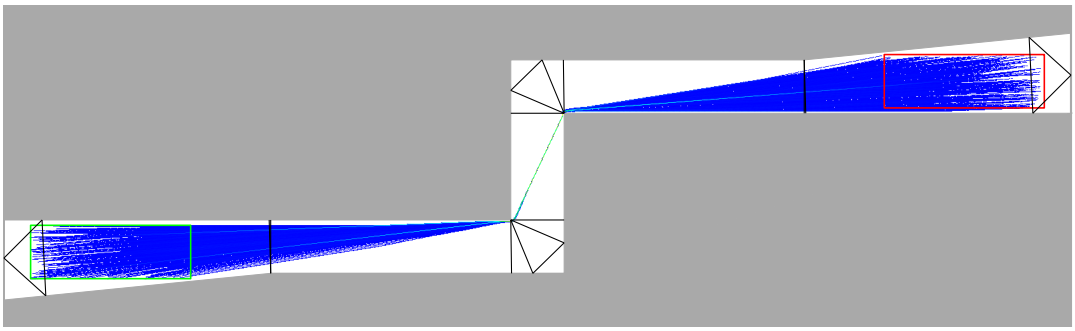
Figure 31c shows that if the density threshold D is high enough (*D=283*) characters will follow the shortest path as intended. This causes heavy clumping in the corners. Table 1c shows the corresponding high range and also a large quantity of occurrences in these high ranges to reflect this.



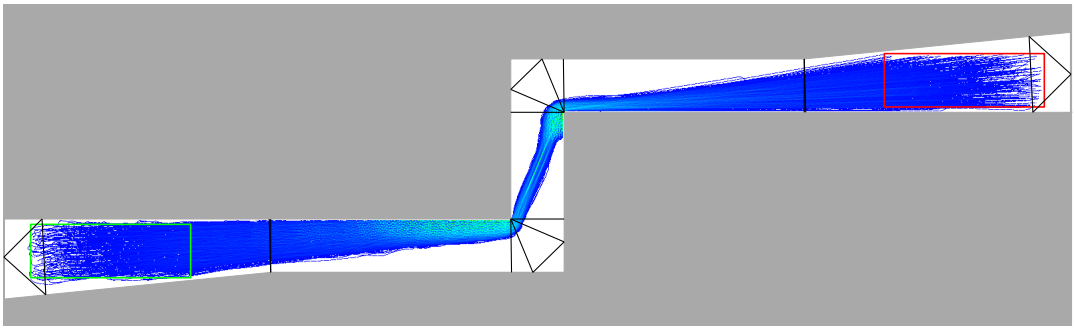
(a) Cell edge based method, high density, D=2



(b) Cell edge based method, medium density, D=12



(c) Cell edge based method, low density, D=283



(d) Standard ECM method, local avoidance enabled

Figure 31: These figures show the heatmaps resulting resulting from the simulation of 500 characters moving from a random position within the green start area and a random goal in the red goal area.

range	quantity
Unused	40694
1 - 22	104758
22 - 44	573
44 - 66	99
66 - 88	89
88 - 110	884
110 - 132	101
132 - 154	2

(a) High density, D=2. Average path length = 1332.6. Max value in the grid = 133. Average value in the grid = 4.52649.

range	quantity
Unused	70710
1 - 155	76422
155 - 310	65
310 - 465	0
465 - 620	2
620 - 775	0
775 - 930	0
930 - 1085	1

(b) Med density, D=12. Average path length = 1301. Max value in the grid = 934. Average value in the grid = 4.41916.

range	quantity
Unused	79489
1 - 125	66777
125 - 250	736
250 - 375	63
375 - 500	126
500 - 625	5
625 - 750	3
750 - 875	1

(c) Low density, D=283. Average path length = 1250.57. Max value in the grid = 750. Average value in the grid = 4.24785.

range	quantity
Unused	61928
1 - 36	81010
36 - 72	3564
72 - 108	584
108 - 144	88
144 - 180	24
180 - 216	1
216 - 252	1

(d) Local avoidance. Average path length = 1763.97. Max value in the grid = 216. Average value in the grid = 5.99175.

Table 1: These tables show on the left hand side a range and on the right side the number of occurrences in the grid within that range for scenario 1. Estimated area of the free space in the 1600x1600 grid = 147200.

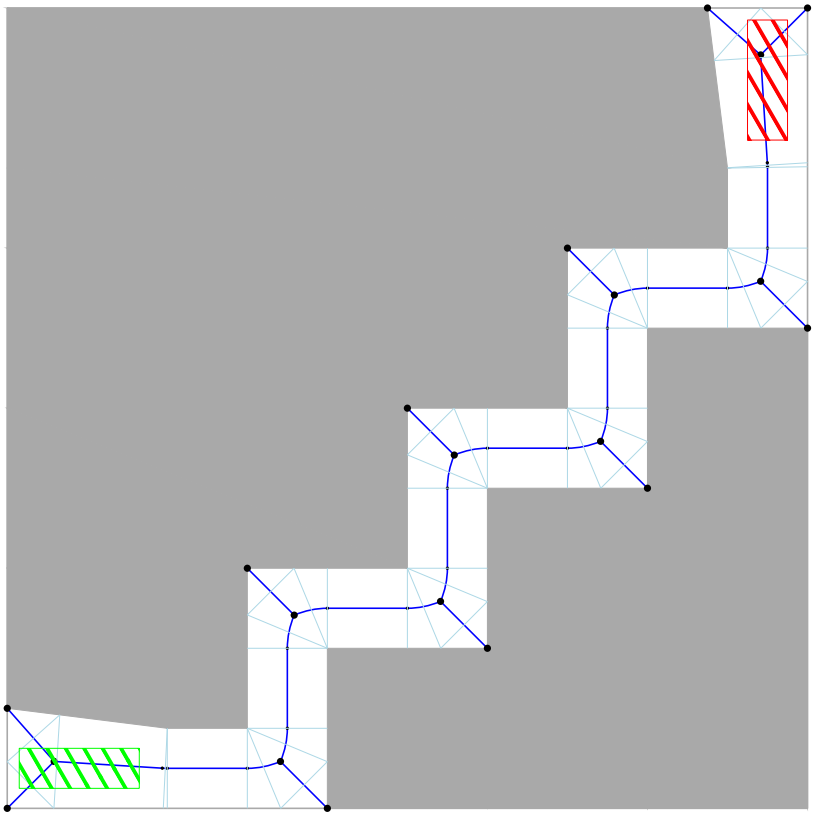


Figure 32: This shows the environment of Scene 2. The medial axis is shown in blue, with vertices (larger) and eventpoints (smaller) marked in black. Cell edges are marked in light blue. Lastly, the start region is marked in green and the goal region is marked in red.

8.1.2 Scene 2 "Ladder"

Scenario 2 was selected to highlight some of the behaviors noted in Section 8.1.1. This scene allows us to show more clearly how depending on the density threshold the shortest path will be followed more or less closely. The obstacles were built using 18 triangles. The resulting ECM structure consists of 16 vertices, 56 event points (including those at vertices, which means they can have the same position if they relate to different edges). The cell division consists of 46 cells.

Figure 31b showed that though characters approach the shortest path at every corner within their density threshold, only if the density threshold was high enough will characters catch up to the shortest path, such as in Figure 31b. In this scenario, we see this behavior reenforced. Figure 33b shows an increasing thinner range of paths after every corner as characters slowly approach the shortest path. It also shows however that because of the counteracting corners, characters struggles to reach the shortest path.

In Table 2b we once again see the high spikes resulting from the retraction to the shortest path happening at a lower distance. It is important to note

range	quantity
Unused	278720
1 - 14	203681
14 - 28	1458
28 - 42	543
42 - 56	758
56 - 70	1192
70 - 84	47
84 - 98	1

(a) High density, D=2. Average path length = 2291.65. Max value in the grid = 84. Average value in the grid = 2.35573.

range	quantity
Unused	388523
1 - 147	97801
147 - 294	72
294 - 441	0
441 - 588	0
588 - 735	1
735 - 882	2
882 - 1029	1

(b) Med density, D=18. Average path length = 2153.97. Max value in the grid = 883. Average value in the grid = 2.2142.

range	quantity
Unused	435191
1 - 135	49603
135 - 270	493
270 - 405	378
405 - 540	580
540 - 675	130
675 - 810	24
810 - 945	1

(c) Low density, D=283. Average path length = 1866.67. Max value in the grid = 750. Average value in the grid = 1.91887.

range	quantity
Unused	384837
1 - 32	94207
32 - 64	6624
64 - 96	599
96 - 128	101
128 - 160	26
160 - 192	4
192 - 224	2

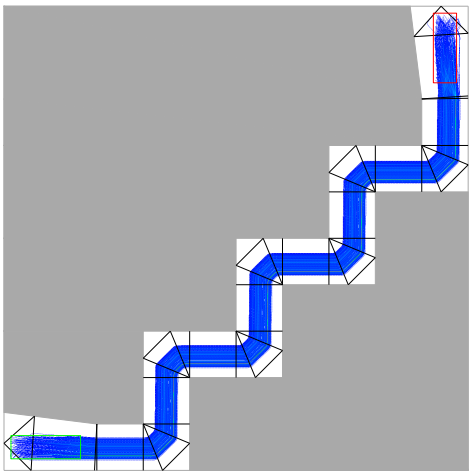
(d) Local avoidance. Average path length = 2323.95. Max value in the grid = 197. Average value in the grid = 2.38893.

Table 2: These tables show on the left hand side a range and on the right side the number of occurrences in the grid within that range for scenario 2. Estimated area of the free space in the 1600x1600 grid = 486400.

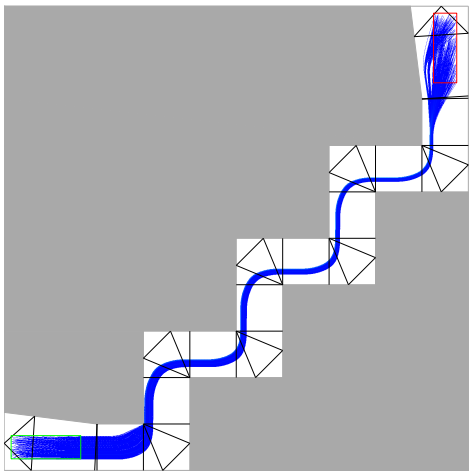
that with density thresholds being generated dynamically, we obtain a small variance in the density thresholds. Characters at the front find less character density ahead of them and as such have higher density thresholds. This causes the behaviors shown here discretely to blend. As already noted in 8.1.1, local avoidance methods also decrease these spikes considerably.

What is more important to note is that overall there is far less cells of the grid reporting high quantities, whereas Table 2c shows greater counts of highly used cells. This is consistent with the expected behavior.

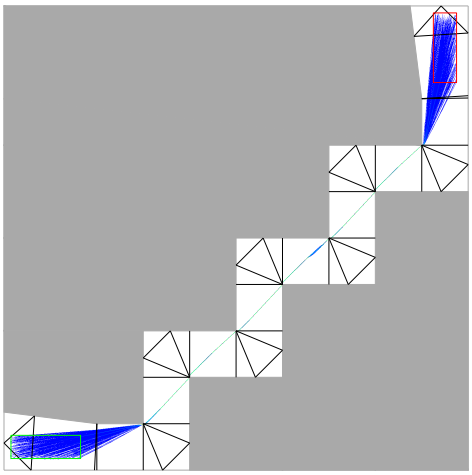
Table 2a and Table 2d show that characters following cell edge based paths use much more of the available space than the standard ECM package using local avoidance. The characters show a much a much better spread in Figure 33a, whereas in Figure 33d we see considerable clusters on the inside of corners and along the shortest path.



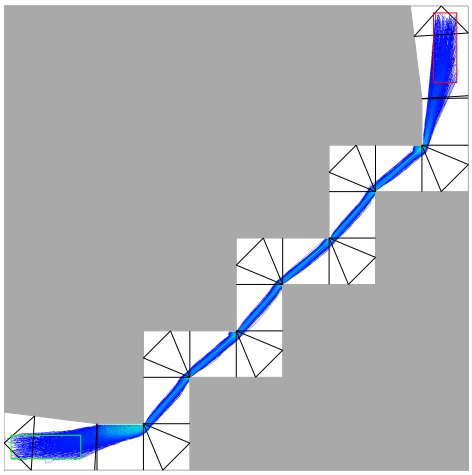
(a) Cell edge based method, high density, D=2



(b) Cell edge based method, medium density, D=18



(c) Cell edge based method, low density, D=283



(d) Standard ECM method, local avoidance enabled

Figure 33: These figures show the heatmaps resulting resulting from the simulation of 500 characters moving from a random position within the green start area and a random goal in the red goal area.

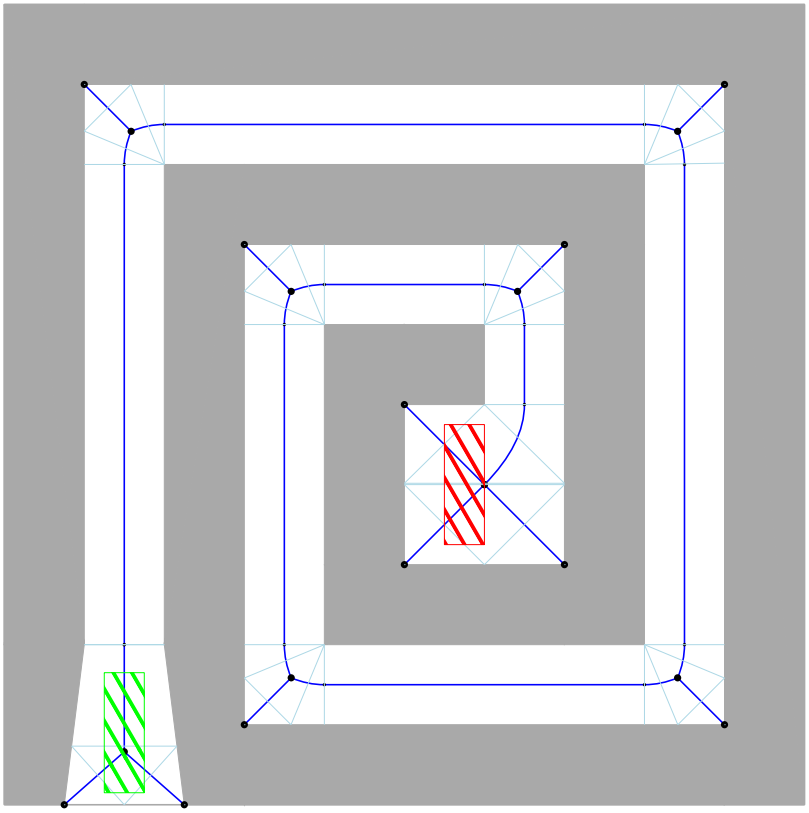


Figure 34: This shows the environment of Scene 3. The medial axis is shown in blue, with vertices (larger) and eventpoints (smaller) marked in black. Cell edges are marked in light blue. Lastly, the start region is marked in green and the goal region is marked in red.

8.1.3 Scene 3 "Spiral"

Scenario 3 was selected to show that at different density thresholds characters show different rates of convergences to the shortest path. Rather than counteracting corners as in Section 8.1.2, this scenario consists of a series of corners leading to a total rotation of over 540 degrees. The obstacles were built using 24 triangles. The resulting ECM structure consists of 20 vertices, 52 event points (including those at vertices, which means they can have the same position if they relate to different edges). The cell division consists of 42 cells.

Important to note in Figure 35d are the clusters of characters at every corner for the standard ECM package. These pile ups of characters are one of the behaviors we attempt to relief with the cell edge based approach outlined in this thesis. Table 3d shows a drastically higher average path length because of characters struggling to get around the corner and around each other, all the while still retracting to a shortest path.

The average path length for the standard ECM package is even higher than the average path length of the cell edge based package at high density as shown

range	quantity
Unused	706691
1 - 28	334066
28 - 56	2709
56 - 84	4508
84 - 112	681
112 - 140	943
140 - 168	1
168 - 196	1

(a) High density, D=2. Average path length = 4776.09. Max value in the grid = 168. Average value in the grid = 2.27519.

range	quantity
Unused	901806
1 - 148	143169
148 - 296	3174
296 - 444	1218
444 - 592	219
592 - 740	7
740 - 888	6
888 - 1036	1

(b) Med density, D=10. Average path length = 4576.91. Max value in the grid = 888. Average value in the grid = 2.18031.

range	quantity
Unused	972818
1 - 136	71456
136 - 272	1595
272 - 408	1550
408 - 544	2157
544 - 680	9
680 - 816	13
816 - 952	2

(c) Low density, D=283. Average path length = 4504.03. Max value in the grid = 817. Average value in the grid = 2.14559.

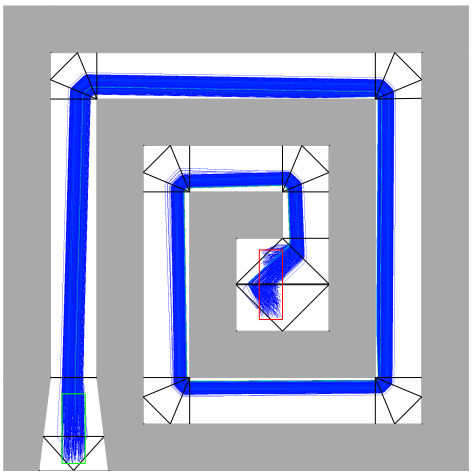
range	quantity
Unused	878784
1 - 49	160482
49 - 98	7063
98 - 147	2578
147 - 196	688
196 - 245	4
245 - 294	0
294 - 343	1

(d) Local avoidance. Average path length = 5238.25. Max value in the grid = 299. Average value in the grid = 2.49535.

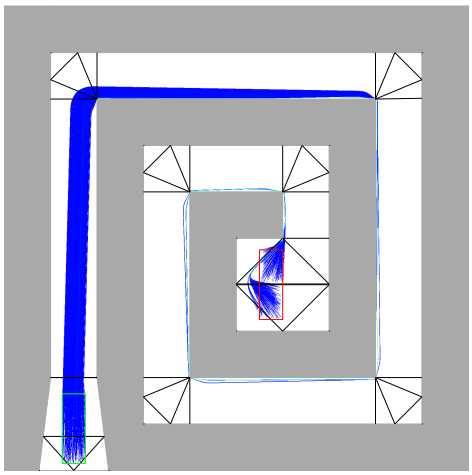
Table 3: These tables show on the left hand side a range and on the right side the number of occurrences in the grid within that range for scenario 3. Estimated area of the free space in the 1600x1600 grid = 1049600.

in Table 3a. This even though characters in Figure 35a clearly take considerably wider corners than those in Figure 35d.

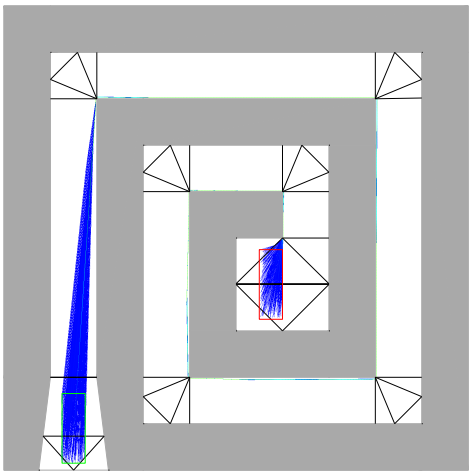
Figures 35a, 35b and 35c very clearly show the different rates of convergence to the shortest path. With a high density threshold (*D=283*) characters instantly take the shortest path and stick to it. With a low density threshold (*D=2*) characters stay spread out even until the end of the paths.



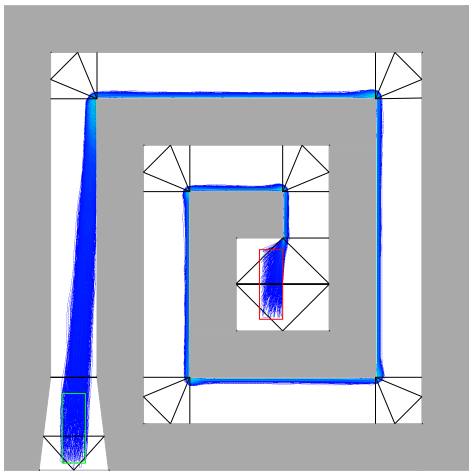
(a) Cell edge based method, high density, D=2



(b) Cell edge based method, medium density, D=10



(c) Cell edge based method, low density, D=283



(d) Standard ECM method, local avoidance enabled

Figure 35: These figures show the heatmaps resulting resulting from the simulation of 500 characters moving from a random position within the green start area and a random goal in the red goal area.

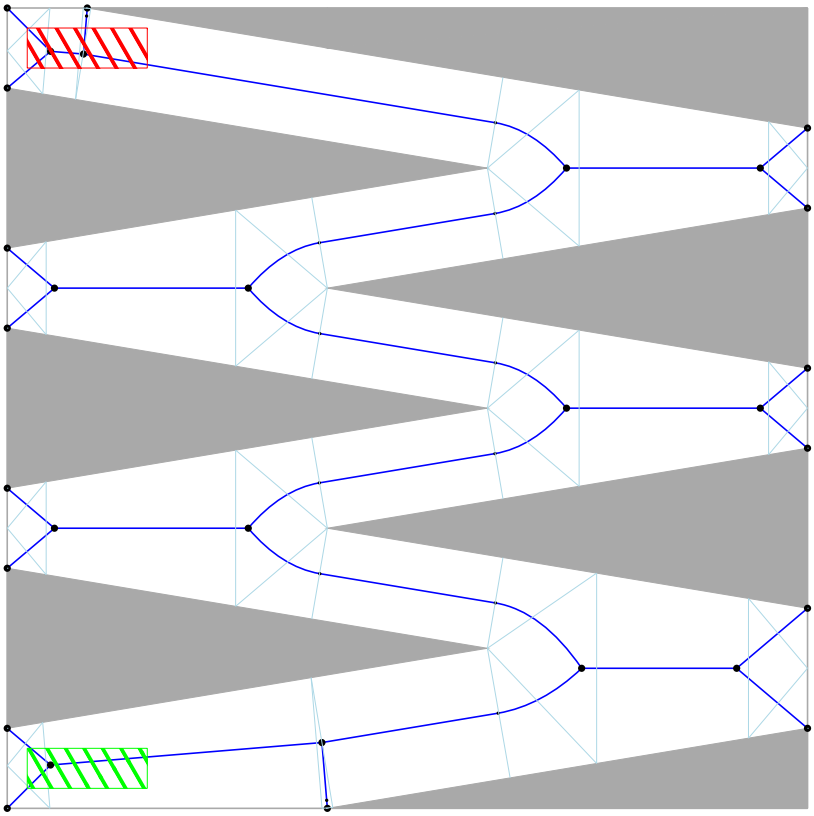


Figure 36: This shows the environment of Scene 4. The medial axis is shown in blue, with vertices marked in black. Cell edges are marked in light blue. Lastly, the start region is marked in green and the goal region is marked in red.

8.1.4 Scene 4 "Slalom"

Scenario 4 is an attempt at a difficult scenario for cell edge based package. Section 5.1 identified very sharp corners as tough to plan around. Section 5.4.3 does show us that cells can cover the exact same space as the corridor. The obstacles were built using 7 triangles. The resulting ECM structure consists of 30 vertices, 68 event points (including those at vertices, which means they can have the same position if they relate to different edges). The cell division consists of 51 cells.

Figure 37c shows characters following the shortest path, just like the cell edge based package does in all scenarios with a high density threshold. For the standard ECM package, we see in Figure 37d the same issues with clusters forming at corners. However it is interesting to note that the clusters around corners get progressively smaller.

The clusters in Figure 37d get smaller because characters have a much harder time making their way around the corner. This leads to characters forming a long thin stream which stretches out more at each corner. If the scenario

extended far enough, eventually characters would end up progressing almost in single file.

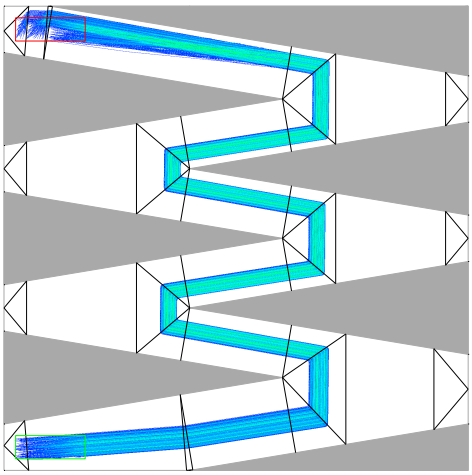
This is also why the peaks in Tables 4b, 4c and 4d can be found in the first corner.

Figure 37a shows a really great spread along the paths in the heatmap. Aside from minor local issues primarily around the goal area, Table 4a shows us very low values in the grid across the board. It also highlights the downside of using straight lanes as described in Section 6.1 instead of curved paths as described in Section 6.2.

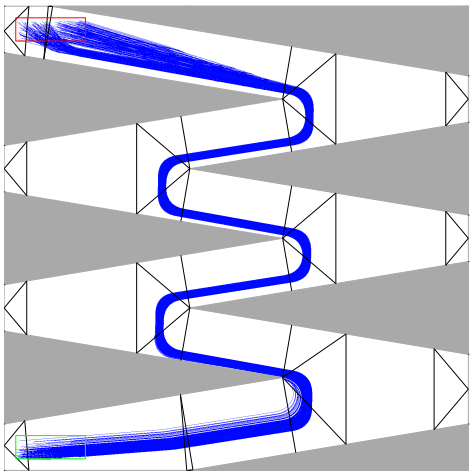
If the density threshold is set very low and straight lanes are used, instead of neatly curved paths around the bend, the resulting paths end up being very straight as well. This speaks in favor of the curved lanes as described in Section 6.2 as it results in much smoother paths even if the lanes are followed very strictly.

Figure 37b shows that with a sufficiently high density threshold, the characters swiftly catch up to the shortest path at each corner. The shortest path can be found in Figure 37c.

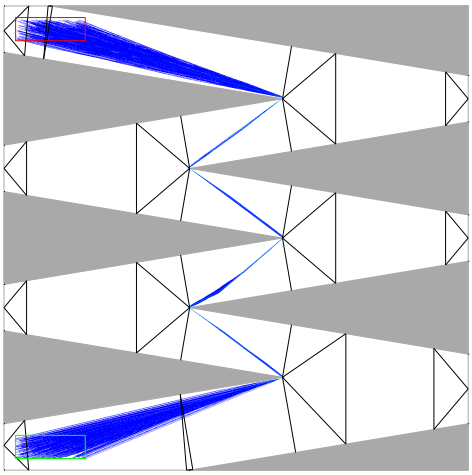
Figure 37b also shows a refusal to retract to the shortest path on the straights. This stresses the importance of selecting the appropriate density threshold and to do so based on the local density.



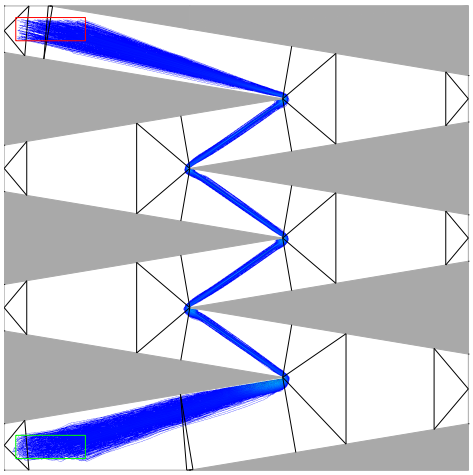
(a) Cell edge based method, high density, D=2



(b) Cell edge based method, medium density, D=16



(c) Cell edge based method, low density, D=283



(d) Standard ECM method, local avoidance enabled

Figure 37: These figures show the heatmaps resulting resulting from the simulation of 500 characters moving from a random position within the green start area and a random goal in the red goal area.

range	quantity
Unused	1194163
1 - 5	113074
5 - 10	182294
10 - 15	50414
15 - 20	1940
20 - 25	505
25 - 30	8
30 - 35	2

(a) High density, D=2. Average path length = 4278.67. Max value in the grid = 31. Average value in the grid = 1.38702.

range	quantity
Unused	1308012
1 - 122	234385
122 - 244	2
244 - 366	0
366 - 488	0
488 - 610	0
610 - 732	0
732 - 854	1

(b) Med density, D=16. Average path length = 3970.07. Max value in the grid = 735. Average value in the grid = 0.764466.

range	quantity
Unused	1424458
1 - 184	117099
184 - 368	728
368 - 552	97
552 - 736	13
736 - 920	4
920 - 1104	0
1104 - 1288	1

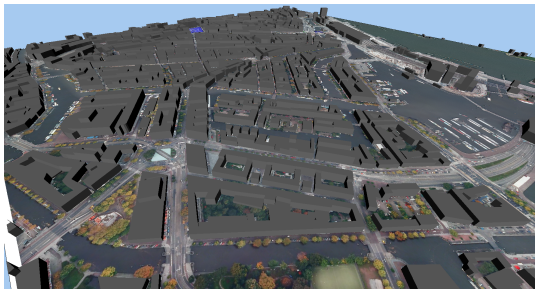
(c) Low density, D=283. Average path length = 2996.68. Max value in the grid = 1109 Average value in the grid = 0.788804.

range	quantity
Unused	1362585
1 - 89	177814
89 - 178	1561
178 - 267	392
267 - 356	34
356 - 445	9
445 - 534	4
534 - 623	1

(d) Local avoidance. Average path length = 3767.93. Max value in the grid = 534. Average value in the grid = 1.22145.

Table 4: These tables show on the left hand side a range and on the right side the number of occurrences in the grid within that range for scenario 4. Estimated area of the free space in the 1600x1600 grid = 1542400.

(a) A 2D view of the section of Amsterdam that was used in this thesis



(b) A 3D view of the section of Amsterdam that was used in this thesis

Figure 38: Scene 1 "Amsterdam"

8.2 Complex scenes

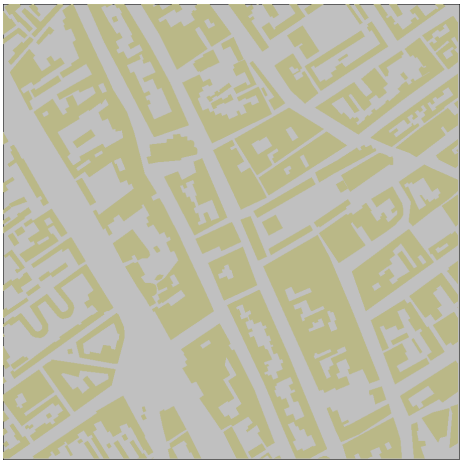
We have shown in Section 5.3 that the Closest Points Cell division can be calculated in a single pass through the Explicit Corridor Map graph. Here we will evaluate the runtime of our implementation. The scenes used in this section were provided by INCONTROL Simulation Solutions (©2014) who use such scenes for their crowd simulation and modeling software Pedestrian Dynamics (©2014).

8.2.1 Scene 1 "Amsterdam"

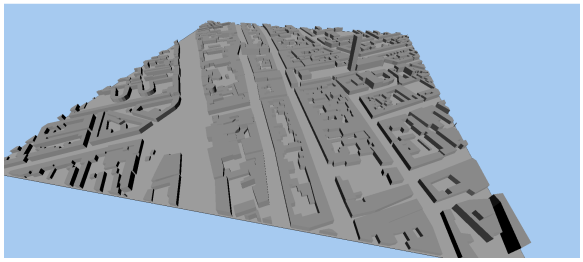
The Amsterdam scene shown in Figures 38a and 38b contains 3633 primitives, composed out of 13347 primitive vertices. Processing this using the cell edge based package results in 2348 ECM vertices, 2344 edges and 5884 Closest Points Cells. Over 100 runs, the average construction time for the ECM was 189ms. The average time to convert the ECM graph to a Closest Points Cell division was 6ms.

8.2.2 Scene 2 "Delft"

The Delft scene shown in Figures 39a and 39b contains 1802 primitives, composed out of 6628 primitive vertices. Processing this using the cell edge based package results in 1848 ECM vertices, 1841 edges and 4805 Closest Points Cells. Over 100 runs, the average construction time for the ECM was 113ms. The average time to convert the ECM graph to a Closest Points Cell division was 4ms.



(a) A cleaned up 2D view of the section of Delft that was used in this thesis



(b) A 3D view of the section of Delft that was used in this thesis

Figure 39: Scene 2 "Delft"

9 Conclusions

This section offers a summary of the findings of this thesis. We give a summary of the theoretical work, a number of experimental conclusions and the contributions of this thesis.

9.1 Summary

In this thesis, we first offered a series of observations showing that in densely crowded areas, characters do not follow the shortest path locally. Rather, people follow paths relative to their environment. In densely crowded regions, people walking on the right, stay on the right, where possible. The more space available, the more characters converge towards the shortest path.

Second, we looked at different ways to generate paths that held the observed properties in high density regions. We defined the important properties that made these alternatives suited from crowd simulation and evaluated them accordingly.

We presented a cell structure using the information available in the Explicit Corridor Map. We showed how to explicitly generate the *closest point cell structure* in $O(n)$ running time, where n is the number of event points in the ECM.

We showed two ways of generating paths from one cell edge to the next. For each alternative we offered algorithms to calculate a local offset taking a character's position within the cell.

Lastly we showed two alternatives to generate attraction points with a slightly modified Indicative Route Method.

A straightforward method using the space guaranteed obstacle free by the cell extending beyond the area of the cell using the closest point information. This method is very local and potentially will not extend far enough forward for characters to converge on the shortest path quickly.

The funnel based approach allows characters to look as far ahead as the furthest visible point along the path. This allows characters to gradually interpolate between following the shortest path and a path strictly holding the character's position relative to the relevant obstacle edges depending on the local density threshold.

Our implementation was developed during an internship at INCONTROL Simulation Solutions (http://www.incontrolsim.com ©2014). The algorithms were developed for Pedestrian Dynamics, which is a pedestrian crowd simulator by INCONTROL (http://www.pedestrian-dynamics.com ©2014).

9.2 Experimental conclusions

From the experiments in Section 8 we can draw a number of conclusions.

- The paths created from closest point cells allow a much better spread of characters. With low density thresholds, characters achieve a considerable spread even without local avoidance. Characters do not continue to all be drawn to the shortest path, but paths specifically relevant to their local position.
- The quality of the results is greatly influenced by whether characters have an appropriate density threshold. We have shown that a high density threshold in a non-crowded region can lead to excessively roundabout paths. This makes it important for characters to have an accurate sense of the density ahead of them. Storing crowd density information on cells helps with this.
- Local density dependent paths for IRM are not intended to replace local collision avoidance methods and can not achieve such. Rather, local collision avoidance methods, by changing the offset position of the character in the cell, help influence the dynamic paths as introduced in this thesis.
- In some scenarios, characters following cell edge based paths actually reached their goals considerably quicker. This can largely be attributed to characters getting drawn together into the inside of corners when all characters follow a similar shortest path. The resulting clusters take considerable effort to resolve as characters can find themselves stuck in the cluster before being allowed to progress.

9.3 Contributions

- In this thesis we have shown that the shortest path becomes progressively less relevant to a character's movement the more densely crowded the region around it becomes.
- Also in this thesis we introduce the concept of planning paths from an edge to an edge, instead of from a point to a point. This increased flexibility means paths adapt dynamically to changes.
- The Indicative Route Method can provide direction for characters through attraction points on any indicative route. IRM is commonly used in practice on shortest paths and provides smooth and realistic paths for individual characters.

  IRM has always depended on local methods to deal with other characters nearby and global methods to avoid crowded regions. With the algorithms in this thesis we provide modifications to the attraction point selection process which makes IRM itself capable of adapting in densely crowded regions. By making IRM dynamic to the local density, we have addressed crowds on a new meso level between local collision avoidance methods and global path selection methods.

- The closest point cell division provides a better, more relevant, area to store density information on. We have shown that the cells handle regions around Voronoi vertices much better than event point to event point cells.

10 Future work

We have raised questions within this thesis as to whether crowd density is something that can be handled at a global and a local level of path planning separately. We have presented a method that takes information which was previously purely used by local avoidance methods and applied it to a higher level. Harmonizing global and local path planning methods is becoming increasingly tempting research rather than looking for a single silver bullet solution that attempts to address both. In this section we will offer open problems and opportunities for further research.

10.1 Density thresholds

An opportunity for future research most directly related to this thesis are density thresholds.

In this thesis we have introduced the concept of adapting how far characters look ahead in selecting their attraction point. Characters should do so dynamically based on the local density.

We spent considerable time tweaking density scores at different densities, resulting in 5 levels of density with a respective density score. As described in Section 7.2, we use this density score as a multiplier to make path segments in dense regions more expensive.

There is room for more research here. To determine exactly what density scores are best suited for which densities, extensive testing is required. Additionally, it could be the case that a tiered density system is not the best solution and a continuous scale provides better results.

10.2 Local avoidance method

By modifying the way IRM finds attraction points so it can adjust to local density, the method now is less likely to conflict with local avoidance methods. This follows from using indicative routes that try to maintain a character's position relative to its local environment rather than always following a shortest path which will often overlap many, if not all, characters around it.

Now that IRM is influenced by local conditions, we might be able to modify local avoidance methods to better respond to IRM. We have shown that by changing the local offset, the indicative routes as presented in this thesis dynamically change with it.

Introducing for example social forces that more actively cause characters to spread out more than is needed, or now that the attraction point is less likely to cause local issues, pulling characters together more if we want group behavior, could lead to more realistic results. We have given local methods the tools to influence the level of path planning above it, now we should look how to best take advantage of it.

10.3 Different interpolation based on density

We have shown in this thesis that by adapting how far characters can look ahead to find an attraction point, we can interpolate between our strict offset path and the shortest path.

A different approach would be to always take the shortest path attraction point and the strict offset path attraction point and weigh them depending on their distance and density. This would allow characters to respond to corners earlier, only the degree to which would depend on local density. This way characters can show a measured response to changes in the path even on straight sections.

10.4 Trigger cells

When planning density dependent global paths on the Explicit Corridor Map, the density is generally computed as a summation of the area covered and total area along the event points of the ECM. Within this paper we have offered an improvement on this with our closest point cell division.

With this approach, potentially, a lot of time could be lost traversing partial edges before finding clusters of characters that make a path less attractive. Rather than calculating the density as needed, a novel approach could be to Voronoi edges a listener to each cell. When a cell reaches a critical density, that cell could report this status to its Voronoi edge.

Global planning methods can then consider the density along a certain edge, before starting to plan through it. Especially when involving high level routing solutions, this kind of bottom-up density notification could make them more efficient.

10.5 High level routing considering offsets

Though high level routing is something that has been utilized often within GPS route navigation, practical performance depends heavily on considering good nodes for precomputed paths. (Schultes & Sanders, 2007)

To utilize this technique effectively within crowd simulation, we can do better than purely selecting definitive high level routing nodes. This is because we can not depend on the sparse nature of graphs based on for example highways. Where it is easy to consider on/off ramps for highways as highlevel routing nodes, we do not have such easy to select points in many scenarios.

Our motivation for high level routing points is slightly different as well. Rather than pure efficiency which has popularized high level routing in GPS applications, interesting would be to research if we could detect viable alternative global paths based on the range of offsets a character could have within a cell. That is to say, if an character has an offset that puts it close to its left obstacle border, it should consider passing an obstacle on the left more than on the right.

To that point in Addenda A we offer a way of considering the relative position of a character to its surrounding environment. This method has issues still, as shown in Addenda A, but could potentially prove useful with further research.

To make the approach in Addenda A more viable, the approach should be used to select high level routing points. Namely those points that offer more than one global path that could be viable depending on the local offset of a character.

This would allow us to store paths and precomputed path lengths with the offset ranges at which they are viable.

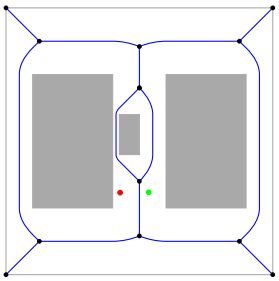


Figure 40: The figure shows a simple environment featuring 3 obstacles. Two characters are shown, in red and green. The medial axis is shown in blue.

A Using offsets to influence global routing

In this section we'll provide a method that allows us to precompute and store feasible alternative paths on an ECM, and decide which path to take while considering local density and the local position of the character within the ECM. These criteria were selected because most global path selection methods utilized with IRM focus on looking only at the medial axis length and the density surrounding its relevant edges. Even methods that consider density wait far too long to switch to alternative routes which may be available with little extra cost or might even be shorter depending on the position of the character in the ECM.

A good example of this is an obstacle that split a corridor in two segment before merging again as shown in Figure 40. Its clear that the medial axis edge between the two central vertices on the right side of the middle obstacle is considerably shorter than that on the left side.

Depending on how advanced, current methods will only send characters along the right side of the middle obstacle. At least until density or other factors make the right side path less attractive.

Even if characters alternate between the left and right hand paths around the middle obstacle as a result of clever global path planning, issues remain. Global methods generally do not consider a character's position relative to its environment. These methods generally look for the closest point on the medial axis and then plan from the vertices on either end to find a global path.

Looking at Figure 40, this could result in the red character passing the middle obstacle on the right and the green character passing the middle obstacle on the left. Its clear to see that the opposite would make more sense. The red character has a considerably shorter path around the left side of the middle obstacle.

This is also why methods planning on a graph based on the medial axis can only find the shortest path on that graph, but not the truly shortest path for a character. Within this thesis, by formalizing offsets (Section 6), we now have a way of describing a character's relative position within a segment.

Section 6 showed us how we can create alternative indicative routes for the Indicative Route Method. In Section 7 we show how to utilize these routes to

create paths that consider the density around and ahead of the character as well as its position within the segment.

When applying A-star (Section 2.1.2) on the ECM, we use the edge lengths with a density factor as a weight function. What we suggest in this section, is that rather than using the ECM edge lengths, we use an edge length which is a combination of the length of the lane for a character as described in Section 6 and the shortest path from the offset of the character to the other edge of the cell.

Where l_p is the path length of the lane, l_s is the shortest path from the point on the cell edge to any point on the other cell edge and d a density factor calculated by dividing the area covered by characters by the total area of the cell, we end up with the following edge length to be used as a weight function for A-star.

$$(l_p \times d) + (l_s \times (1 - d)) \tag{1}$$

The higher the density, the more the local position of a character is considered.

The main issue with this approach is that the shortest path from a point on one cell edge to the other cell edge often is nothing like the true shortest path for a character. It is at best an approximation because the true shortest path can not be known until a full path is computed.

B CELL format

Within this thesis we have defined a closest point based cell division of the walkable space (Section 5.2.4). As we explicitly generated this cell division, we had to devise a way of storing after generation and loading it before simulation.

We based the format on XML for the same reasons it was used with the PRIX and ECMX formats (van Toll, 2011). XML files consist of *elements* and *attributes*. Elements and attributes have the following syntax: <**element name attributename_1="value" ... attributename_n="value"**>.

B.1 Structure

ECMCells. This is the root element of a CELL file. It lists the bounding box of the scene, the number of layers and an attribute to describe scale from pixels to meters. An ECMCells element has zero or more cell elements as its children.

Cell. A cell element groups a number of edge elements, that together define the boundaries of the cell. Each cell element also has a unique id among its attributes and information about which layer and composition it is part of.

Edge. Lastly an edge element has as its attributes the positions of the closest points that define the boundary of the cell and the origin of the cell edge. Additionally each edge has an attribute that points to a cell id which links it to another Cell.

B.2 Document Type Definitions

Here we offer the Document Type Definition (DTD), i.e. formal definitions of how all CELL files were used during this thesis. structured.

```
<?xml version="1.0"?>
<!ELEMENT ecmcells (cell*)>
    <!ATTLIST ecmcells nr_cells                 CDATA #REQUIRED>
    <!ATTLIST ecmcells nr_layers                CDATA #REQUIRED>
    <!ATTLIST ecmcells xmin                     CDATA #REQUIRED>
    <!ATTLIST ecmcells ymin                     CDATA #REQUIRED>
    <!ATTLIST ecmcells xmax                     CDATA #REQUIRED>
    <!ATTLIST ecmcells ymax                     CDATA #REQUIRED>
    <!ATTLIST ecmcells meters_per_pixel         CDATA #REQUIRED>
        <!ELEMENT cell (edge+)>
            <!ATTLIST cell id           CDATA #REQUIRED>
            <!ATTLIST cell layer        CDATA #REQUIRED>
            <!ATTLIST cell comp         CDATA #REQUIRED>
                <!ELEMENT edge          EMPTY>
                    <!ATTLIST edge lx           CDATA #REQUIRED>
                    <!ATTLIST edge ly           CDATA #REQUIRED>
                    <!ATTLIST edge rx           CDATA #REQUIRED>
                    <!ATTLIST edge ry           CDATA #REQUIRED>
                    <!ATTLIST edge ox           CDATA #REQUIRED>
                    <!ATTLIST edge oy           CDATA #REQUIRED>
                    <!ATTLIST edge target       CDATA #REQUIRED>
```

References

Aurenhammer, F., & Klein, R. (1996). Voronoi diagrams. *Handbook of Computational Geometry*, 201–290.

Blum, H. (1967). A transformation for extracting new descriptors of shape. *Models for the perception of speech and visual form*.

Botea, A., Muller, M., & Schaeffer, J. (2004). Near optimal hierarchical path-finding. *Journal of Game Development*.

Chena, R., Xua, Y., Gotsmanb, C., & Liua, L. (2010). A spectral characterization of the delaunay triangulation. *Computer Aided Geometric Design*.

Chenney, S. (2004). Flow tiles. *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*.

Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., & Thrun, S. (2005). *Principles of robot motion*. MIT Press.

Cibotaru, A. (2013, may). *Alternative algorithms for computing explicit corridor maps using exact and topology-oriented paradigms* [Master's Thesis ICA–3617912].

Curtis, S., Guy, S. J., Zafar, B., & Manocha, D. (2011). Virtual tawaf: A case study in simulating the behavior of dense, heterogeneous crowds. *ICCV Workshops. IEEE*, 128–135.

de Berg, M., Cheong, O., van Kreveld, M., & Overmars, M. (2008). *Computational geometry: Algorithms and applications* (3rd ed.). Springer.

Demyen, D., & Buro, M. (2006). Efficient triangulation-based pathfinding. *21st National conference on Articial intelligence*, 942–947.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271.

Fiorini, P., & Shillert, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research, vol. 17*, 760–772.

Fortune, S. (1986). A sweepline algorithm for voronoi diagrams. *Proceedings of the second annual symposium on Computational geometry, SCG '86*, 313–322.

Geraerts, R. (2010). Planning short paths with clearance using explicit corridors. *International Conference on Robotics and Automation*, 1997–2004.

Geraerts, R., & Overmars, M. (2007a). The corridor map method: a general framework for real-time high-quality path planning. *Comp. Anim. Virtual Worlds 2007; 18:*, 107–119.

Geraerts, R., & Overmars, M. (2007b). Reachability-based analysis for probabilistic roadmap planners. *Robotics and Autonomous Systems 55 (2007)*, 824–836.

Geraerts, R., & Overmars, M. (2008). Enhancing corridor maps for real-time path planning in virtual environments. *Computer Animation and Social Agents*, 64–71.

Guibas, L., Hershberger, J., Leven, D., Sharir, M., & Tarjan, R. (1987). Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*.

Guy, S., Chhugani, J., Curtis, S., Dubey, P., Lin, M., & Manocha, D. (2010). Pledestrians: A least–effort approach to crowd simulation. *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2010)*.

Hart, P., Nilsson, N., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, 4(2)*, 100–107.

Helbing, D., Farkas, I., & Vicsek, T. (2000). Simulating dynamical features of escape panic. *Nature, vol. 407, no. 6803*, 487–490.

Helbing, D., Johansson, A., & Al-Abideen, H. Z. (2007). Dynamics of crowd disasters: An empirical study. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics), vol. 75, no. 4*.

Helbing, D., & Molnar, P. (1995). Social force model for pedestrian dynamics. *Physical Review E Vol 51*, 4282–4286.

Held, M. (1998). Voronoi diagrams and offset curves of curvilinear polygons. *Computer-Aided Design 30*, 287–300.

INCONTROL Simulation Solutions. (©2014). *Accessed: 2014-08-30.* `http://www.incontrolsim.com`.

Jaklin, N., IV, A. C., & Geraerts, R. (2013). Real–time path planning in heterogeneous environments. *Computer Animation and Virtual Worlds (CAVW)*, 285–295.

Karamouzas, I., Geraerts, R., & Overmars, M. (2009). Indicative routes for path planning and crowd simulation. *International Conference on the Foundations of Digital Games*, 113–120.

Kavraki, L., Svestka, P., Latombe, J., & Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation vol.12 no.4*.

Kneidl, A., & Borrmann, A. (2011). How do pedestrians find their way? results of an experimental study with students compared to simulation results.

LaValle, S. (2006). *Planning algorithms.*

Lerner, A., Chrysanthou, Y., Shamir, A., & Cohen-Or, D. (2009). Data driven evaluation of crowds. *MIG '09 Proceedings of the 2nd International Workshop on Motion in Games*.

Lewin, K. (1951). *Field theory in social science.* Harper & Brothers, New York.

Moussaïd, M., Helbing, D., & Theraulaza, G. (2011). How simple rules determine pedestrian behavior and crowd disasters.

Musse, S., & Thalmann, D. (1997). A model of human crowd behavior: Group inter-relationship and collision detection analysis. *Proc. Workshop of Computer Animation and Simulation of Eurographics97*, 39–51.

ÓDúnlaing, C., Sharir, M., & Yap, C. (1983). Retraction: A new approach to motion planning. *ACM Symposium on Theory of Computing*, 207–220.

Pedestrian Dynamics. (©2014). *Accessed: 2014-08-30.* `http://www.pedestrian-dynamics.com`.

Preparata, F. (1977). The medial axis of a simple polygon. *Mathematical Foundations of Computer Science, vol. 53*, 443–450.

Prorail. (2009). Monitoring spoorgebruik.

Reynolds, C. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics, 21(4) (SIGGRAPH 87 Conference Proceedings)*, 25–34.

Reynolds, C. (1999). Steering behaviours for autonomous characters. *Game Developers Conference, Miller Freeman Game Group*.

Schultes, D., & Sanders, P. (2007). Dynamic highway-node routing. In *In proc. 6th workshop on experimental and efficient algorithms. lncs* (pp. 66–79). Springer.

Still, G. K. (2000). Crowd dynamics. *Ph.D. dissertation, University of Warwick*.

Sturtevant, N. (2012). Benchmarks for grid-based pathnding. *Transactions on Computational Intelligence and AI in Games*.

Treuille, A., Cooper, S., & Popovic, Z. (2006). Continuum crowds. *ACM Trans. Graph., vol. 25*, 1160–1168.

van den Akker, M., Geraerts, R., Hoogeveen, H., & Prins, C. (2010). Path planning for groups using column generation. *The Third International Conference on Motion in Games, Springer Lecture Notes in Computer Science (LNCS) 6459*, 94–105.

van den Berg, J., Lin, M., & Manocha, D. (2008). Reciprocal velocity obstacles for real-time multi-agent navigation. *IEEE Int. Conf. on Robotics and Automation*, 19281935.

van den Berg, J., Patil, S., Sewall, J., Manocha, D., & Lin, M. C. (2008). Interactive navigation of multiple agents in crowded environments. *Proceedings of the 2008 symposium on Interactive 3D graphics and games, ser. I3D '08*, 139–147.

van Goethem, A. (2012). *A stream algorithm for crowd simulation to improve crowd coordination at all densities.* (Master's thesis ICA–0215872). Utrecht University.

van Toll, W. (2011). *A navigation mesh for efficient density-based crowd simulation in multi-layered environments.* (Master's thesis ICA–3117456). Utrecht University.

Yu, W., & Johansson, A. (2007). Modeling crowd turbulence by many-particle simulations.

Zhou, S., Chen, D., Cai, W., Luo, L., Yoke, M., Low, H., ... Hamilton, B. (2010). Crowd modeling and simulation technologies.