Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Miloš P r á g r

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Scalable Agent Navigation in Crowd Simulation

Guidelines:

Potential maps are used for global navigation of agents in crowd simulations.
The advantage of using potential maps is that it scales well with a number of navigated agents.
On the other hand the scaling with both size of the grid and number of exits is limiting.
A regular grid is especially inefficient in large open areas.

1. Analyze the existing potential navigation in AgentCrowd framework.
2. Consider a triangulation technique to overcome the limitations of the grid approach.
3. Design and implement a solution into the AgentCrowd framework.
4. Evaluate the proposed solution in comparison to the original one.

Bibliography/Sources:

[1] Patil, S.; van den Berg, J.; Curtis, S.; Lin, M.C.; Manocha, D., "Directing Crowd Simulations Using Navigation Fields," in Visualization and Computer Graphics, IEEE Transactions on , vol.17, no.2, pp.244-254, Feb. 2011doi: 10.1109/TVCG.2010.33
[2] Xiaogang Jin, Jiayi Xu, Charlie C.L. Wang, Shengsheng Huang, Jun Zhang, "Interactive Control of Large-Crowd Navigation in Virtual Environments Using Vector Fields", IEEE Computer Graphics and Applications, vol.28, no. 6, pp. 37-46, November/December 2008, doi:10.1109/MCG.2008.117
[3] Russell, S., & Norvig, P. (2010). Artificial Intelligence: A Modern Approach (p. 1132). doi:10.1017/S0269888900007724

Bachelor Project Supervisor: doc. Ing. Jiří Vokřínek, Ph.D.

Valid until: the end of the summer semester of academic year 2016/2017

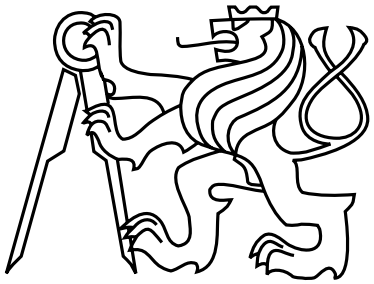
L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, December 18, 2015

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Bachelor's Thesis

Scalable Agent Navigation in Crowd Simulation

Miloš Prágr

Supervisor: doc. Ing Jiří Vokřínek, Ph.D.

Study Programme: Otevřená Informatika, Bakalářský

Field of Study: Informatika a počítačové vědy

May 26, 2016

Aknowledgements

Foremost, I would like to thank my thesis supervisor doc. Ing. Jiří Vokřínek, Ph.D.. I also want to thank other members of the AgentCrowd project, which this thesis is a part of, mainly Ing. Martin Schaefer, Ing. Michal Štěpanovský, Ph.D, and Bc. Martin Fadrhons. Finally, I would like to acknowledge all the members of Faculty of Electrical Engineering academia.

Declaration

I declare that I elaborated this thesis on my own and that I mentioned all the information sources and literature that have been used in accordance with the Guideline for adhering to ethical principles in the course of elaborating an academic final thesis.

In Prague on May 23, 2016 .....

Abstract

AgentCrowd crowd simulation framework uses a potential field agent navigation. Defining characteristics of the potential field navigation such as scaling with size of the simulation world and with number of agent flows can be limiting. We consider several alternative approaches based on various meshing and triangulation algorithms. We implement selected algorithms and compare them to the original approach. Finally, we draw a conclusion and select one specific approach for the AgentCrowd framework.

Abstrakt

AgentCrowd, framework pro simulaci davů, používá potenciálová pole pro navigaci agentů. Hlavní vlastnosti navigace pomocí potenciálových polí, jako například škálování s velikostí simulačního světa nebo s počtem proudů agentů, mohou limitovat výkon simulátoru. Tato práce uvažuje několik alternativních přístupů založených na meshovacích a triangulačních aloritmech. Vybrané algoritmy jsou implementovány a porovnány s orginálním přístupem. V závěru je jeden nový přistup zvolen jako vhodný pro AgentCrowd.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this work we[1] present several navigation approaches for the AgentCrowd crowd simulation framework. In the following section we present general overview of crowd simulation approaches, we describe the AgentCrowd framework, and finally we present a general structure of this thesis.

1.1 Crowd Simulation

Crowd simulations model movement and collective behaviour of large amount of entities in some organized or spontaneous group. There are numerous usages for a crowd simulation, ranging cinematography, evacuation and urban planning, or security simulations. In general, the various goals of a simulation may require adoption of specific models, or at least disqualify some approaches.

There are numerous approaches to crowd simulation. As was stated previously, choice of an approach is often related to the specific goals of the simulation. It should be noted that some modular crowd simulation frameworks such as MENGE[2] are modular and allow combination of multiple approaches on various position in its modularity.

In general, the crowd simulation approaches can be divided into two major groups:

- Macroscopic Approaches based on simulating the crowd as a whole, generally suitable for simulation of large crowds
- Microscopic Approaches based on simulating the singular agents with higher level of detail, generally suitable for simulation of small crowds

The macroscopic approaches are generally based on density distributions. Example of macroscopic simulation is the flow based approach, as in [Zhou et al., 2010].

In the following text we will be examining the microscopic approaches. For most such approaches, it is possible to construct abstract levels of crowd simulation describing its various sub-problems. The abstraction used by highly modular crowd simulator MENGE, as described in [Curtis et al., 2014], is as follows:

1. Goal Selection - selection of goal for each of the simulated entities

[1] this and any following usage of 'we' refers to its general academic use, unless the text states otherwise
[2] http://gamma.cs.unc.edu/Menge/

2. Plan Computation - computation of plan based on static data, e.g., map of obstacles
3. Plan Adaptation - adaptation of plan to often dynamic problems of local scale
4. Motion Synthesis - computation of physical characteristics of movement for visual applications
5. Environmental Queries - computation of various sub-problems in environment-agent relationship, such as the visibility or lighting

The following section lists several significant approaches corresponding to levels of plan computation and plan adaptation, as these levels are the focus of the AgentCrowd framework.

1.1.1 Plan Computation

Plan computation methods create a navigation plan for an agent based on mostly static data representing the environment, such as obstacles, surface type, speed limits, etc.. The most notable approaches are:

- Potential Fields, where each goal is assigned a grid-like field. Each voxel in the field is assigned a potential values, with agents travelling to the local goals represented by global extremes in the field.
- Navigational Meshes, where agents move along polygon structures. Typical example of navigational meshes is AnavMesh by [Oliva and Pelechano, 2012] or various Delaunay triangulations.
- Voronoi Graphs, where agents move between vertices. This approach is closely connected to some of the mesh approaches.

1.1.2 Plan Adaptation

Plan Adaptation methods solve dynamic problems that surface during the simulation. Typical example of such a problem is local collision avoidance when interacting with other agents, dynamic doors and other portals, or spreading fire and panic.

There are various approaches to this problem. Some are modification of the plan computation methods and some are standalone and are complementary to the plan computation. Some of the more significant approaches are:

- Social Forces, list of various stimuli on the agent that are predictable, e.g. attractive and repulsive forces to other nearby agents. The social forces model was proposed by [Helbing and Molnar, 1995].
- Cellular Automata, which describe the behaviour of an agent by multilevel cellular automaton. The cellular automata use in crowd simulation was proposed by [Schadschneider, 2002].
- Dynamic Potential Fields, that combine plan computation and adaptation. An example of this approach is [Treuille and Cooper, 2006], whose approach combines velocity and distance terms in order to simulate large, homogeneous crowds.
- Optimal Collision Avoidance Systems, providing fully collision free smooth movements, such as ORCA[3]

[3] http://gamma.cs.unc.edu/ORCA/

1.2 AgentCrowd Framework

AgentCrowd framework is an agent-based crowd simulation framework. It is developed by Artificial Intelligence Center, Czech Technical University in Prague, Prague, Czech Republic. The framework is based on java-based open-source multi-agent toolkit Alite[4].

The framework is centred around simulating agents on a grid-like structure with the emphasis on plan computation and plan adaptation levels (see Section 1.1 for level specification).

At the moment[5] the framework is under development and undergoing modularization. Note that the general description provided in this work follows the framework's state before the current development stage.

According to [Hrstka et al., 2013] the framework is divided by into three abstract layers. The first two layers, that combine into the physical layer, are

- Static layer: static objects in the simulation world, such as obstacles
- Dynamic layer: dynamic objects in the simulation world, such as agents

The remaining layer is

- Abstract layer: layer storing information based on the physical layers, such as potential maps

In general, agents use information from the abstract layer to compute appropriate behaviour.

1.2.1 Physical Layers

Physical layers describe the static and dynamic objects that are present in the simulation. Excluding various input information used in preparation of the abstract layer, there are two major components to the physical layer: the map and the agents.

1.2.1.1 Map

The map is grid-like rectangular field of cells. The cell size is fully adjustable, although originally the intended size was 2x2 feet. In the simulation individual cell's set of neighbours is defined as all the eight neighbours including diagonals, or smaller equivalent for cells located on the border.

Each cell carries information about its position, occupation state, and surface type. The occupation state affects whether an agent can move to the respective cell. Cell is either empty or occupied. There are two reasons for a cell to occupied: either there is an agent present in the cell, or the cell is an obstacle. The surface type affects the speed of agent's movement.

[4] http://jones.felk.cvut.cz/redmine/projects/alite
[5] Spring 2016

1.2.1.2 Agent

In the current implementation, each agent is an independent entity. All agents that are actively in the simulation are located on exactly one cell in the grid.

As there is no high-level goal selection in the current version of the framework, agent's goals are derived from a flow they are assigned to: a pair of entry and exit zone. The agents travel between from their respective entry zone to their respective exit zone. The agents use the abstract layer to navigate in the environment and reach their respective exit zone. As soon as the exit zone is reached, the agent is removed from the simulation.

1.2.2 Abstract Layer

The abstract layer stores various information about the physical layer and helps the agents to navigate in the environment. In the current version of the framework two approaches are combined. The plan calculation (i.e., global navigation) is provided by potential maps and the plan adaptation (i.e., local navigation) is provided by social force model. The agents navigate by combining vectors recovered from these two models.

1.2.2.1 Potential Maps

Potential maps are the plan computation method of the AgentCrowd framework. The maps provide agents with a shortest-path-to-exitzone navigation solution.

For each exit zone a independent potential map is created. The map is a grid-like structure where each position is assigned an appropriate potential values, ranging from $-\infty$ (exit zones) to ∞ (obstacles). The potential represents pseudo-distance to the exit zone. It should be noted that as the agents, normalize the recovered vectors, the actual size of the vector does not matter. The only relevant vector characteristic is its direction.

The potential maps are computed using a fast-marching algorithm starting at their respective exit zone. For the specification of the algorithm see [Hrstka et al., 2013]. Generally, each cell of a potential field correspond to a cell of the map, but this not necessarily a rule. The map and the potential fields may have different scale.

1.2.2.2 Social Forces

Social forces are the plan adaptation method of the AgentCrowd frameworks. The system allows agents to navigate in the local space and avoid collisions with other agents.

The theoretical framework is based on approach presented in [Helbing and Molnar, 1995]. The assumption is that pedestrian reaction is usually automatic and determined by their experience. The social force helps the agents to keep together (in case of members of one team or family) or separated.

It should be noted that the social force approach provide realist pedestrian behaviour. The agents do not have any kind of collective intelligence that would provide perfect collision avoidance and the agents do occasionally collide and suffer from bottlenecks.

1.3 Structure of the Thesis

Out primary goal in scope of this work is provide new navigation solution to the AgentCrowd framework. The concrete changes are to be made in the context of plan computation algorithm. The potential field approach that is currently used in framework has several limitation. From results of a preliminary project it was concluded that the potential map approach has severe limitation in terms of scaling with simulation size and number of flows. However, the approach has distinct advantage in handling complex spaces and large numbers of agents. See Section 4 for numerical data measured in the scope of this work.

In the following sections we will investigate several meshing and triangulation methods and compare results of their respective implementations to the potential map approach. Based on the method evaluation a conclusion will be drawn, either recommend a specific method for general use or assigning various methods to specific types of simulations.

Chapter 2

Navigation Meshes

Navigation meshes are one of the commonly used navigational techniques. A navigational mesh allows division of a 2D or 3D space into multiple subspaces. Generally, it could be claimed that the created subspaces are convex polygons. [1] The idea of navigational mesh is attributed to [Snook, 2000], who proposed use of polygonal mesh . Navigational mesh usually consists of convex polygon representation of aforementioned subspace, and by some representation of connection to neighbouring spaces. It should be noted that while most of the described techniques are transferable into 3D space, this work is mostly concerned with 2D navigation, as the AgentCrowd frameworks simulates movement of agents on a 2D plane.

This section is concerned with the Delaunay triangulation, which is such a triangulation of a point set, that maximizes the minimum angle of any of the triangles. More specifically, it is aim of this section to describe relevant algorithms for creation of Constrained Delaunay Triangulation, which solve the problem of Delaunay triangulation of a polygon that may be non-convex. Following algorithms are examined:

- Flip algorithm (& ear clipping, its supplementary algorithm)
- Divide and Conquer algorithms
- Sweep Line algorithms
- Incremental algorithms

The flip algorithm is used for creation of Delaunay triangulation of planar point sets and can be utilized for constrained Delaunay triangulation. Ear clipping is one of the simplest methods of polygon triangulation and can be used to create first iteration for the flip algorithm. Divide and conquer algorithm uses simple space division methods that create easily-triangulated areas and process them in parallel manner. Sweep line algorithms compute the triangulation using a forward moving division line. Incremental algorithms' basic principle is inserting unprocessed points into the triangulation.

There are various algorithms that create some navigational space. The range of algorithms varies greatly with degree of optimality, type of the resulting polygonal objects, ability to cope with space complexity, or complexity of the algorithm itself. Examples of meshing algorithms range from simple algorithm proposed by [Hertel and Mehlhorn, 1985], which is

[1] The reason for enforced convexity is that non-convex polygons would allow situations where agents travelling on a straight line between two points located in one polygon could leave the polygon. See Figure 2.1 for illustration.

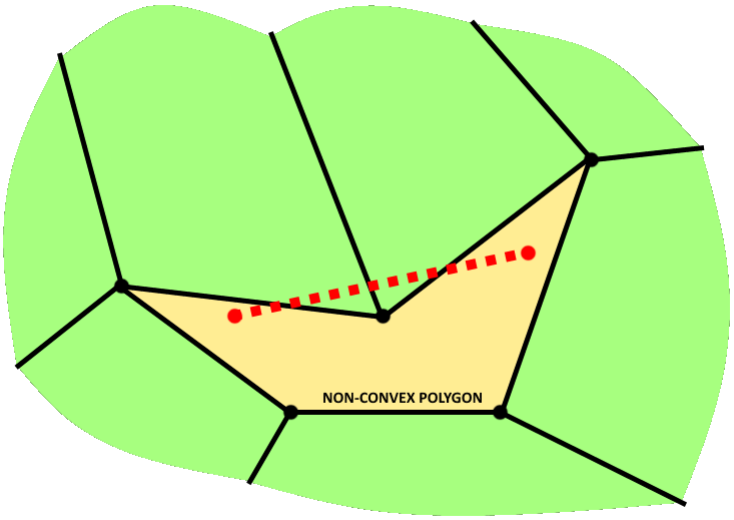


Figure 2.1: Limiting Feature of Non-convex Polygons
Agents travelling on a straight line between two points located in one polygon leaving the polygon.

limited to space with no holes, algorithms creating generalized convex polygon on space with holes, such as the ANavMG by [Oliva and Pelechano, 2012], to various triangulations such constrained Delaunay triangulation, that are near-optimal. Sections concerned with the ANavMG algorithm and constrained Delaunay triangulations are sections 2.1 and 2.3 respectively.

2.1 Automatic Navigation Mesh Generator

The Automatic Navigation Mesh Generator is a sub-optimal mesh creating algorithm, proposed by [Oliva and Pelechano, 2012]. The algorithm identifies convex vertices, for any vertex defines its interest area described by intersection of half-spaces defined by lines extending the edges neighbouring the vertex and the space being divided. See Figure 2.2 for further illustration. The space is then divided by creating division line between the respective vertex and the closest of following objects in the interest space - other vertex, point on an edge of the outer polygon or inner polygon, or a vertex on end of an edge created in already executed division of non-convex space. In the case of created edge, check is run to explore the possibility to create a convex union of polygons by removing the older division edge. The main advantage of this algorithm is that the number of created sub-space polygons is $\mathcal{O}(r)$, where r denotes the number of non-convex vertices.

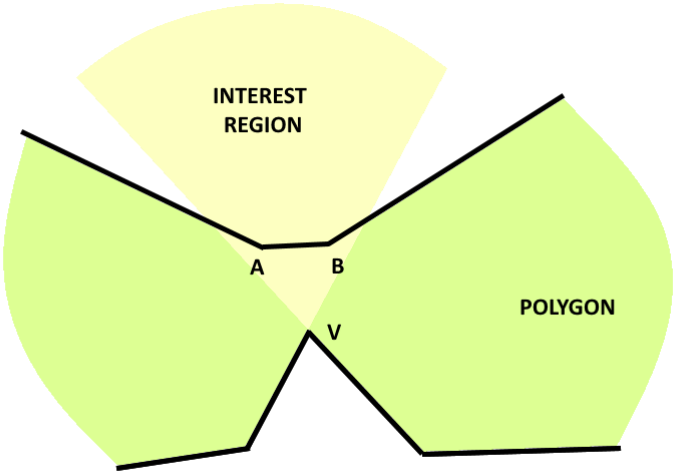


Figure 2.2: Interest Region in AnavMesh algorith
Interest region originating from vertex V. Vertices A and B and segment AB are located inside the interest region. Remaining segments containing A and B are partially partially inside.

2.2 Delaunay Triangulation

Delaunay triangulation is such a triangulation of a point set, that maximizes the minimum angle of any of the triangles. This is connected to the fact that in a Delaunay triangulation, vertices of any triangle lie outside of circumcircle of any other triangle, with the obvious exception of the vertices shared by two neighbouring triangles, that lie precisely on both the circumcircles. Also, a true delaunay triangulation uses only the points that were part of the input. The angle-minimizing property is truly intriguing from the point of view of mesh creation, as it seems that triangles with a high difference between angles are inconvenient - while long and thin triangles may be beneficial when representing a space that is also long and thin, e.g. a narrow street, presence of such triangles in most open areas would case arbitrary path, possibly with a 'zig-zag' pattern.

Definition 2.2.1. Delaunay Triangulation for a Planar Point Set (as presented by [Cheng et al., 2012])
In the context of a finite point set S, a triangle is Delaunay, if its vertices are in S and its open circumdisk is empty: i.e., contains no point in S. Note that any number of points in S can lie on a Delaunay triangle's circumcircle. An edge is Delaunay if its vertices are in S and it has at least one empty open circumdisk. A Delaunay triangulation of S , denoted Del S , is a triangulation of S in which every triangle is Delaunay.

For a Delaunay triangulation of planar point set, there is an alternative description of Delaunay property: the local Delaunay property of triangles. The description is a result of the Delaunay lemma, proved by R. Delaunay.

Definition 2.2.2. Locally Delaunay Triangle Let e be an edge in a triangulation T in the plane. If e is an edge of fewer than two triangles in T, then e is said to be locally Delaunay. If

e is an edge of exactly two triangles τ_1 and τ_2 in T, then e is said to be locally Delaunay if it has an open circumdisk containing no vertex of τ_1 nor τ_2. Equivalently, the open circumdisk of τ_1 contains no vertex of τ_2. Equivalently, the open circumdisk of τ_2 contains no vertex of τ_1.

However, while the properties of Delaunay triangulation are intriguing, there are limitations to this approach. A true Delaunay triangulation can be created only for a planar point set. If an application in meshes used to navigate in a polygonal environment is to be considered, these limitations become apparent. The polygons representing such an environment are often non-convex a may contain holes. In such a case, a planar triangulation is not possible, as there is no guarantee that the triangulation would be constraint by these limitations. There are several possible ways how to solve this problem:

- Consider such a subdivision of polygon that creates only convex polygonswith no holes, such as the ANavMG described in Section 2.1, and apply the Delaunay triangulation to these polygons independently.
- Create a constraint Delaunay triangulation (CDT). A CDT is such a triangulation, which requires some edges to be present in the triangulation. This approach requires us to abandon the idea of true Delaunay triangulation of a planar point set, but still provides us with some guarantees towards the optimality of the solution.

2.3 Constrained Delaunay Triangulations

Constrained Delaunay Triangulation is a intriguing solution for the problem Delaunay triangulation of a polygon that may be non-convex. There appears to be some degree of confusion in the terminology. While sometimes the term of Constrained Delaunay Triangulation is used for such a triangulation that solves the problem of Delaunay triangulation with enforced edges without introducing any additional points. [Kallmann et al., 2004] and definitions used in the Triangle mesh triangulator[2], however, consider Steiner triangulations to be a subtype of constrinaed Delaunay triangulations. Steiner triangulation is such a constrained Delaunay triangulation, the solves the Delaunay triangulation problem while introducing new addition vertices. Triangle further divides the triangulation containing Steiner points into conforming Delaunay triangulation, that are true Delaunay triangulations, and constrained conforming Delaunay triangulation, which are not, and claims that in general, as the triangles in constrained conforming triangulations are not trully Delaunay (but still locally Delaunay with respect to the edges of the polygon), fewer vertices are needed to reach a required quality of CDT. In this work, I will consider Steiner triangulations to be a subtype of constrained Delaunay triangulations. Also, the term of constrained Delaunay triangulation will be abbreviated as CDT. In general, this is the case in literature concerned with triangulations, although sometimes this is used only for conforming Delaunay triangulation.

In general, there are three approaches how to create a constraint Delaunay triangulation, although some algorithm, such as the flip algorithm, may not be part of any such group.

- **Divide and Conquer Algorithms**
  Divide and Conquer algorithms use various simple space division methods that create areas that are easier to triangulate. The main challenge in this type of algorithms is in

[2] https://www.cs.cmu.edu/ quake/triangle.html

merging the sub-areas together. The original CDT algorithm proposed by [Chew, 1989] is of this kind, using strips containing only one vertex as the smallest areas. These algorithms do not always contain Steiner points.

- **Sweep Line Algorithms**
  Sweep Line algorithms use a sweep line to divide the polygon into area that was already triangulated and the remaining area and gradually add new triangles. While this is not the original approach, in the paper proposing the first CDT algorithm [Chew, 1989] actually recognized the feasibility of this approach, more precisely the technique of Voronoi diagram construction used by [Fortune, 1987], but at the time he was unsure how this would be done.
- **Incremental Algorithms**
  Incremental algorithms are based on creating triangles by inserting edges or vertices. Nowadays, this is the most popular method, mostly because this is also the easiest method to implement.

It should be noted that while this categorization of CDT algorithms appears to be used in most relevant literature, it seems that it was originally created as Voronoi diagram algorithm categorization. Early example of this categorization is [Fortune, 1987], who notes the existence of divide and conquer and incremental algorithms and presents a new, sweep line based approach. This is not surprising, as while the Delaunay triangulations and Voronoi diagram are strictly speaking different structures, there is a high degree of connection between the two, because Delaunay triangulation is a dual graph to the Voronoi diagram. It is possible to create one from the other or modify some of the algorithms to produce the other as output. Although this work considers the [Chew, 1989] approach to be the oldest Delaunay triangulation algorithm, as it is the oldest algorithm specialized on creation of Delaunay triangulations, Fortune did acknowledge the possibility of modifying his algorithm for Delaunay triangulation.

2.3.1 Flip Algorithm

The flip algorithm is an algorithm for creation of Delaunay triangulation of planar point sets that can be utilized for constrained Delaunay triangulation. The flip algorithm is connected to the definition of the locally Delaunay triangles. Generally, the algorithm functions in a following way: consider two neighbouring triangles (i.e., the triangles sharing an edge) in a triangulation. Together, these triangles form a quadrilateral. If the quadrilateral is convex, it is possible to flip the triangles, if it is beneficial. In this context, flipping triangles in a quadrilateral means to create two new triangles that share a different edge - as the original triangles have to share one of the quadrilateral's diagonals, the flipped triangles simply share the other one. The condition for flipping is whether the edge is not locally Delaunay. If this is so, the other diagonal is necessarily locally Delaunay. See Figure 2.3 for illustration. Also, it should be noted that if it is not possible to flip the triangles (i.e., the quadrilateral is non-convex), edge is always Delaunay. While a step in the algorithm can make the outer edges of the quadrilateral non-Delaunay, the algorithm is guaranteed to terminate after $\mathcal{O}(n^2)$ steps. While full proof as presented by Gärtner (2012) is not provided, it is based on the usage of a lifting map.

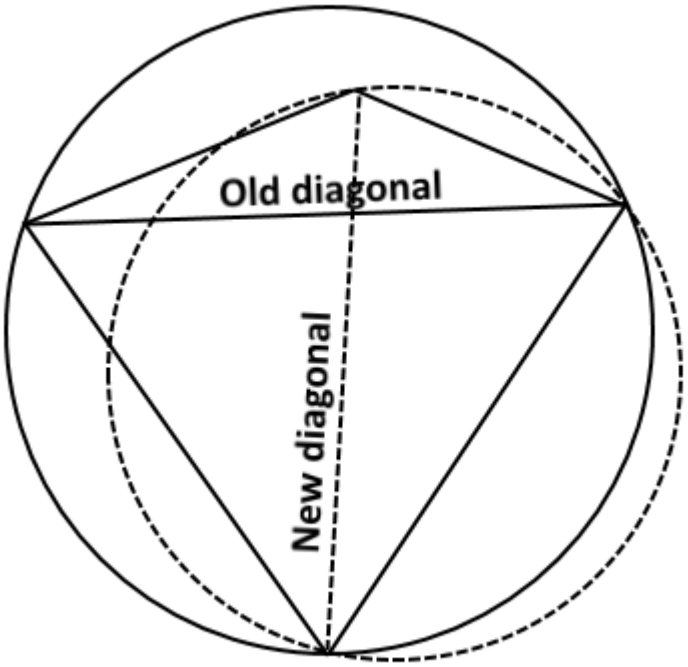


Figure 2.3: Flipping the Diagonal
Circumcircle illustration of flipping the diagonal in a quadrilateral defined by two neighbouring triangles in order to attain the local Delaunay property; Old triangles are not locally Delaunay, New triangles are locally Delaunay.

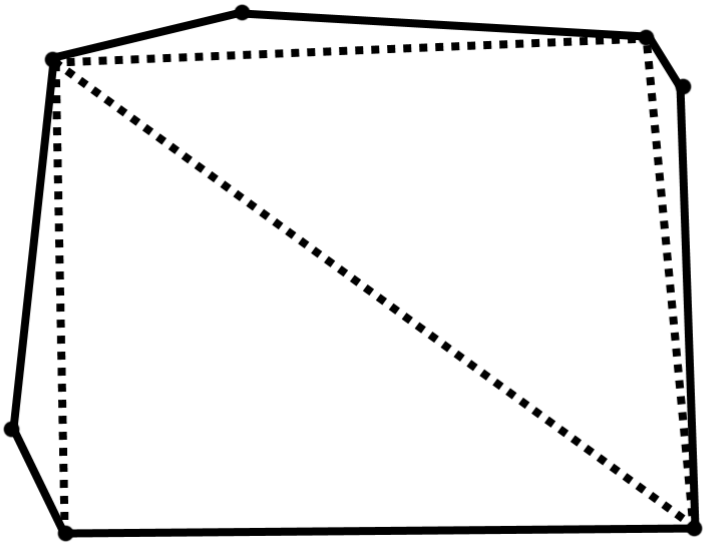


Figure 2.4: Inconvenient result of ear-clipping
As ear clipping result has no guarantees in terms of mesh triangle shape, the result may not be appropriate for navigation.

There are several properties of the flip algorithm that make it beneficial in the context of polygon meshing. First, the algorithm itself is not limited by the constraints of the polygonal space. The algorithm does flip only such pair the of triangles that create a convex quadrilateral, so any triangles surrounding a non-convex vertex of the polygon are not interfering with the algorithm directly. Therefore, it can be claimed that any edge is either part of the boundary or has the Delaunay property, as otherwise it would be flipped. In general, there are several drawback in use of the flip algorithm. First, it has a limitation in its complexity of $\mathcal{O}(n^2)$. Second, the algorithm is similar to the planar set Delaunay triangulation, as it uses only the given point set. While this may be advantageous in some cases, as the number of triangles is limited, the space representation may suffer, since in some cases this allows existence of the thin triangles.

2.3.2 Ear Clipping

Ear clipping (here as described by [Eberly, 2002] is one of the simplest methods of polygon triangulation, that can be utilized to create first iteration for the flip algorithm. The ear clipping creates such a triangulation, that uses only vertices defined in the original polygon or it's inner hole-polygons. While this method is simple to implement, it has several drawbacks. First, the algorithm is of order $\mathcal{O}(n^2)$, where n denotes number of vertices. Second, the triangulation result is not optimal in any way. See the the Figure 2.4 for illustration of case, where the ear clipping result is highly inconvenient. However, due to the simplicity the algorithm is a good starting point a for any complex triangulation algorithm, that is iterative a requires an initial triangulation (e.g. flip algorithm for Delaunay triangulation).

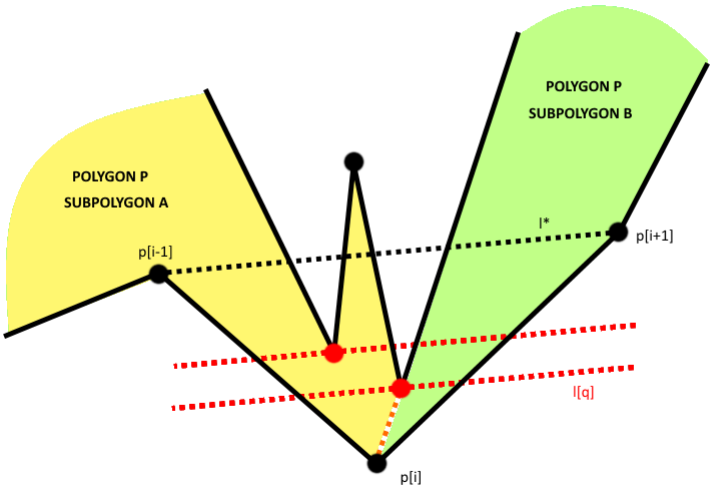


Figure 2.5: Two-Ears Theorem
Polygon Division in Case B

Ear is such a sub-space of a polygon which is defined by 3 consecutive vertices of the polygon (i.e., a triangle), where the middle vertex is convex and no other vertex is placed inside the triangle, i.e., the edge between the two other vertices is inside the polygon. It has been proven that any non-triangle simple polygon has at least two non-overlapping ears. The so-called two-ears theorem was published by [Meisters, 1975].
Definition 2.3.1. Two-Ears Theorem Except for triangles, every Jordan polygon has at least two non-overlapping ears

The proof is inductive. The base claim is that any polygon with 4 vertices has exactly two ears. For the induction itself, consider a polygon P_n with n vertices, $n > 4$, and 3 consecutive vertices p_{i-1}, p_i, p_{i+1} of the polygon, where p_i is convex. Now, two cases are possible. First, it may be possible to remove an ear located at p_i. In such a case, the polygon P_{n-1} resulting from such removal contains at least 2 non-overlapping ears from the induction. Since the ears are non-overlapping, at least one does not contain both the vertex p_{i-1} and p_{i+1}. Together with the removed ear at p_i they represent the two non-overlapping ears of polygon P_n.

Second, there is no ear located at p_i. In this case, consider a line l^* defined by p_{i-1} and p_{i+1} and vertex q located inside the triangle defined by p_{i-1}, p_i and p_{i+1}, which is defined by line l^q, which is parallel to l^* and is closest to p_* of any such lines. Construct a line between q and p_i and split the polygon into two sub-polygons. See Figure 2.5 for illustration. Each of these two polygons has lower number of vertices and therefore is either a triangle or has two non-overlapping ears. If a polygon is a triangle, locate an ear of the original polygon by locating the vertex other than q or p_i. If it is not a triangle, it has at least two non-overlapping ears. Locate an ear that is not at vertex p_i or q. As these two ears are originating from different sub-polygons and are ear of the polygon P_n (i.e., are not located at q or p_i), we found the two non-overlapping ears.

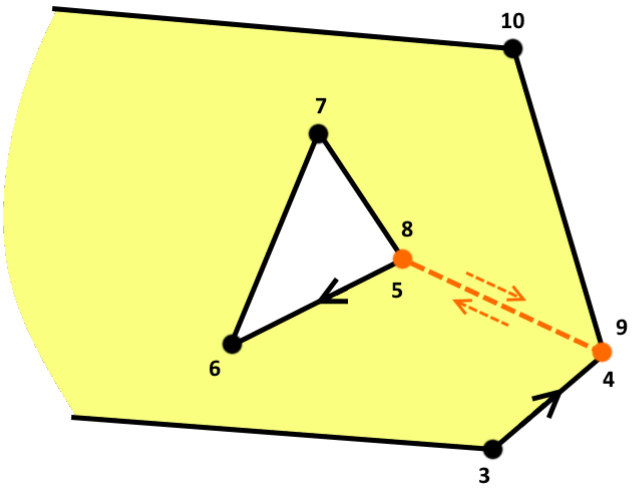


Figure 2.6: Ear Clipping
Connecting a Hole to the Polygon

The ear clipping algorithm itself is not dissimilar from the theorem. In each step of the algorithm, find an ear of the polygon and remove it. Do this until the polygon is a triangle. The resulting triangulation is represented by this triangles and the removed ears. As in each step it is necessary to check the whole polygon in the the worst case, and the number of steps is $n-2$, the algorithm is of complexity $\mathcal{O}(n^2.)$

While the basic algorithm can be applied only to a simple polygon with no holes, it is possible to generalize it to algorithms with holes by adding the holes into the basic polygon. Choose a coordinate from the coordinate system. Without loss of generality, in the following paragraph the chosen coordinate is denoted x. Find a hole polygon with most extreme x-value and the respective vertex. Connect the polygon to a visible vertex of the main polygon in such a way that the vertices representing the hole polygon are in anti-clockwise order if the main polygon is clockwise, or vice-versa. Repeat until there are no holes not connected to the main polygon. See Figure 2.6 for illustration.

To identify a vertex visible from the extreme vertex H on the hole polygon, intersect the edges of the outer polygon with a half-line described as $H + (0, t), t > 0$ with the edges of the main polygon. As H is the most extreme point of all the hole polygons, there will be no intersection of the half-line and any of the edges of the hole polygons. Denote the intersection of an edge and the half-line that is closest to H as I. If I is a vertex of the main polygon, a visible vertex was found. Otherwise, denote the endpoint of the edge where I is located as P and collect all reflex vertices of the main polygon other than P. If all these are outside of triangle HIP, P is visible. Otherwise, chose such a vertex R from the set that the angle RHI is minimized. See Figure 2.7 for illustration.

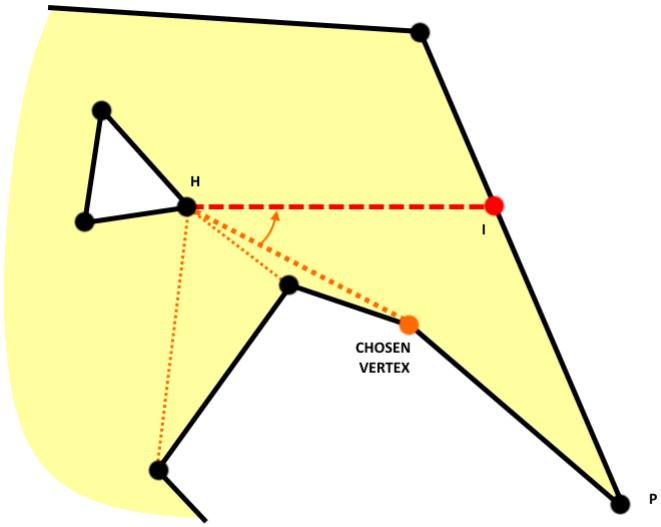


Figure 2.7: Ear Clipping
Finding Angle Minimizing Reflex Vertex

2.3.3 Divide and Conquer Algorithms

Divide and Conquer algorithms for CDT creation use simple space division methods that create easily-triangulated areas and process them in parallel manner. Examples of such algorithms are [Chew, 1989] and [Hardwick, 1997]. While it is possible to modify the algorithm to use Steiner points in order to ensure the optimality of triangles in both area and angles using a method created by [Ruppertt, 1993], the following paragraphs describe the algorithm in its original form presented by [Chew, 1989]. We introduce the simple original variant for illustration, as alternative approach with same space and time complexity which is easier to implement was adopted into the framework.

Generally, in any CDT divide and conquer algorithm we need to divide the space into subproblems, where the initial calculation is executed. In case of Chew's original algorithm, the chosen spaces are strips. Each strip contains one of the vertices. Chew uses vertical strip, but if, by any chance, there are vertices with same x coordinate, it is possible to change the chosen coordinate or rotate the whole coordinate system by some appropriate angle without loss of generality. After creation of the strips, areas of interest have to be located in each strip in order to improve the complexity of the algorithm. Area is defined by edges of the graph, or in our case edges of the polygon, intersecting with the strip. Only edges that have both endpoints outside of the strip are considered. Area of interest is such an area in the strip around a vertex, that is defined by the 'walls' of the strip and two edges from the mentioned set: the closest edge that is above the vertex, and closest below. A structure containing information about such edges can be created in $\mathcal{O}(nlogn)$, where n denotes the number of vertices. This is necessary, as intersecting all the edges could be of complexity $\mathcal{O}(n^2)$. See Figure 2.8 for illustration. After this step, the triangulation is created for area of interest. This is done introducing artificial vertices of the areas of interest. These

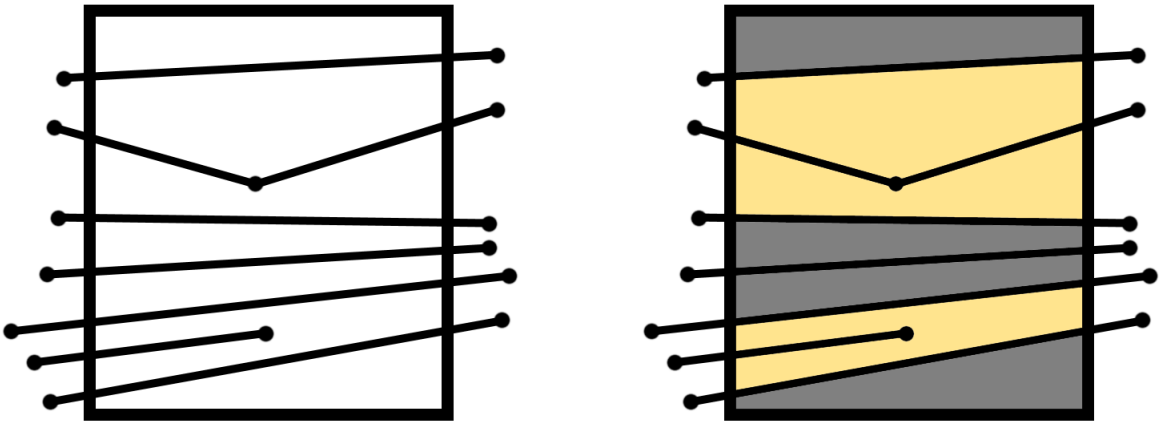


Figure 2.8: Divide & Conquer - Strip Creation
Strips are created if and only if the strip (of minimal size) contains a point.
Based on Figures in [Chew, 1989]

vertices are not represented by their actual coordinates, but rather by infinite coordinates (i.e., $(+\infty, +\infty)$, $(-\infty, +\infty)$, $(+\infty, -\infty)$, $(-\infty, -\infty)$). The variant of infinite coordinate is chosen by its position in regards of the area of interest. For example, left-down corner of the area of interest is $(-\infty, -\infty)$.

The next step in the algorithm is merging neighbouring strip together. This is done in two steps. First, new areas of are created. Second, as this process removes some of the vertices that are artificially created by creation of areas of interest, it is necessary to partially recalculate the triangulation. The first step is done by moving in the strip in vertical fashion, and starting an area of interest if there is a strip beginning in one of the merged strips, and ending it if there is not an area of interest in either. This process can be done in $\mathcal{O}(n)$, where n denotes number of vertices, as it is affected by the number of vertices in the strips and number of edges that have only one end in the combined strip.

To combine and recalculate the triangulations, first it is necessary to remove the artificial vertices that are placed in the interior of the new strip. While in general case removal of vertices can disrupt the triangulation, in the case where only infinite vertices are removed, finite CDT is not affected. See [Chew, 1989] for the formal proof. After the removal of such vertices, it is necessary to add new artificial vertices where necessary. For new edges that cross the boundary between the two strips, it is necessary to investigate adding new edges to triangulation. Denote such an edge as AB. There is no indication whether the desired third vertex, denoted X, is in the left or right strip. It is necessary to find best candidate in both strips and chose one by comparison. Without loss of generality, X lies in left strip. It can be proven that there exists an edge AX in the left strip. Moreover, the edge is Delaunay. Again, for the proof see [Chew, 1989]. From all edges AY the appropriate one can be chosen in the following way. Order the suspect vertices counter-clockwise around A from AB. Choose candidate Y if triangle AYZ, where AZ is the next edge around A, does not exist or AC is one of the original edges of the polygon. If the triangle does exist, consider circumcircle defined by triangle ABC. If D is inside, eliminate AC and test AD.

Although the presented algorithm is of same complexity as other CDT algorithms, as its complexity is of $\mathcal{O}(nlog(n))$, it has two major disadvantages. First, the implementation is more complex than in case of other CDT algorithms. Second, the algorithm does not provide any guarantees for quality of created triangles, as it does not include Steiner points. However, a divide and conquer algorithm with Steiner points was created by [Ruppertt, 1993]. Third, the size of the strips, and therefore the shape and size of the triangles, is dependent on the distribution. [Hardwick, 1997] claims, that for a stripping algorithms to work optimally, the distribution would have to be uniform, but this is an unrealistic assumption in many situations.

2.3.4 Sweep-Line Algorithms

Sweep line algorithms are CDT algorithms that compute the triangulation using a forward moving division line. The area behind the line is triangulated, or at least processed in some other manner. On the contrary, the area that the line is moving towards is waiting for processing. Originally, this approach was developed as Voronoi diagram algorithm by [Fortune, 1987]. It should be noted that as the Voronoi diagram is dual graph of Delaunay triangulations, the algorithm can be considered as Delaunay triangulation algorithm. While its main output and purpose is not the triangulation, and therefore we can consider (and this work continues to do so) the Chew's approach to the first specialized Delaunay triangulation algorithm, in his work Fortune did acknowledge the possibility of using the algorithm for Delaunay triangulations and proposed several modifications to switch the algorithms output directly to triangulation, so that it is not necessary to create the triangulation from the Voronoi diagram.

Again, we introduce the simple original variant for illustration, as alternative approach with same space and time complexity which is easier to implement was adopted into the framework. The algorithm is based on moving two boundary forward. First is the sweepline. This is the line that determines whether a point was already put into the diagram. The second is the 'beachline' or 'borderline'. Contrary to its name, this boundary is not a line, but rather a combination of parabolic segments defined around the Voronoi diagram points. The points where these segments intersect have therefore same distance to both the points. In this manner the movement of the second line creates the diagram itself.

There are two lists in the algorithm. There is the point list and the region and boundary list. The algorithm runs through the point list ordered by the coordinate the lines move along. There are two types of points. First are the site points. Those are the points defining the Voronoi diagram, added before the start of the algorithm. If such a point is encountered, appropriate region is located in the region list and new boundaries are added represented as rays. Also, respective intersection points are added into the point list. If an intersection point is encountered, the boundaries that created it are switched to line segments and any other intersection they may create are removed. New possible intersections are added.

The algorithm is of the same time complexity as the other CDT algorithms, specifically of $\mathcal{O}(nlogn)$, where n denotes the number of point. Its space complexity is $\mathcal{O}(n)$. The time complexity is limited mostly by number possible boundaries and points in the lists, limited by $\mathcal{O}(n)$, and $\mathcal{O}(log(n))$ to operate in such structure if implemented efficiently. See [Fortune, 1987] for detailed proof.

2.3.5 Incremental Algorithms

Incremental algorithms are the third traditional variant of CDT algorithms. The basic principle of such algorithms is inserting a new, yet unprocessed point into the triangulation. [Guibas et al., 1992] is considered to introduce this approach, although in his work it is acknowledged that this approach is not entirely novel. However, he is the first to present an incremental algorithm that is of $\mathcal{O}(nlog(n))$ time complexity and $\mathcal{O}(n)$ space complexity. The previous incremental algorithms were of $\mathcal{O}(n^2)$ time complexity.

The Guibas incremental algorithm in its original form does not include Steiner points. However, from the nature of the algorithm it appears to be the optimal algorithm to be improved upon by adding such a feature. The algorithm is simple when compared to the two previously mentioned types while it retains the same complexity. According to (not only) [Guibas et al., 1992], this simplicity is often considered to be the main advantage of the algorithm, along with the algorithm needing no additional structures to store information needed for computation. However, it also appears that unlike the divide and conquer approach, the incremental algorithm has limited potential for dynamic approach.

The algorithm works as follows. In each step, an unprocessed point is chosen at random. The triangulation created so far is searched for a triangle the point belongs in. The respective triangle is split into three new triangles and each of the edges of the original triangle is checked for Delaunay property, if this is needed (i.e., the existence of the edge is not enforced in some way). If this check fails, the two triangles are flipped. This step is not dissimilar from the flip algorithm examined in Section 2.3.1. The initial triangle is defined by infinite point. This does not interfere with the Delaunay property of the triangulation in similar manner as the infinite border points employed in [Chew, 1989].

The most demanding step of the algorithm is locating the triangle a point belongs to. Following method is employed to perform efficiently. By storing full history of triangulation, it is possible to trace the desired triangle by starting in the initial infinite triangle and following the structure by checking the triangles that originate from the last located triangle. When utilizing such structure, it is possible to run the triangulation in $\mathcal{O}(nlogn)$. See [Guibas et al., 1992] for proper formalization and proof.

While the original Guibas algorithm can be easily modified to retain the enforced CDT edges in the flipping phase, it does not provide a possibility to insert such edges into the graph. The following edge insertion method was created by [Anglada, 1997]. The edge insertion is executed in three steps. First, any triangles cut by the inserted edge are removed. Second, the edge is inserted into the graph. Third, the two regions created by removal of triangles and divided by the edge are triangulated. While the first two steps are simple, the third requires to be investigated further. There are two facts to be noted about the triangulation of the two regions. First, as demonstrated in the Figure 2.9, the regions may not be polygons even if the global triangulated region is a polygon, especially if the algorithm includes Steiner points insertion. Second, it is not necessary to check for any edges of the triangles to be other enforced edges, as this would implicate that the edges cross, which is not possible.

The triangulation of the two regions is a recursive process. First, locate such a point p among the points in the region that the circumcircle of triangle defined by the dividing edge and p does not contain any of the other points and form the triangle. Second, divide the region (i.e., the list of points) by the triangle into two new regions. Apply the process recursively

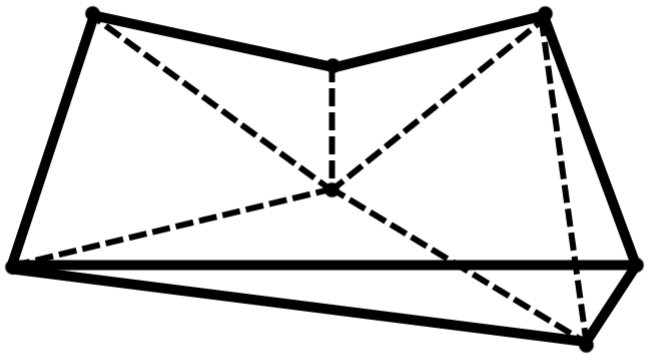


Figure 2.9: Incremental algorithm - Edge Insertion Region
Example of a region created by removing obstructive triangles as used in edge insertion. Notice that two vertices with same projection on the edge make the area non-polygonal in the strict sense.
Based on Figures in [Anglada, 1997]

for any of the regions, under the condition of being non-empty, while using the respective edge of the added triangle in same way as the dividing edge was used in the first step. If a point q is the cause of the original region being not a polygon as was demonstrated in Figure 2.9, add it into both the regions. This process does retain the constrained Delaunay property of the triangulation, as the points are the same as in the previous triangulation and therefore the circumcirles are still empty. For proper formalization and proof of this claim see [Anglada, 1997].

2.3.6 Delaunay Refinement

While better than other counterparts, Delaunay triangulations often contain some badly shaped triangle, whose presence would not be convenient in the final triangulation. Often, the criterium for acceptable shape is the size of smallest angle and ratio between circumradius and inradius. While not the only method to refine the triangles, as quadtree algorithms are also used, originally proposed by [Baker et al., 1988], elements, the triangulation is often refined by insertion of additional points, most commonly Steiner points located at circumcentres of the badly shaped triangles, although there are several other methods how to choose the desired points, such as the offcentres proposed by [Üngör, 2009]. In general, there are several limitation that has to be considered when implementing any refinement algorithms. As both Rupert and Ungor remark, most of the algorithms presume that angles between two enforced graph edges are higher then 90, which may not always be the case in real application.

The following section describes the Steiner point selection and insertion roughly as proposed by [Ruppertt, 1993]. Rupert recognizes two actions: splitting a triangle, and splitting a segment. Splitting a triangle is done by inserting a circumcentre point of the said triangle. Segment, that is an edge existing in the triangulation, is split by inserting a point that divides the line segment into two equal halves. The algorithm first requires a Delaunay triangulation. Even though there is no requirement on the computation process of the Delaunay triangulation, it would seem beneficial to employ an incremental algorithm, as it is possible to use the same approach in inserting the Steiner points. Rupert presumes that a normal Delaunay triangulation is computed, and therefore in his algorithm is also necessary to enforce existence of any enforced edge that is not present. In this description this step is omitted, as the algorithm is presumed to be combined with an incremental CDT algorithm. Rupert enforces the existence of the edges by checking in each step and splitting the edge in similar fashion as in Steiner point insertion, and not in the generalized point insertion step, as incremental algorithms do. However, as both Rupert and Ungor remark, most of the algorithms presume that angles between two enforced graph edges are higher then 90, which may not always be the case in real application. Therefore, even if using algorithm with guaranteed termination, it is usually necessary to enforce termination in some way.

After obtaining the triangulation, repeat the following: choose any badly shaped triangle and obtain its Steiner point. If Steiner point is encroaching on the any enforced edge of the triangulation, i.e., the point is in a circumcirle of said edge, split all such edges. If no edge is encroached, split the triangle. Repeat until no badly shaped triangle is present, or other termination condition, present for reasons stated previously, is reached. To categorize the elements as badly shaped, it is necessary to choose a threshold minimal angle. Rupert provides proof, that for angles larger than 20 the algorithm is bound to terminate. It should, however, be noted, that it is possible to choose larger angles, especially if there is an enforced termination condition. The proof is not provided, as it is lengthy and, from definition, its importance is diminished by the fact that it is not applicable to any set of enforced edges in the triangulation, as its requirements cannot be met in triangulation of a general polygon.

Chapter 3

Implementation

In the following section we describe the implementation of mesh navigation on the agent crowd project. The aim differs from the previous section, which described various algorithms on general theoretical basis. This section does not describe only the mesh creation algorithms, but also other tasks that were necessary to implement. In general, the structure of the implementation is based around the creation of navigation meshes. Taking some representation of the simulation world, its representation in convex mesh polygons is created the polygons are assigned some potential values.

3.1 General Structure

In order to reach the aim of mesh creation, wide variety of object types is needed. Some objects, such as the implementations of MeshPolygon, represent the mesh polygon structure. Others represent one-use creators with the task of processing input data into the mesh objects. Finally, other objects serve as auxiliary in the process of mesh creation. Excluding changed structure of the agent and sensor classes, there are four main object types that were added into the framework. It should be noted that 3 of these are either abstract classes/interface. This approach was adopted so that different new algorithms and special polygon types could be easily inserted. The 1 non-abstract class is

- InputPolygon, which stores information about the world regions, obstacles and exit zones for the mesh creation process

The 3 abstract classes are as follows

- MeshPolygon, which represents a presumably convex polygon in the representation used in navigation
- MeshCreator, which is responsible for processing the world represented by InputPolygons into structure of MeshPolygons
- PotentialCalculation, which assigns each MeshPolygon with a value based on some potential calculation

The following paragraphs describe these classes as they are in their abstract form. For information on the implementation of the abstract classes see Sections 3.2.1, 3.2.2, 3.2.3 and 3.2.4.

3.1.1 InputPolygon

InputPolygon is an area in the simulation input world. The area represented by an InputPolygon may vary substantially. It may be a representation of the complete space, an obstacle, or an exit zone. Input Polygon is represented by an array of points, representing its boundary, list of obstacles and other objects that are located inside the InputPolygon, and array of String tags categorizing the area as an impassable obstacle, exit zone, empty space, etc..

The array of points representing the boundary is a rather simple structure. The points are ordered in either clockwise or counter-clockwise fashion. There is no guarantee for the ring to start at an extreme point.

List of obstacles (also represented as InputPolygons) a tree-like structure of polygons originating from the highest level polygons. It is guaranteed that while the obstacle InputPolygons located in the same InputPolygon can touch each other, they do not cover any shared area and do not touch the outer boundary. This is done in following way: starting with a general InputPolygon representing the whole area with all polygons suspected to be inside, first check for every polygon whether every obstacle is located in the interior. Obstacles placed outside the boundary are removed. If any obstacles is touching the boundary from inside, the boundary is cropped and the process is restarted and separate procedure is executed for the obstacle, which is made into new InputPolygon located on the same level in the InputPolygon hierarchy. After successful boundary check the obstacles are repeatedly checked against each other. If an obstacle is encapsulated by another, it is removed from the obstacle list and added to the obstacle list of the encapsulating obstacle (i.e., the obstacle is moved down in the InputPolygon hierarchy). If the obstacles have an shared area, but are not encapsulated, new obstacles representing the smallest continuous areas representing the union, obstacle 1, and obstacle 2 are created. If neither of the previous cases happens (i.e., the obstacles share no area), nothing is changed.

3.1.2 Mesh Polygon

MeshPolygon represents an non-overlapping convex area in the simulation world. While the basic function of MeshPolygon is close to the InputPolygon, the two differ substantially both in concept and stored data. First, unlike InputPolygon that is used only in the computation and has no connection to the information stored in the abstract layer of the simulation, the MeshPolygon's only use is in the navigation. Every MeshPolygon is assigned a set numerical value representing its pseudo-distance to a respective target zone. Second, unlike the InputPolygons that store information about other possible polygons located in their interior, convex MeshPolygons store no such information. On the other hand, MeshPolygons store information about their neighbours. Two MeshPolygons neighbour each other if and only if they share a border segment. Both InputPolygon and MeshPolygon carry String tags describing the type of area being represented. To illustrate on difference of possible world breakdown into InputPolygons and MeshPolygons, see Figure 3.1.

MeshPolygon is represented by its points, neighbours, tags, and potential values. However, as the MeshPolygon is a modular object that is represented as an abstract class in the framework, concrete storage structure of the defining features is not enforced. Also, it should

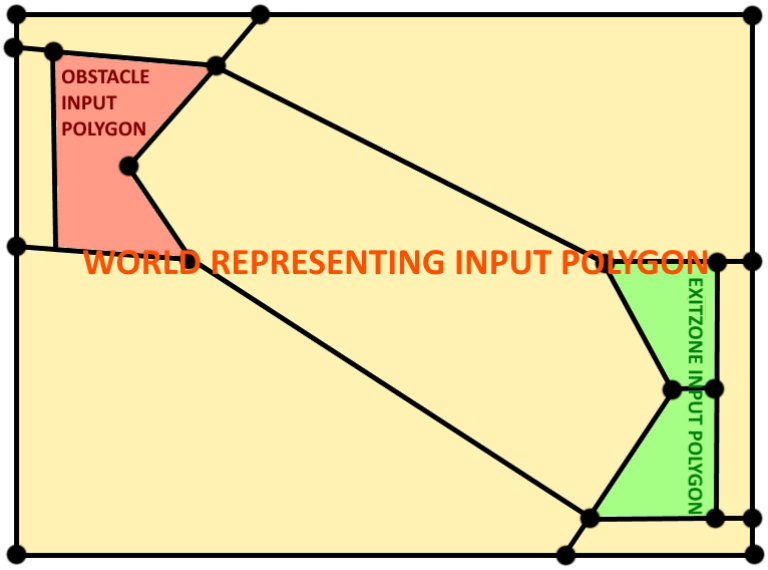


Figure 3.1: InputPolygon & MeshPolygon Representation of Space
Example of the difference between InputPolygon and MeshPolygon space representation. InputPolygons are represented by background colour and captions, MeshPolygons are represented by edges. Note that:

- the largest InputPolygon contains smaller input polygons
- there are no MeshPolygon representing the non-traversable space, i.e., the obstacle InputPolygon
- all the MeshPolygon are convex
- the MeshPolygon representation is roughly equivalent AnavMesh algorithm output

be noted that existence of ¨0 area¨ MeshPolygons is possible. This is necessary, as some of the chosen mesh-creation solutions may result in existence of such. See Sections 3.2.3 and 3.2.4 for further clarification.

3.1.3 MeshCreator

MeshCreator is an abstract class that processes InputPolygons/world information into MeshPolygons. The InputPolygons structure is either inserted into the MeshCreator as a Collection of highest level InputPolygon objects, or similar collection is computed from the world information. Some implementation may require additional parameters. For example, the IncrementalDelaunay, described in Sections 2.3.5 and 3.2.3 requires minimal area threshold. Also, a MeshCreator offers possibility to extract list of tags that are used in the simulation, i.e., there exists a MeshPolygon that has such a tag.

3.1.4 PotentialCalculation

PotentialCalcualtion is an abstract class that assigns InputPolygons with set of values used for global navigation. In the current implementation, the set of values represents a pseudo-distance to respective exitzones. However, this is not enforced and if the framework were to be expanded with goal selection, more specific navigation approaches could be implemented.

3.2 MeshCreator Implementations

At the present time there are four MeshCreation algorithms implemented in the AgentCrowd framework. Of the four, two serve mostly as proof of concept of mesh navigation in the AgentCrowd framework. The two are

- EarClipping, implementation of ear clipping method of triangulation
- FlipDelaunay, a Delaunay triangulation method with no refinement

The other two methods are

- AnavMesh, implementation of meshing approach proposed by [Oliva and Pelechano, 2012]
- IncrementalDelaunay, an CDT triangulation with further refinement using Steiner points

3.2.1 EarClipping

EarClipping is the first implementation of MeshCreator in the AgentCrowd framework. As such, its main purpose was proof of concept for agent navigation via mesh polygons. Secondary purpose of EarClipping is to provide easy implementation for triangulation algorithms that require an initial triangulation. While EarClipping creates a system of triangular meshes, it is not suited to be the chosen solution, as the space representation can create triangles that are highly inconvenient. See Figure 2.4 for illustration. See Section 2.3.2 for background information about the ear clipping algorithm.

The algorithm is implemented as in [Eberly, 2002] with few modification that are required for it to function as a MeshCreator implementation. The most important modification is that

the algorithm tracks neighbours while creating the triangles. This is done in each step after cutting the ear. As finding the neighbour is $\mathcal{O}(t)$ where t denoted the number of triangles and is located on same level of the algorithm as locating the ear, which is $\mathcal{O}(n)$ where n denotes number of points and trivially $t < n$, the algorithm is of unchanged complexity.

```
Data: Polygon and its ¨hole¨ obstacles represented as list of points; the polygon and
      the obstacles should have different clockwise order
Result: structure of mesh triangles representing the polygon without the obstacles;
        the triangles contain neighbourh information
processPolygons(polygons)
while border polygon size > 2 do
    forall vertex V in border polygon do
        if V is convex then
            forall vertex U other than V_-1, V, V_+1 in border polygon do
                if U is inside triangle V_-1, V, V_+1 then
                    go to next V
                end
            end
            create triangle T as V_-1, V, V_+1
            remove V from border polygon
            forall triangle S do
                if should be neighbour then
                    add neighbour information
                end
            end
        end
    end
end
forall obstacle O do
    run ear clipping
    forall triangle T in this triangulation do
        forall triangle S in obstacle triangulation do
            if should be neighbour then
                add neighbour information
            end
            add S in this triangulation
        end
    end
end
```

Algorithm 1: Ear Clipping

3.2.2 FlipDelaunay

FlipDelaunay is the first implementation of a constrained Delaunay triangulation and second implementation of the MeshCreation. This is the implementation of flip delaunay approach

```
Data: outer border polygon and obstacle polygons
Result: outer border polygon containing points from the other polygons
while there is an unprocessed obstacle polygon do
    forall unprocessed obstacle polygon P do
        forall point X in P do
            if X.x > maxX.x then
                maxX ← X
                maxP ← P
            end
        end
        forall edge E in main border polygon do
            suspectI ← intersection((1,0) + maxX, E
            if distance(maxX, suspectI) < mindist then
                I ← suspectI
                IE ← E
                mindist ← distance(maxX, suspectI)
            end
        end
        if IE.tail I then
            addPoint ← I
        end
        else if IE.head I then
            addPoint ← I
        end
        M ← maxOnX(IE.tail, IE.head)
        forall reflex vertex V in main border polygon do
            if V inside triangle I-M-maxX & V ≠ maxX&V ≠ M then
                hasOneInside ← true
                add(insideVecrtices, V)
            end
        end
        if hasOneInside then
            addPoint ← I
        end
        else if  forall vertex V in insideVertices do
            if degree(vector(1,0), vector(minX,V)) < mindeg then
                mindeg ← degree(vector(1,0), vector(minX, V))
                addPoint ← V
            end
        end
         then
        end
        remove addPoint from border polygon
        add addPoint,rest of maxP (starting and ending with maxX), addPoint in
        the border polygon
        mark maxP as processed
    end
end
```

Algorithm 2: Ear Clipping Sub-procedure: process polygons

described in section 2.3.1. Again, the main purpose of this algorithm was to serve as proof of concept, a the algorithm itself is inferior to IncrementalDelaunay in both time and space complexity and is not well suited for Delaunay refinement.

Any MeshCreator that produces triangles can be used as initial implementation of this algorithm. In the original implementation EarClipping is used. As in the case of ear clipping, it was necessary to make some additions to the algorithm in order to track neighbours. Similarly, the complexity of the algorithm is unchanged, as it is easily possible to track while flipping the triangles.

Data: any structure of mesh triangles representing the polygon without the obstacles; the triangles contain neighbour information
Result: similar structure with local Delaunay property representing the polygon
while *goCondition* **do**
  goCondition $\leftarrow$ *false*
  forall *triangle T* **do**
    forall *neighbour triangle R of T* **do**
      if *tags of R & T differ* **then**
        continue to next R
      end
      quadrilateral $\leftarrow$ *considerQuadrilateralFromTriangles*(T, R)
      if *sum of angles at vertices at vertices without diagonal* $> \pi$ **then**
        remove R & T
        add new triangles created by dividing quadrilateral using the second diagonal
        goCondition $\leftarrow$ *true*
        break all for cycles
      end
    end
  end
end

Algorithm 3: Flip Delaunay

3.2.3 IncrementalDelaunay

IncrementalDelaunay is the main implementation of constrained Delaunay triangulation. The implementation is based on algorithm described in Section 2.3.5 modified by adding the Steiner point insertion as described in Section 2.3.6. The algorithm is parametrized by two values: the smallest angle, and the smallest triangle area. The smallest angle denotes threshold minimal angle that any triangle can have. Naturally, angles of triangles with two or three enforced edges (i.e., edges that are outer or inner boundary of the polygon) are not affected. However, such triangles are still affected by the algorithm. The smallest area denotes threshold minimal area of triangle that can generate a Steiner point in the refinement process. This threshold ensures termination of the algorithm.

The algorithm is as follows: process the InputPolygon structure for highest level to the lowest, not unlike in the EarClipping. When processing an InputPolygon, create a list of

points to be processed. The list contains points from the boundary of the InputPolygon and boundaries of the relevant obstacles polygons. It should be noted that in order to retain consistent border sections in neighbouring areas, relevant point on the boundaries that are located only on the boundary of the neighbouring polygons are added. The initial triangle representing the highest level triangle not created infinite as in the original algorithm, but rather large enough to cover all the points. This does not have lead to different triangulation result, as the triangulated object is polygon and the triangles relevant for the resulting triangulation are all isolated from the outermost triangles by enforced edges. After the lists and initial triangle are created, add the points to triangulation until the list is empty.

After any new triangle is created, if the angle and are thresholds add relevant Steiner points to the point list. It should be noted that when the a Steiner point is to be added to the triangulation, the triangle it originated from is checked for relevancy - the addition process is only executed for triangles that are still leaves in storage structure (i.e., if the result was to be recovered in this step, these triangles would be part of the final triangulation; see Section 2.3.5 for more information on the triangle storing structure).

After the point insertion is finished, three additional have to be executed. First, locate the the relevant triangles that represent interior of the processed polygon. This is done by propagation via the neighbourhood information and is of time complexity $\mathcal{O}(t)$ where t denoted the number of triangles. Second, check every triangle that is located on a border (i.e., some of its edges does not have a neighbour information) and check, whether any of the original polygonal edges was divided by the point insertion process. If this was so, create zero area connectors in order to ensure edge consistency between triangulation of neighbouring polygons. See Figure 3.2. Third, after obtaining the triangulations of lower lever obstacle polygons, add the missing neighbour information for edges bordering these polygons. This step may of space complexity $\mathcal{O}(b * t)$ where t denotes the number of triangles in the lower level triangulations and b denotes the number of triangles with missing border information in the triangulation. This may equal space complexity of $\mathcal{O}(t^2)$, however, this would be so only when a triangulation where no or very few additional points were inserted is considered. This is not the case in most examples of incremental triangulations.

3.2.4 AnavMesh

AnavMesh is the main implementation of non-triangulation meshing algorithm in the AgentCrowd framework. It is based on the algorithm proposed by [Oliva and Pelechano, 2012]. See Section 2.1 for more more detailed description of the algorithm.

The proposition of the AnavMesh algorithm by [Oliva and Pelechano, 2012] appears to be too general to be implemented directly into the framework. While Oliva generally refers to polygons while describing the algorithm, we found out that the algorithm is easier to implement if vertices and edges are considered as the main building blocks of the world and the polygons are only reconstructed afterwards. Each point is represented by its location and its edges (i.e., references to other points). In the algorithm steps the non-convex vertex is located by simply checking angles between edges at points. For this goal the edges are sorted based on their angle to a fixed vector $(1, 0)$. For a selected non-convex point, the process is similar to Oliva's operation on the polygon - the closest point, edge, or portal edge is selected. As the selected entity is both the closest to the vertex and it is located in

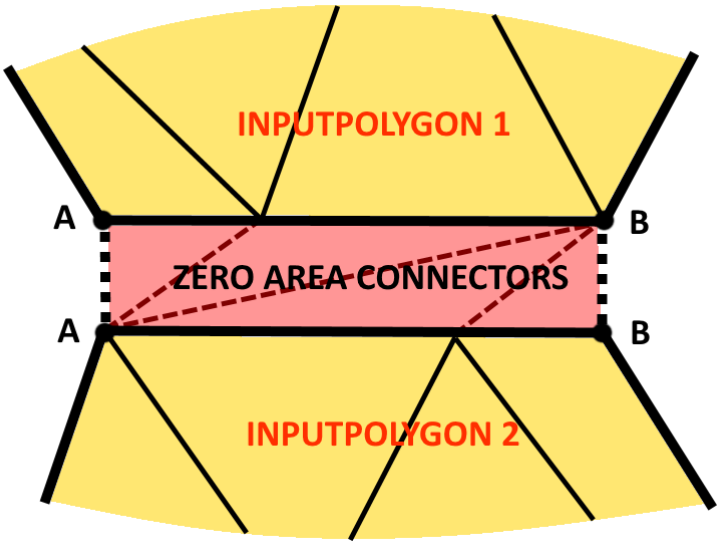


Figure 3.2: Zero Area Connectors
Zero area connectors are included in the triangulation in order to enforce consistency between neighbouring input polygons. Note that the two InputPolygons in this figure were triangulated in such a way that the triangles cannot consistently border each other. Note that the height of red area is 0 and therefore the connector triangles are 3-vertex lines with zero area.

```
Data: InputPolygon structure representing the simulation world
Result: triangle structure with local Delaunay property representing the polygon;
        Steiner points may be added
forall point P in border/obstacle polygon do
    add P in point queue
    set edge information: P has has enforced edge to neighbouring vertices in the
     polygon
end
forall polygon neighbouring the main polygon do
    forall point P in its outer border do
        forall edge E in the main polygon border do
            if P is on E and point queue does not contain P then
                add P in point queue, rest relevant enforced edge information
            end
        end
    end
end
E ← createEncompassingTriangle(allPoints)
while point queue is not empty do
    P ← poll(queue)
    processPoint(P)
end
forall leaf triangle L : process based on neighbourhood do
    if L represents non-obstacle polygon interior then
        retain L
    end
    else
        forget L
    end
end
forall edge E of outer polygon with added points from neighbour polygons do
    if multiple triangles represent the border then
        create zero area triangle
    end
end
forall obstacle O do
    run incremental Delaunay
    forall triangle T in this triangulation do
        forall triangle S in obstacle triangulation do
            if should be neighbour then
                add neighbour information
            end
            add S in this triangulation
        end
    end
end
```

Algorithm 4: Incremental Delaunay

```
Data: point P
Result: all sub-triangles E (including E) that contain P
if P is bound to triangle and the triangle is a not leaf then
    continue to next P
end
trianglesToDivide← locate(E, P)
if trianglesToDivide size 0 then
    continue to next P
end
else if trianglesToDivide size 1 then
    P is in interior of a triangle
    divide such a triangle into 3 new triangles
    set their neighbour information based on neighbour information of divided
    triangles
    try flipping the new triangles with their outside neighbours
    set new triangles as sub-triangles of the divided triangle
end
else if trianglesToDivide size 2 then
    P is on an edge
    forall triangle with P do
        do similar procedure as in the previous case, but divide triangle into 2 new
        triangles
    end
end
forall points N with enforced edges to P do
    if N is already processed then
        locate triangles cut by the edge
        re-triangulate the area in recursive manner
    end
end
forall leaf triangle L the is a sub-triangle of trianglesToDivide do
    if did not pass quality criterium & did pass min size criterium then
        queue: add Steiner point bound to L
    end
end
```

Algorithm 5: IncrementalDelaunay sub-procedure: process point

Data: triangle E, point P
Result: all sub-triangles E (including E) that contain P
if *E does not contain P* **then**
  return
end
if *E has sub-triangles* **then**
  forall *sub-triangle S* **do**
    $add(resultList, locate(S, P))$
  end
end
else
  $add(resultList, E)$
end
return *resultList*

Algorithm 6: IncrementalDelaunay sub-procedure: locate

the interest region, it is obvious that it has to be located inside the same polygon that the non-convex angle belonged to. See Figure 2.2 for illustration of an interest area. It should be noted that other than the vertices representing the polygon and its inner obstacles, there are also 4 vertices representing the area encompassing the polygon. For those 4 vertices the convexity check always succeeds and therefore the algorithm does not try to connect them to any other vertex (note that this would be problematic as there is no vertex they could be connected to).

The polygon reconstruction process is based on similar principles. Propagating from and edge of the encompassing polygon, the polygons are created based on one of their oriented edges. A new polygon is created starting from an edge and then choosing the next edge by looking at next the sorted edge belonging to the tail vertex. The propagation to next polygons is done by starting from the encountered edges in the opposing direction. Note that this creates all the edges either clockwise or counter-clockwise.

Several other fact should be stated about the AnavMesh implementation. First, unlike the previously mentioned MeshCreator implementations which use the Triangle, AnavMesh uses GeneralMeshPolygon, a different MeshPolygon implementation. While the difference is presumed to be mostly insignificant, this could lead to slightly slower performance of the simulation, as in the case of simpler Triangle various assumptions and simplifications could have been made. Second, it should be noted that in order to store neighbour information, steps not dissimilar to those used in IncrementalDelaunay implementation had to be made. See Section 3.2.3 for further information.

```
Data: InputPolygon structure representing the simulation world
Result: convex polygon structure representing the simulation world
forall point P in border/obstacle polygon do
    add P in point queue
    set edge information: P has has enforced edge to neighbouring vertices in the
     polygon
end
forall polygon neighbouring the main polygon do
    forall point P in its outer border do
        forall edge E in the main polygon border do
            if P is on E and point queue does not contain P then
                add P in point queue, rest relevant enforced edge information
            end
        end
    end
end
createEncompassingRectangle(allPoints)
forall convex Point P do
    R ← createInterestRegion(P)
    forall Point Q in R do
        if distance(P,Q) < minDist then
            I ← Q
            mindist ← distance
        end
        forall Neighbour N of Q do
            if distance(QN section in R,P) < minDist then
                I ← QN
                mindist ← distance
            end
        end
    end
    if I is a Point then
        connect P to Q
    end
    if I is line segment and not a portal then
        connect P to its projection on I
        mark the new edge as a portal
    end
    if I is line segment and a portal then
        connect P to its end vertices of Imarkthenewedgesasportalsend
    end
    constructPolygons(AllPoints)
    forall obstacle polygons of this polygon do
        repeat this
        set neighbourhood information
    end
```

Algorithm 7: AnavMesh algorithm

Chapter 4

Evaluation

In this section we compare the adopted solution, i.e., the IncrementalDelaunay and AnavMesh, with the original potential map navigation. In general, total of three variations of adopted approaches are measured. However, in some sections, additional variations of the adopted approaches may be used. The three fully measured method are

- IncrementalDelaunay with small triangle size (minimal size of triangle $\doteq$ 1/1000 of simulation space)
- IncrementalDelaunay with large triangle size (triangles rarely divided by Steiner points)
- AnavMesh

The adopted solution are compared to the original implementation by using three approaches. Two of the approaches are objective measurements of tick time, computation time, and size variables,

- Navigational Structure - measurement of memory used to represent the structure, memory used in structure creation, and time spent on structure creation
- Tick Measurements - measurements of time variables in simulation - e.g. time to calculate navigational vector for an agent

the third is mix of subjective and objective observation of agent movement and positioning

- Agent Pathing - observation of agents' paths

4.1 Navigational Structure

The navigational structure is analysed using memory and time measurements of various aspects of the navigation structure and its creation process. The measurements are

- Representation Size - bytes - the size of the mesh representing entities
- Heap Size After Structure Creation - measurement of memory requirements of the mesh/potential map creation - the results may be inaccurate as there are no guarantees that JVM did not run garbage collection in the creator process
- Computation Time - time needed to compute the mesh/potential map

The variables are measured in three scenario types

Cells	10000	20000	30000	40000	50000	60000	70000	80000	90000
AnavMesh	2.3	2.3	2.3	2.3	2.3	2.3	2.3	2.3	2.3
IncLarge	2.9	2.9	2.9	2.9	2.9	2.9	2.9	2.9	2.9
IncSmall	281	131	166	135	169	186	180	178	145
Potential	5424	10809	16194	21579	26963	32348	37733	43118	48503

Table 4.1: Size Tests: Representations Size <kB>

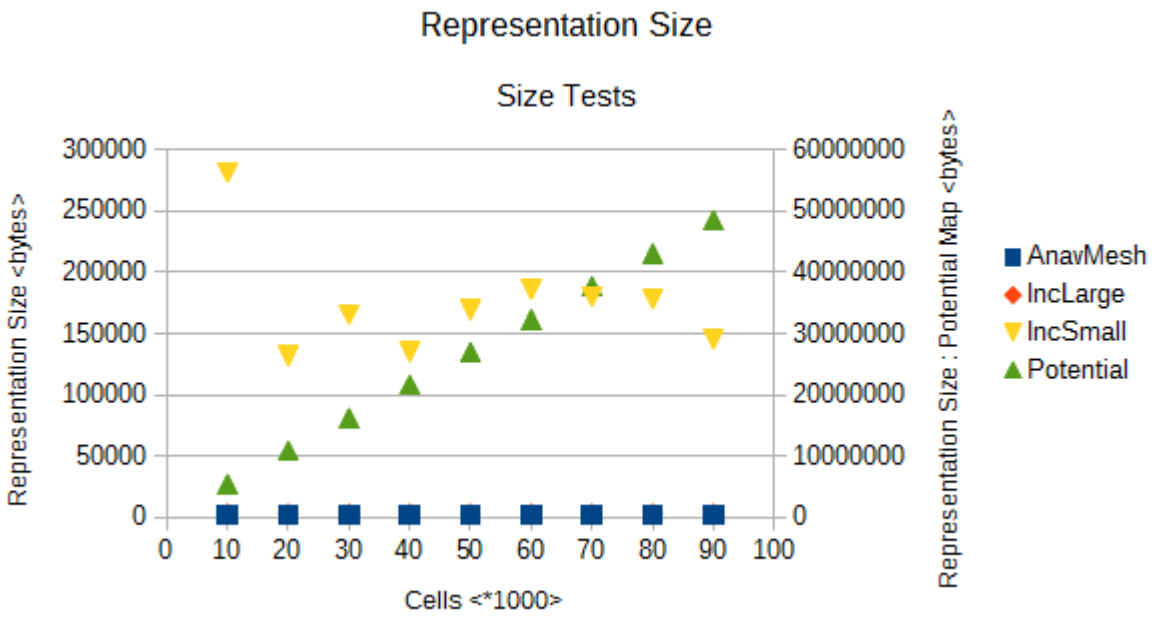


Figure 4.1: Scaling with Scenario Size - Representation Size

- Size Tests - tests based on varying size of the scenario - the scenario is generally empty plane with one entry zone and one exit zone.
- Flow Tests - test based on varying number of agent flows - the test is based on one of the smaller empty size tests, however there are multiple exit zones
- Obstacle tests - test based on varying number of obstacles in the scenario

In the following subsection the results will be provided and interpreted listed by the scenario type.

4.1.1 Size Tests

Size tests are based on varying size of the scenario. The scenarios are generally empty planes with one entry zone and one exit zone. The varying size is provided by its x-axis, while the agents travel along the y-axis. In the result, the travel time of the agents is not affected. There are 9 scenarios ranging from 100000 to 900000 cells.

We can safely conclude that the new approaches have significantly smaller memory representation (Table 4.1.1, Figure 4.1) than the potential maps. Also, it should be noted that

Cells	**10000**	**20000**	**30000**	**40000**	**50000**	**60000**	**70000**	**80000**	**90000**
AnavMesh	46	29	34	40	45	51	56	61	67
IncLarge	44	64	34	40	46	51	55	62	67
IncSmall	29	33	37	43	49	54	59	66	70
Potential	41	70	84	116	128	113	122	144	167

Table 4.2: Size Tests: Heap Size <MB>

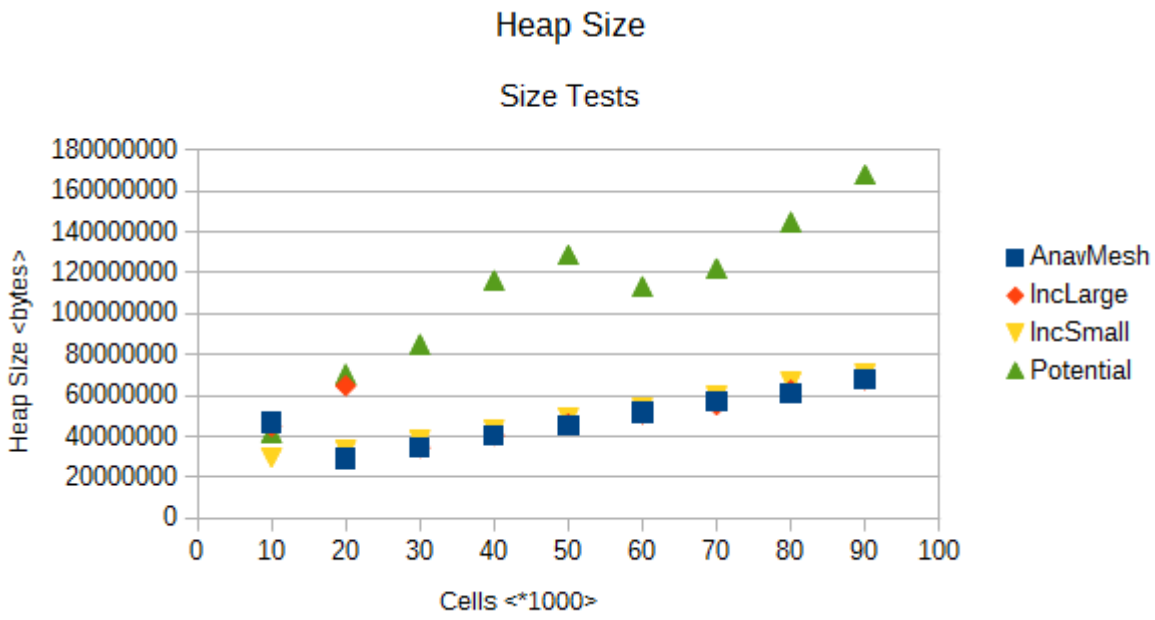


Figure 4.2: Scaling with Scenario Size - Heap Size
Note that some of the series may overlap

unlike the potential maps, the mesh representation does not increase its memory requirements when only the size of scenario is changed. However, several remarks should be noted about the IncrementalDelaunay approach. While the results of IncrementalDelaunay with small triangle size does appear to be somewhat stable with changing cell number, it does not reach the stability of IncrementalDelaunay with large triangles and the AnavMesh. This is caused by the fact that the parametrization of the triangulation actually varies, as it is bound to a fraction of the total simulation space.

The heap size measurement (Table 4.1.1, Figure 4.2) shows linear like trends for all the new and old approaches. However, the old potential map approach is significantly steeper. As the heap size measures the full JVM heap after the mesh/potential map creation, we can presume that linear trend present in the new approach data is caused not by the MeshCreator class itself, but rather by other structures present in the simulation that scale with the simulation size. Some of these structures, such as the grid, are vital for the simulation and cannot be removed to make the measurement more precise. Nonetheless, it would appear that the potential maps scale worse with the amount of cells in the simulation.

While the newly adopted approaches appear to be superior with memory scaling with number

Cells	**10000**	**20000**	**30000**	**40000**	**50000**	**60000**	**70000**	**80000**	**90000**
AnavMesh	218	237	288	282	286	322	415	309	413
IncLarge	154	233	211	249	388	260	280	494	489
IncSmall	3008	1779	1690	1473	1515	1735	1672	1715	1473
Potential	179	280	396	372	394	1015	817	656	722

Table 4.3: Size Test: Creation Time <ms>

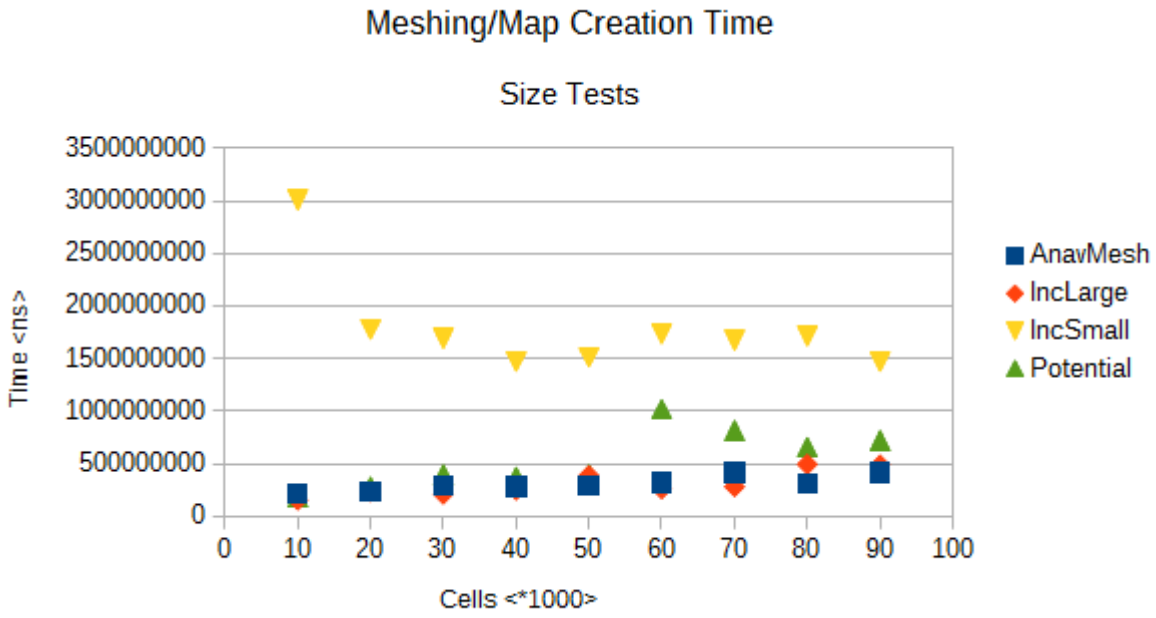


Figure 4.3: Scaling with Scenario Size - Computation Time

Flows	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
AnavMesh	2.3	2.6	3.0	3.3	3.6	4.0	4.3	4.7	5.0
IncLarge	2.9	3.5	4.1	4.7	5.3	6.0	6.6	7.2	7.8
IncSmall	281	257	302	348	394	439	485	530	576
Potential	5424	5564	5705	5845	5985	6126	6266	6406	6547

Table 4.4: Flow Test: Representation Size <kB>

Flows	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
AnavMesh	24	23	24	24	24	24	24	24	25
IncLarge	24	24	24	24	23	24	24	24	25
IncSmall	31	29	29	29	30	29	30	30	32
Potential	45	54	72	82	103	113	36	42	55

Table 4.5: Flow Test: Heap Size <MB>

of cells, the time measurement results (Table 4.1.1, 4.3) show that there is similar scaling with of time with cell amount between the old and new approaches. There appears to be weak linearity/constant trend for IncrementalDelaunay with large triangles and AnavMesh and weak/medium linearity for potential maps. IncrementalDelaunay with small triangles, while also probably constant, has significantly larger time requirement.

In conclusion, the new approaches do not depend on or depend weakly with the number of cells while the potential map approach has linear treand. In all the memory measurements the new approaches proved to be superior. In the time measuremts, the AnavMesh proved to be superior, but it was observed that IncrementalDelaunay result highly depends on its parametrization.

4.1.2 Flow Tests

Flow test are based on varying number of flows. Again, there are 9 scenarios, ranging from 1 to 9 flows. Each scenario is empty space with one entry zone and respective number of exit zones. All the exit zone are located on the same position (i.e., they are represented by same sets of mesh polygons).

From the results of Table 4.1.2 and Figure **??** we can conclude that the size of representation of the new method does not depend on or depends weakly on the number of flows. The potential map representation grows linearly with the number of flows. Several remarks should be made. First, the potential maps have significantly higher memory requirements than the new approaches. Second, the IncrementalDelaunay with small triangles has larger memory requirements than the other new approaches. However, the results are still significantly lower than potential map results. Finally, it should be noted that it was investigated that the weak trend of the new approaches is not caused by the mesh objects itself, which are identical, but rather by the potential value representations of these objects (i.e., target-potential maps in each object), that still have to grow with number of flows.

From the results of Table 4.1.2 and Figure 4.5 we can conclude similar results. The new approaches seem to have constant heap size requirement while the potential map grow linearly with the number of exits. Note that for the higher number of flows the heap size of potential map creator reached its allocated memory and garbage collection was likely enforced by the JVM.

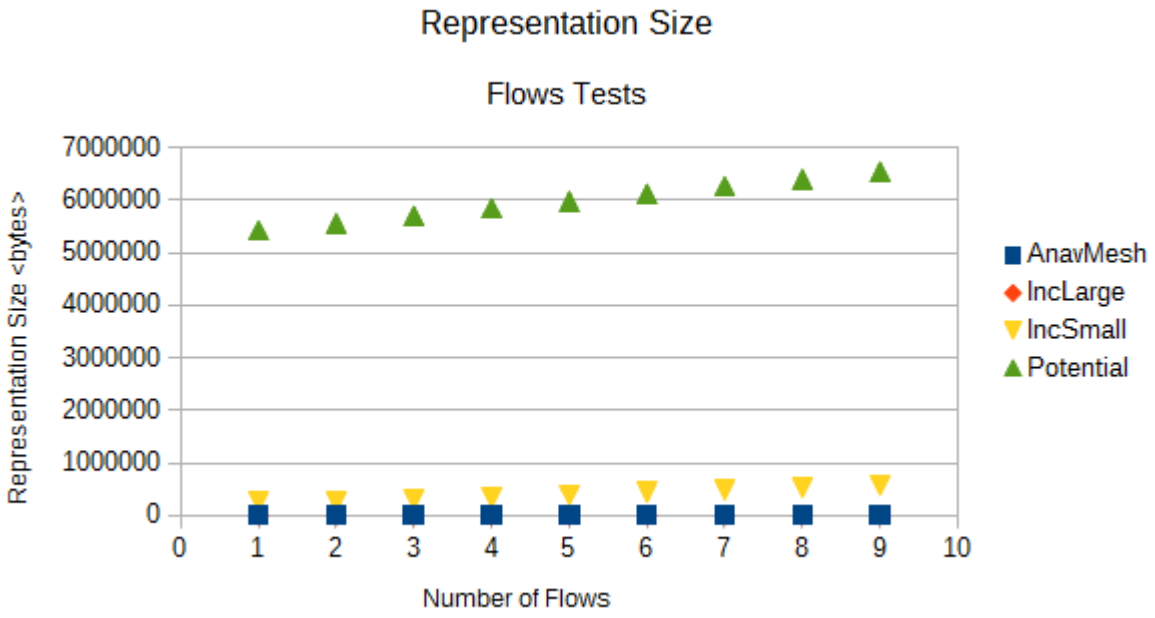


Figure 4.4: Scaling with Number of Flows - Representation Size

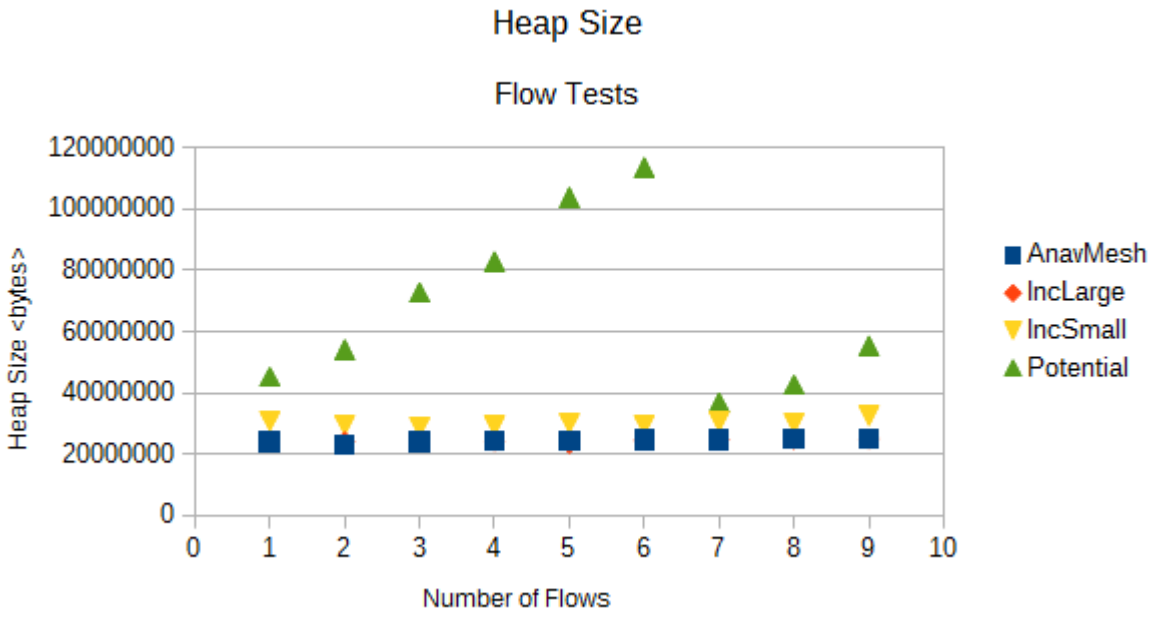


Figure 4.5: Scaling with Number of Flows - Heap Size

Flows	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
AnavMesh	235	284	324	303	208	344	262	291	188
IncLarge	217	230	233	196	174	274	228	238	297
IncSmall	3009	3070	3525	3554	3609	3845	4142	4901	4710
Potential	238	324	370	424	586	519	680	635	739

Table 4.6: Flow Test: Creation Time <ms>

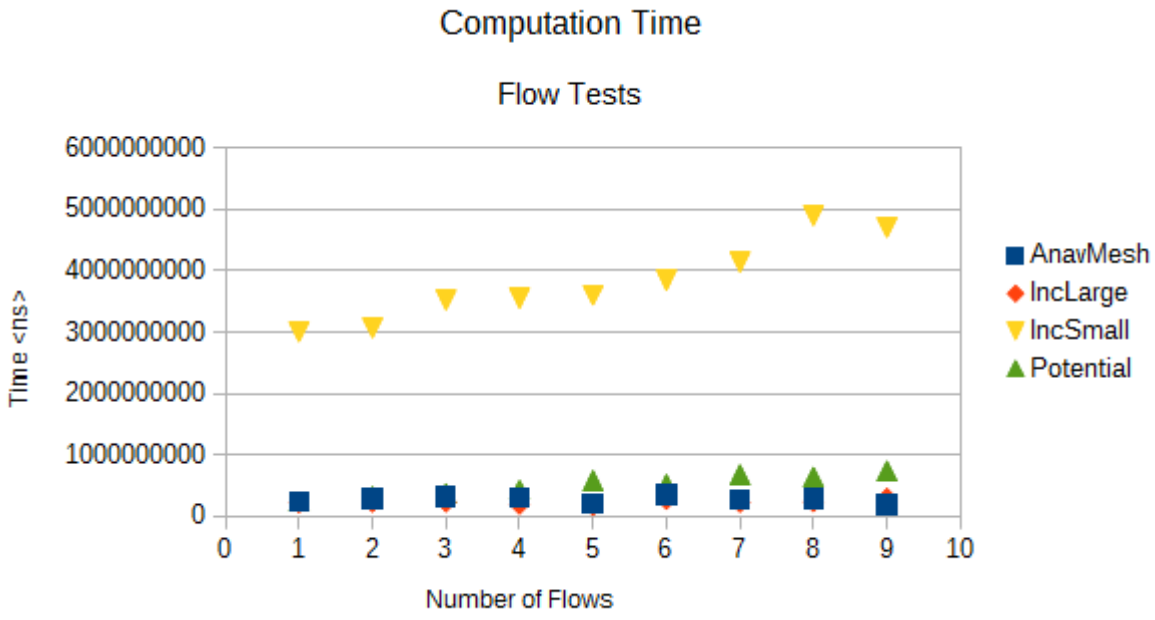


Figure 4.6: Scaling with Number of Flows - Computation Time

Obstacle Rows	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
AnavMesh	10	25	40	23	48	63	78	86	101
IncLarge	26	74	122	29	146	193	241	265	313
IncSmall	403	540	563	304	423	495	453	472	487
Potential	10805	10805	10805	10805	10805	10805	10805	10805	10805

Table 4.7: Obstacle Tests: Representation Size <kB>

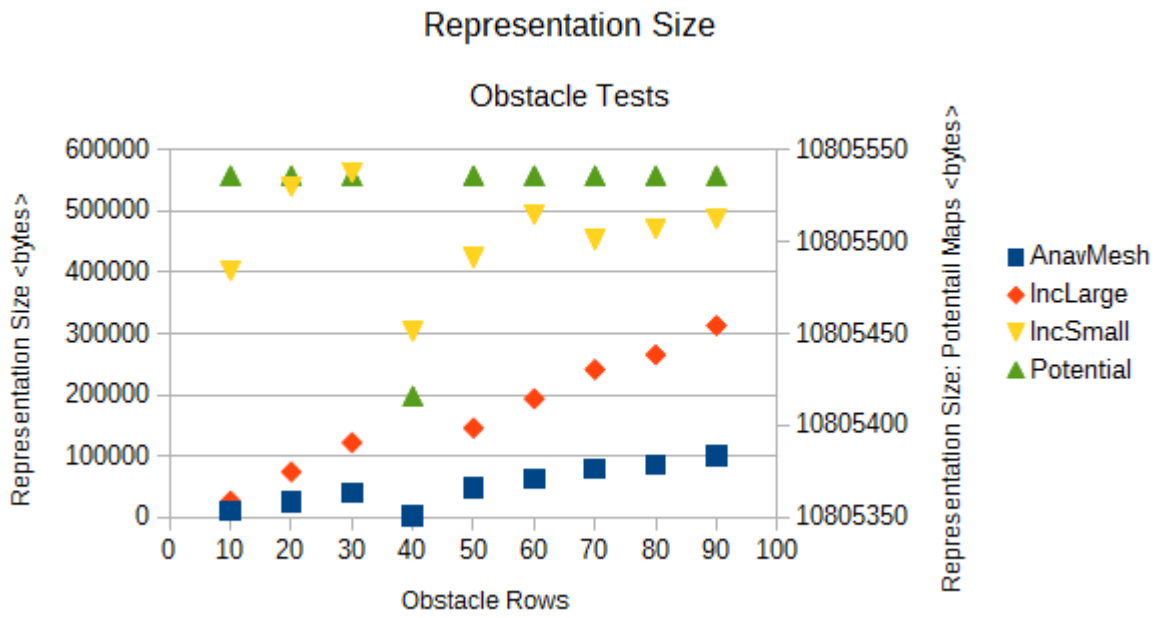


Figure 4.7: Scaling with Space Complexity - Representation Size

From the Table 4.1.2 and Figure 4.6 we can conclude that the IncrementalDelaunay with small triangles is again the most time consuming method employed. The linear trend of some of the methods is presumably caused by the potential representation, as in the previous case.

In conclusion, the memory requirements of the new approaches do weakly grow with number of flows. However, the potential map trend is generally much steeper. In the time measurement, the AnavMesh proved to be superior, but it was observed that IncrementalDelaunay result highly depends on its parametrization.

4.1.3 Obstacle Tests

The obstacle tests illustrate dependence of the used approaches with space complexity. There are 9 scenarios. The scenarios are equally sized. In each scenario, there is a number of cell rows where obstacles can be generated. The obstacles are 2x2 squares separated by 2 cell wide empty space. The scenarios range from 10 rows (i.e., about two rows of obstacles) to 90 rows with potential obstacle placement.

From Table 4.1.2 and Figue 4.7 we can draw several conclusions. Firstly, the old potential map approach representation, while being larger in absolute values, does not grow with

Obstacle Rows	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
AnavMesh		39	54	29	63	88	56	35	94
IncLarge	31	34	40	28	44	52	63	68	80
IncSmall	36	40	46	35	48	59	67	73	86
Potential	71	65	65	66	66	72	63	66	63

Table 4.8: Obstacle Tests: Heap Size <MB>

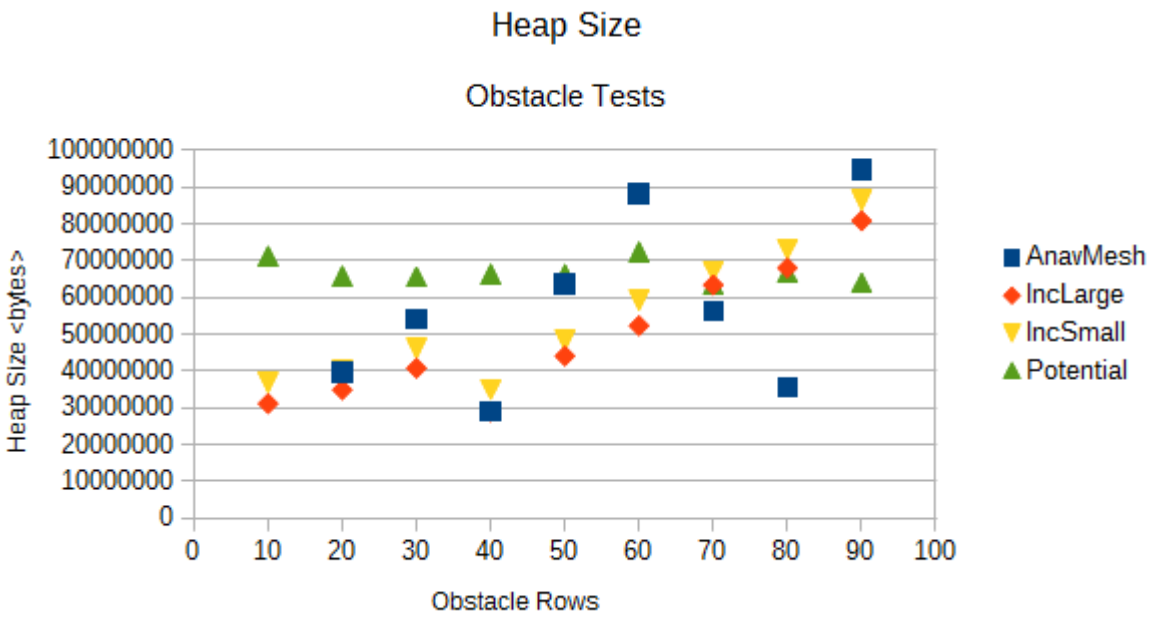


Figure 4.8: Scaling with Space Complexity - Heap Size

the space complexity. Second, the IncrementalDelaunay with small triangles does depend weakly on the space complexity, as the number of adjustments made is smaller thanks to its relative density. Third, the IncrementalDelaunay with large triangles and AnavMesh are highly affected by increased space complexity, as they utilize large open spaces that had to be divided to fit the space. Nonetheless, AnavMesh and IncrementalDelaunay with large triangles still have the lowest memory require of all the approaches. While the representation of the new approaches depends linearly and the potential map representation remains constant, it appears unlikely that the absolute values would be ever equal, as the obstacles would have to be significantly smaller than individual cells to reach such a state.

To no surprise, we can conclude from Table 4.1.2 and Figure 4.8 that heap size of potential map creator is not affected by the space complexity. Heap sizes of the new approaches grow linearly with the space complexity, with AnavMesh being the largest and the most varying. Unlike the previous case of representation size, the heaps the new approaches did reach size higher than the potential maps.

From Table 4.1.2 and Figure 4.9 we can conclude that computation time for the approaches grows linearly with the space complexity, while the potential map computation remains constant. If the absolute values are to be considered, the potential map are the lowest and

Obstacle Rows	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
AnavMesh		39	54	29	63	88	56	35	94
IncLarge	31	34	40	28	44	52	63	68	80
IncSmall	36	40	46	35	48	59	67	73	86
Potential	71	65	65	66	66	72	63	66	63

Table 4.9: Obstacle Tests: Computation Time <ms>

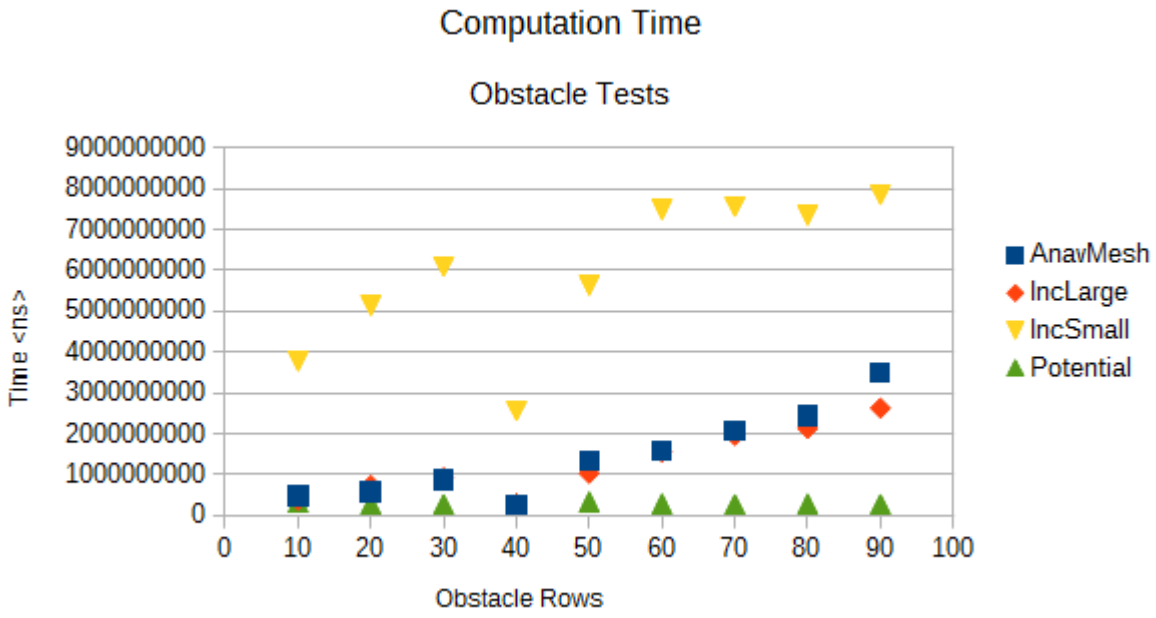


Figure 4.9: Scaling with Space Complexity - Computation Time

	AnavMesh	**IncLarge**	**IncSmall**	**Potential**
Total Ticks Time	292	452	711	17
Total Ticks No	13235	13813	18354	9337
Total Ticks Av	**22132,61**	**32726,94**	**387694,34**	**1850,52**
Unchanged Ticks Time	279	425	418	
Unchanged Ticks No	12952	13382	13257	
Unchanged Ticks Av	**21561,15**	**31795,03**	**31550,85**	
Unchanged Ticks No Percentage	**0,98**	**0,97**	**0,72**	
Unchanged Ticks Time Percentage	**0,95**	**0,94**	**0,06**	
Changed Ticks Time	13	26	6697	
Changed Ticks No	283	431	5097	
Changed Ticks Av	**48286,63**	**61661,54**	**1314002,8**	
Changed Ticks No Percentage	**0,02**	**0,03**	**0,28**	
Changed Ticks Time Percentage	**0,05**	**0,06**	**0,94**	

Table 4.10: Tick Measurements
Time: <ms>

the IncrementalDelaunay with small triangles the highest.

In conclusion, the results of this subsection differ greatly from the previous two, where the new approach proved to be superior. The new approaches trade-off growth of computation time and heap space used with growing space complexity for lower memory requirement of the final representation.

4.1.4 Conclusion

To conclude the memory structure analysis, it appears that the new approaches, mainly the AnavMesh algorithm, perform better when scaled with number of cells or number of flows, where the dependence is either constant or weakly linear. The potential map approach performs better in the computation phase, especially with growing space complexity.

4.2 Tick Measurements

In this section we provide results and analysis time measurements of tick in the simulation. In this context, a tick refers to time required for an agent to compute his next direction based on the navigation structure, i.e., not the time required to compute social forces and execute the movement itself.

The measurement was executed on a map from the empty map series used in size tests. For the IncrementalDelaunay structures, the agents were moving towards centre points of the next mesh polygon. For AnavMesh agents were moving towards the edge of the next mesh polygon.

The results are as in Table 4.2. For the new approaches, there are two tick types:

- Tick: Unchanged - the agent did not move to a new mesh polygon
- Tick: Changed - the agent did move to a new mesh polygon

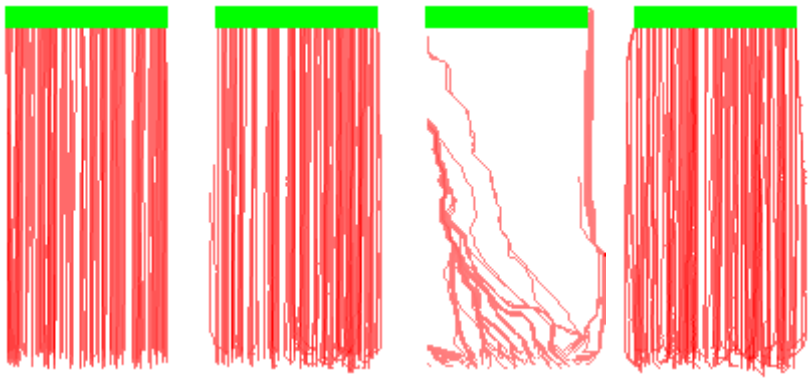


Figure 4.10: Scenario with Obstacles - Potential Map, AnavMesh, Incremental Delaunay with small triangles, IncrementalDelaunay with large triangles

From the results we can conclude that in general, the potential map representation is much faster. The potential map computes each tick about 10 times faster than the new approaches. New representations suffer mostly from slow ticks with change, but even the unchanged ticks are slower than potential representation ticks. Despite using more complex navigation computation, the lower number of polygons causes the AnavMesh to have the fastest ticks. The IncrementalDelaunay with small triangles suffers greatly from large number of mesh polygons, which cause frequent polygon changes and therefore slow the computation.

4.3 Agent Pathing

It is one of the aims of this work to implement a new approach to navigation that, while scaling better with size, retains some similarities in behaviour to potential map model used in the AgentCrowd framework. To this end this sections provides comparison of the implemented approaches to the original potential maps in terms of path planning.

The main method used in this section is comparison using heat maps of agents movement. The heatmaps represent agent density on a cell in the simulation grid. Each time agent steps on a cell, its position is logged. The tone of the heatmap represents the percentage value of the number of agents stepping on a cell, where 100% is the number of agents that stepped on the cell with the highest number.

First, consider Figure 4.10 representing heatmaps of the various approaches used on scenario with several rows of obstacles. The order of approaches is (from left to right): potential map, AnavMesh, IncrementalDelaunay with small triangles, IncrementalDelaunay with large triangles. From brief observation we can conclude that in this scenario both the AnavMesh and IcrementalDelaunay with large triangles behaves similarly as the benchmark potential map. The IncrementalDelaunay with small triangles behaves quite differently. The agents do not move directly to one the exit zone while choosing any of the holes between the obstacles, they rather prefer several distinct paths.

Similar behaviour was observed on several other scenarios. Consider an empty space scenario. The benchmark Figure 4.11 represents the various approaches, listed in the same order as

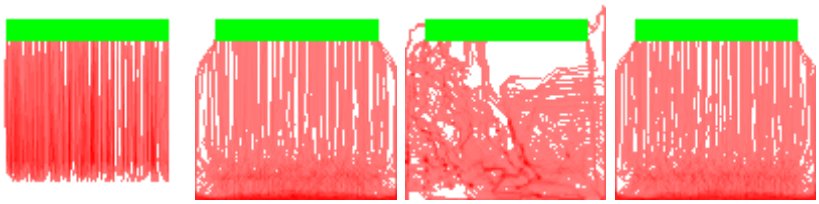


Figure 4.11: Empty Scenario - Potential Map, AnavMesh, Incremental Delaunay with small triangles, IncrementalDelaunay with large triangles

in Figure 4.10. Again, while the IncrementalDelaunay with large triangles and AnavMesh are quite similar to the benchmark, the IncrementalDelaunay with small triangles presents several arbitrary paths.

In general, we can observe some differences between the benchmark scenarios, AnavMesh scenarios and IncrementalDelaunay with large triangles scenarios. While the agents move in similar fashion, the potential map agents are closer to each other, while the other scenarios are little more spread into the space, with Incremental Delaunay being the extreme. However, the difference appears to be insignificant.

To conclude this section, it appear that the AnavMesh does pass the comparison to benchmark potential map scenarios. The IncrementalDelaunay passes the comparison if large triangles are used. However, it should be noted that this could pose a problem, as in different size scenarios/scenarios with different space complexity the parametrization of IncrementalDelaunay would have to change accordingly.

4.4 Evaluation Conclusion

In this section the previous solution, i.e., the potential map approach, was compared to the newly adopted approaches. See Table 4.12 for comparative illustration of the approaches. We found out that in terms scaling with scenario size, the AnavMesh and Incremental Delaunay with large triangles are superior. Incremental Delaunay with small triangles still proved better in terms of memory used, but took much longer to compute. Moreover, the approaches scaled similarly when number of flows was increased.

In terms of scaling with space complexity, the results were not as straightforward. Even though the potential maps did scale well, the absolute size of their representation was still much larger. On the other hand, the computation time of potential maps did not only scaled well, but was also the lowest. The new approaches, especially the Incremental Delaunay with small triangles, were considerably slower.

The potential maps also proved to be the fastest in the simulation itself. The potential map ticks, i.e., the time an agent needs to compute the next move, are about 10 times faster than the ticks of the new approaches. Again, the IncrementalDelaunay with small triangles was the slowest.

Finally, the agent pathing behaviour was examined. Here, the potential maps were considered to be a benchmark approach. Both AnavMesh and Incremental Delaunay with large triangles

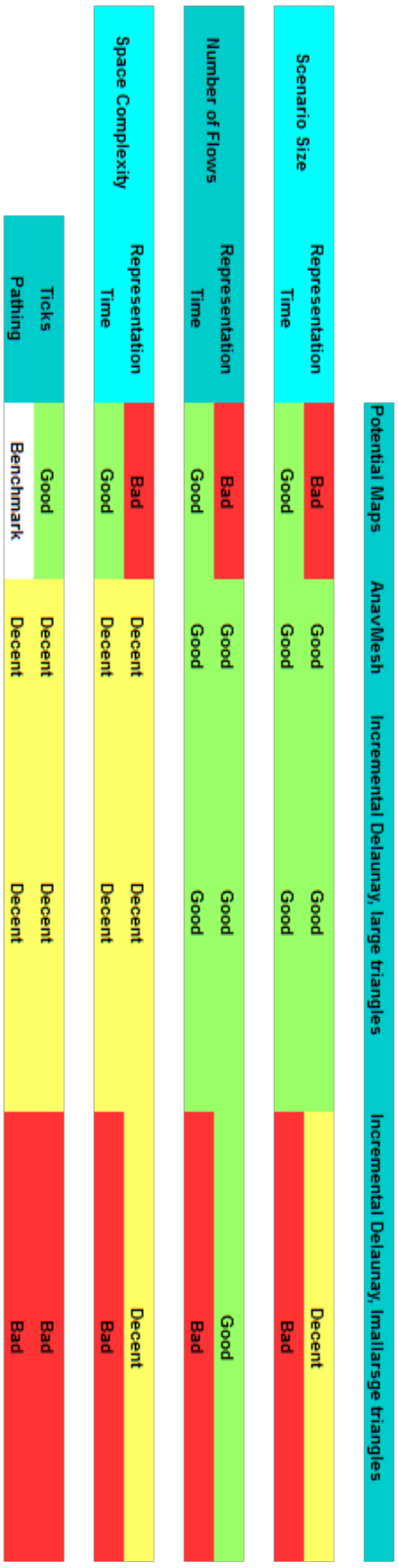


Figure 4.12: Evaluation - Comparative Table

provided agent behaviour similar to the potential maps. The Incremental Delaunay with small triangles, however, seems to encourage agents to take strange, arbitrary paths.

Chapter 5

Conclusion

In scope of this work, we implemented and evaluated two new navigation approaches for the AgentCrowd framework: AnavMesh, a convex polygon mesh algorithm based on work of [Oliva and Pelechano, 2012] and Incremental Delaunay triangulation with additional Steiner points. Both the algorithm proved to be a possible replacement of the potential maps approach used in the AgentCrowd framework. However, we also identified several drawbacks of these approaches.

The potential maps have severe limitations when scaled with scenario size and number flows. Also, their representation in the memory is generally much larger than the representation of the new approaches. On the other hand, the potential maps are relatively fast to compute and can provide the agents with their navigation vectors very quickly.

The AnavMesh proved to be the dominant solution. The approach is superior to both the potential maps and incremental Delaunay triangulation in terms of representation size, regardless of the scaling with scenario size, number of flows, or scenario complexity. It also appears to be the fastest of the new approaches, even though it is still significantly slower than the potential maps.

The incremental Delaunay triangulation proved to be problematic. While it can approach the quality of the AnavMesh when its parametrization is set to large triangle size, the approach is severely limiting when set to small triangle size, as the large number of entities not only takes larger proportion of memory, but also slows down various calculation. Finally, the small triangles may in some cases cause arbitrary pathing of agents.

To draw the final conclusion to this work, we propose the AnavMesh as the new approach of choice for AgentCrowd framework. While the potential maps still have several advantages, mostly in terms of computation time, the AnavMesh approach provides solution that is comparable in terms of speed and superior in terms of memory usage.

Bibliography

[Anglada, 1997] Anglada, M. V. (1997). An improved incremental algorithm for constructing restricted Delaunay triangulations.

[Baker et al., 1988] Baker, B. S., Grosse, E., and Rafferty, C. S. (1988). Nonobtuse triangulation of polygons. *Discrete & Computational Geometry*, 3(2):147–168.

[Cheng et al., 2012] Cheng, S. W., Dey, T. K., and Shewchuk, J. (2012). Delaunay triangulations. In *Delaunay Mesh Generation*, chapter 2 Two-dime.

[Chew, 1989] Chew, L. P. (1989). Constrained Delaunay Triangulations. *Algorithmica*, pages 97–108.

[Curtis et al., 2014] Curtis, S., Best, a., and Manocha, D. (2014). Menge: a modular framework for simulating crowd movement. 39:1–39.

[Eberly, 2002] Eberly, D. (2002). Triangulation by ear clipping.

[Fortune, 1987] Fortune, S. (1987). A Sweepline Algorithm for Voronoi Diagrams. *Algorithmica*, pages 153–174.

[Guibas et al., 1992] Guibas, L. J., Knuth, D. E., and Sharir, M. (1992). Randomized Incremental Construction of Delauna and Voronoi Diagrams. *Algorithmica*, pages 381–413.

[Hardwick, 1997] Hardwick, J. C. (1997). Implementation and Evaluation of an Efficient 2D Parallel Delaunay Triangulation Algorithm.

[Helbing and Molnar, 1995] Helbing, D. and Molnar, P. (1995). Social force model for pedestrian dynamics. *Physical review E*, 51.

[Hertel and Mehlhorn, 1985] Hertel, S. and Mehlhorn, K. (1985). Fast Triangulation of the Plane with Respect to Simple Polygons.

[Hrstka et al., 2013] Hrstka, O., Drchal, J., Micka, V., Štěpanovský, M., Buk, Z., and Čepek, M. (2013). Traffic Flow Modeling: Project Report. Technical report, Agent Technology Center, Department of Compuer Science and Engineering, Czech Technical University in Prague, Prague, Czech Republic.

[Kallmann et al., 2004] Kallmann, M., Bieri, H., and Thalmann, D. (2004). Fully Dynamic Constrained Delaunay Triangulations. pages 1–18.

[Meisters, 1975] Meisters, G. (1975). Polygons Have Ears. *The American Mathematical Monthly*, 82(6):648–651.

[Oliva and Pelechano, 2012] Oliva, R. and Pelechano, N. (2012). Automatic Generation of Suboptimal NavMeshes.

[Ruppertt, 1993] Ruppertt, J. (1993). A New and Simple Algorithm for Quality 2-Dimensional Mesh Generation *.

[Schadschneider, 2002] Schadschneider, A. (2002). Cellular automaton approach to pedestrian dynamics -theory. *Pedestrian and Evacuation Dynamics*, page 75–86.

[Snook, 2000] Snook, G. (2000). Simplified 3d movement and pathfinding using navigation meshes. In DeLoura, M., editor, *Game Programming Gems*, pages 288–304. Charles River Media.

[Treuille and Cooper, 2006] Treuille, A. and Cooper, S Popovic, Z. (2006). Continuum crowds. *ACM Transactions on Graphics*, 25.

[Üngör, 2009] Üngör, A. (2009). Computational Geometry : Theory and Applications Off-centers : A new type of Steiner points for computing size-optimal quality-guaranteed Delaunay triangulations. *Computational Geometry: Theory and Applications*, 42(2):109–118.

[Zhou et al., 2010] Zhou, S., Chen, D., and Cai, W. (2010). Crowd modeling and simulation technologies, acm, transactions on modeling and computer simulation. 20.

Appendix A

Attachments

The attachment disc contains:

- .pdf file of this text
- AgentCrowd project
- scenarios for the AgentCrowd project
- readme.txt concerning the AgentCrowd project