Problem description

By representing a map of a physical location as a domain on the complex number-plane, conformal mappings can be used to simplify the geometry of the domain. This could be used to simplify pathfinding. This master assignment aims to investigate the possibility of using conformal mappings to optimize pathfinding problems in the 2D-plane. To achieve this, the following tasks are proposed:

- Do a literature survey on state-of-the-art pathfinding algorithms
- Do a literature survey on conformal mappings and how to create them
- Propose an algorithm that utilize conformal mappings to do pathfinding
- Investigate the properties of conformal mappings from a pathfinding perspective
- Investigate the possibility of using conformal mappings to find shortest paths
- Conduct experiments on different maps and compare the conformal pathfinding algorithm to state-of-the-art algorithms

Abstract

In this thesis, conformal mappings are used to solve pathfinding problems. By conformally mapping the 2D plane to simpler representations, the cost of pathfinding is reduced. The main benefits of this procedure is that the full path is not required to be computed to guarantee completeness, which means that the path can be computed in real-time. This means that the computational cost for calculating the path can be spread out in time, which increase the responsiveness of the pathfinder. Another benefit of this approach is that because the entire path is not computed ahead of time, there are less wasted computations if the agent changes destinations prematurely.

An algorithm is proposed for finding one or more conformal maps that are well-suited for pathfinding. Various map topoligies with, -and without obstacles are considered.

Practical experiments are conducted in Unity3D, a commercial game-engine, and in MATLAB, to demonstrate and benchmark the algorithm. The source-code is written in C# and can be executed in real-time.

Sammendrag

I denne oppgaven er konforme avbildninger er brukt for å løse stifinningsproblemer. Ved å konformt transformere 2D planet til en enklere representasjon, blir kostnaden av å gjennomføre stifinning redusert. Hovedfordelen av prosedyren er at hele stien ikke trenger å beregnes for å garantere at den er korrekt. Dette medfører at stien kan beregnes i sanntid. Siden ikke hele stien må beregnes før agenten kan agere, er responstiden til den konforme stifinningsalgoritmen raskere enn hvis den måtte bergnet hele stien. En annen fordel med konform stifinning er at ingen beregninger er bortkastet på å generere en sti som ikke vil bli fulgt, i tilfeller der agenten endrer destinasjon før målet er nådd.

En algoritme blir her foreslått for å finne konforme avbildninger som er velegnet for stifinning. Kart med ulike topologier, med og uten hindere, er vurdert.

Praktiske eksperimenter blir utført i Unity3D, som er en kommersiell spillmotor og i MATLAB, for å demonstrere og sette mål på algoritmen. Kildekoden er skrevet i C# og kan bli kjørt i sanntid.

Preface

The work presented in this thesis has been carried out at the Department of Engineering Cybernetics at The Norwegian University of Science and Technology (NTNU), under the supervision of Professor Morten Dinhoff Pedersen who proposed the project.

While writing this thesis, I have also been lead-programmer on upcoming online multiplayer real-time strategy game. The game will be hosted on dedicated servers. The interest in this project came from the possibly huge economic savings associated with an online algorithm for solving pathfinding problems as opposed to an algorithm that must compute entire paths ahead of time.

The research and development of this algorithm will continue after this thesis with the ambition of producing a commercial application.

I have independently implemented all parts of the conformal pathfinding algorithm that is used for testing. The conformal pathfinding algorithm was written in C# and simulated with with the Unity engine developed by Unity Technologies. The built-in pathfinding algorithm in Unity was used as reference when benchmarking the conformal pathfinder. For researching the geometric properties of conformal maps, I have used the Schwarz-Christoffel Toolbox in MATLAB developed by Tobin A. Driscoll [1].

Contents

Problem description **i**

Abstract **ii**

Sammendrag **iii**

Preface **iv**

Table of Contents **vii**

List of Figures **xi**

1 Introduction **1**

1.1 Motivation 1

1.2 Problem description 2

1.3 Outline 4

2 Theory **5**

2.1 Pathfinding 5

2.1.1 The initialization stage 5

2.1.2 The search stage 6

2.1.3 The A* algorithm 7

2.2 Conformal mappings 8

2.2.1 Riemanns' mapping theorem 8

2.2.2 Mapping to simple domains 8

2.2.3 Computing maps 9

2.3 Considerations of conformal pathfinding 14

2.3.1 Bend-compression 14

2.3.2 Path configuration 15

2.3.3 Shortest-path 16

2.3.4 Spatial dilation 16

2.4 Discrete conformal mappings 17

2.4.1 The circle packing algorithm 17

2.5 Flattening algorithm 20

2.6 A* algorithm with circular obstacles 24

3 The conformal pathfinding algorithm **27**

3.1 Multistage pathfinding 29

3.1.1 Macro-scale pathfinding 30

3.1.2 Micro-scale pathfinding 30

4 Experiments **33**

4.1 Spatial warping 34

4.1.1 Protrusions into the domain 34

4.1.2 Retrusions out of the domain 37

4.1.3 Elongated and narrow domains 38

4.2 Simply-connected domains 41
4.3 Pathfinding with many obstacles 43
4.4 Entangled obstacles 45
4.5 Infeasible maps 47

5 Discussion 51

5.1 Further work 53
5.1.1 Using continuous functions to generate the conformal graphs . . . 53
5.1.2 Modifying graphs in real-time 53
5.1.3 More accurate modifications to support shortest paths 54
5.1.4 Composite maps 54

Bibliography 54

Appendix A 57

List of Figures

1.1 By simplifying the domain, computing paths between two destinations can be as easy as drawing a straight line. Original domain on the left, virtual domain on the right. 2

1.2 Convex mappings remove dead ends, which significantly simplify the pathfinding problem 3

2.1 A polygon with exterior angles 10

2.2 Three neighbouring circles' radius can be used to compute the angles of the triangle they form. 12

2.3 Circle-packed rasterization and resulting mapping 14

2.4 The quantization problem. If the step-length when quantizing the continuous path is too big, the resulting poly-line can intersect the boundary of the domain. 15

2.5 With two obstacles, there are 4 different configurations of paths between endpoints 16

2.6 Example of a circle-packing rasterization 18

2.7 The even-odd rule. 19

2.8 Hexagonal grid indices. In this simple graph, the center hexagon has six neighbours and each perimeter node has three neighbours. 19

2.9 The position of v_k can be derived from its radius and its neighbours radius and position 20

2.10 Premature termination of the discrete Ricci-flow algorithm can lead to arbitrary graphs 21

2.11 Clockwise nodes connects and and counter-clockwise node connects . . . 24

2.12 External bi-tangents 25

2.13 Internal bi-tangents 25

2.14 bi-tangents to a point 26

3.1 A map that represent the movable area for a navigation agent 27

3.2 Circle packing of map and the results of the Ricci flow algorithm 28

3.3 Conformal mapping between original and virtual domain 28

3.4 The circular pathfinding graph that is extracted 29

3.5 Mapping between original and virtual coordinates through a barycentric transform. Virtual map on the left and original map on the right. 29

3.6 The circular path in the virtual domain and its original dual. The green and blue lines represent the extended graph that includes the start and endpoint as nodes. The red line represents the path. The reason for it not hugging the circles is because a radial padding has been added to avoid the path visiting dead ends. 30

3.7 Micro-scale pathfinding can be done to find the shortest path online . . . 30

3.8 Path is transformed by barycentric mapping based on where the virtual path intersect with triangle edges 31

4.1 Domains with sharp protrusions. The left domain has a sharper protrusion than the right. The blue curve in the two upper rows represent the shortest path between the two endpoints, the red line shows the virtual shortest path. 35

4.2 Domains with sharp protrusions. The left domain has a sharper protrusion than the right. The blue curve in the two upper rows represent the shortest path between the two endpoints, the red line shows the virtual shortest path. 36

4.3 Domain with sharp retrusion. 37

4.4 Retrusions out of the domain results in clusters that will ”suck in” the path if the virtual path goes through the cluster. 37

4.5 Example of a domain with a even distribution. The heatmap in (c) and (d) show the local node-density of the virtual domain. The yellow corners have a high density of nodes, while the dark blue corners have a very low density, but overall the density is evenly distributed throughout the domain. 39

4.6 Example of a domain with a uneven distribution. The heatmap in (c) and (d) show the local node-density of the virtual domain. In this instance, the spatial warping is severe, and large regions of the original domain is represented by very small virtual regions, making pathfinding difficult. . 40

4.7 Comparison of pathfinders in a simply-connected domain 41

4.8 A tessellation of non-convex obstacles 43

4.9 The virtual domain of the domain in Fig. 4.8 44

4.10 Entangled obstacles doesn’t look entangled in the virtual domain 45

4.11 The zig-zag domain 48

4.12 Domain with nested obstacles. The narrow openings into the ”next level” causes extreme warping 49

4.13 Nested obstacles result in extreme warping and breaks the conformal mapper 50

5.1 Real-time games with many units on large maps with many obstacles can benefit significantly from conformal pathfinding techniques 52

Chapter 1

Introduction

1.1 Motivation

Pathfinding is the task of finding a way to get from one point to another. In this project, a navigational agent (agent) is requesting a set of way-points (a path) that will take the agent to a desired destination. To do this, the pathfinder is provided with a map of the domain, containing the boundary of the region and possible obstacles in the region, the coordinates of current location of the agent, and coordinates for the desired destination.

There are two possible interpretations of pathfinding. The first involves computing the entire path ahead of time. This is especially useful when the problem involves multiple agent coordination. In the second interpretation the agent only needs to know what direction to move in at any time. In this project, the second interpretation will be in focus.

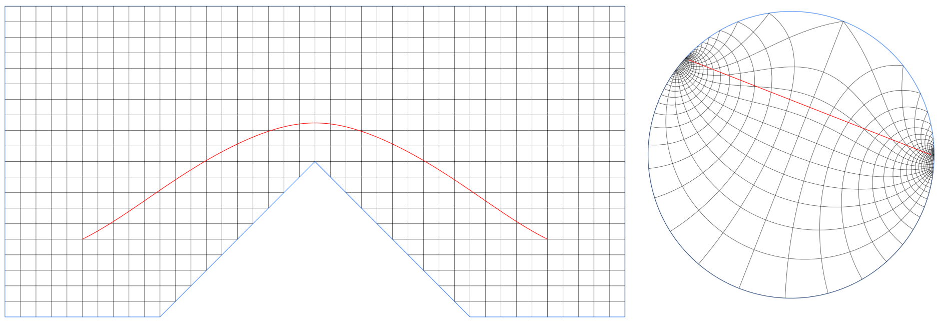
The most common pathfinding algorithms today are search-based. That means that the algorithm starts out at one point, traverses a graph and terminates when a valid path is found. If the geometry of the map is complicated, this can result in the algorithm exploring many dead ends. Convex domains do not have dead ends. By transforming a non-convex domain to a convex domain, pathfinding can be done more efficiently.

The most common pathfinding algorithm today is the A* algorithm, a breadth-first search algorithm that maintains a priority queue for its frontier, based on the current length of the path and the estimated length to the target. The problem with A* and other search-based algorithms is that the navigational agent cannot move before the entire path is calculated. In fact, before the path is found, the A* algorithm maintains many potential paths, but cannot guarantee that any of them will end up at the destination because many of them might be dead ends.

In most cases, it is more than one path that connects two points. In the continuous case, there might be infinitely many. Some paths are better than others. What constitutes a good path varies based on use-case. A good path can for instance be a short path, a quick path or a path with low fuel consumption. Other concerns are the practical costs associated with computing the path such as response time and computational demand. These practical costs can restrict the quality with respect to the aforementioned concerns.

There exists many pathfinding algorithms that have different strengths with respect to the parameters given above. This project investigates if there is a way to geometrically ”re-arrange” a map that simplifies pathfinding. Pathfinding is only needed when there are obstacles that intersect the direct line between two end-points, in other words; when the domain is non-convex. Conformal mappings will be used to transform arbitrary domains to virtual domains where pathfinding is simpler. An example of this is shown in Figure 1.1

Figure 1.1 By simplifying the domain, computing paths between two destinations can be as easy as drawing a straight line. Original domain on the left, virtual domain on the right.

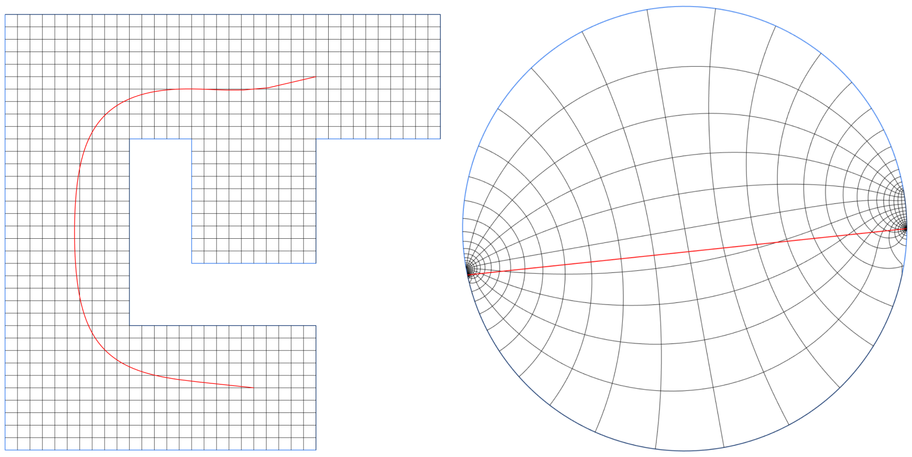


1.2 Problem description

In a convex region, the entire path to the destination does not need to be calculated ahead of time because the convex property guarantees that simply moving directly towards the target will get the agent all the way there. If the distance to the destination is long, and the region has a complicated geometry, not having to calculate the entire path ahead of time is a significant benefit. This will speed up the response time of the agent and potentially save significant computational costs. If the target destination can change before the agent has arrived, all computation spent on calculating the remaining path is wasted, so not having to do that is very valuable. Figure 1.2 examplifies of how convex mappings essentially removes dead ends.

To optimize pathfinding by simplifying the geometry of maps, it must also be possible to map the virtual domain back to the original one. We also need a path in the virtual domain to correspond to a path in the original, which means the mapping cannot “cut” the path. More generally we can say that we want the virtual domain to be a homeomorphism of the

Figure 1.2 Convex mappings remove dead ends, which significantly simplify the pathfinding problem



original. A mapping that satisfies these requirements are called a conformal mapping.

The essence of the algorithm can be summarized in three steps:

1. Map two endpoints to the virtual domain
2. Find a path between the two points in the virtual domain
3. Map the path back to the original domain

The Riemann mapping theorem states that if $\mathcal{D}$ is a non-empty simply connected open subset of the complex number-plane $\mathbb{C}$, then there exists a biholomorphic mapping from $\mathcal{D}$ to the open unit disk $\mathcal{U}$.

$$\mathcal{U} = \{z \in \mathbb{C} : |z| < 1\} \tag{1.1}$$

We will utilize this property to construct a conformal mapping between a non-convex domain $\mathcal{D}$ and a convex domain $\mathcal{D}^*$. Given two points in the original domain, A and B, call the images of A and B in $\mathcal{D}^*$ A^* and B^*. We map the straight line between A^* and B^* back into $\mathcal{D}$ and get a path that connects them. With this approach, we have removed the necessity of search from pathfinding. In D^* a path between A^* and B^* is a simple linear interpolation. This means in practice that it is not required to compute the entire path to verify that it reaches the destination. We can at any time compute the next step. This means that if the target destination changes, we have spent no time computing a path that will not be used. Furthermore, it allows us to ”spread out” the computation of the path in time, which is less computationally demanding.

Using complex analysis for pathfinding can be divided into the following stages. Given a 2D map, a starting point A and a destination B

1. Generate a polygon $\mathcal{P} = \{p_1, p_2, ..., p_N\}$ that encloses the movable area of the map.
2. Compute a conformal mapping $\mathcal{F} : \mathcal{P} \to \mathcal{P}^*$
3. Compute $A^* = \mathcal{F}(A)$ and $B^* = \mathcal{F}(B)$
4. Compute $z^* = \alpha A^* + (1 - \alpha B*), 0 < \alpha < 1$
5. Compute $z = \mathcal{F}^{-1}(z^*)$

Then moving from A to z will bring you closer towards B. As long as B does not change, only step 4 and 5 needs to be repeated with increasing values of α to reach B. The increment of α for each iteration should be as high as possible without generating a path that intersects with any boundaries. This problem is discussed in 2.3.1

1.3 Outline

First the most common pathfinding algorithms today will be introduced. The main downside to these algorithms are explained and conformal pathfinding is shown as a possible solution to that problem. Then the theory behind conformal mappings and how they can be used to solve pathfinding is investigated. The proposed conformal pathfinding algorithm is compared to a commercial pathfinding application and then the results are discussed. The thesis ends with a proposal for further work.

Chapter 2

Theory

2.1 Pathfinding

All pathfinding problems can be divided into two main stages; the initialization stage and the searching stage. The initialization stage involves constructing an environment that can be used to solve pathfinding problems. The searching stage is when agents are requesting paths from the pathfinder for the environment constructed in the initialization stage.

2.1.1 The initialization stage

The initialization stage involves processing a map to produce a solvable pathfinding problem. The process and the resulting environment varies on the pathfinding algorithm chosen. For graph-based algorithms, this stage involves building a graph with nodes that represents physical regions of the map, and edges that connects these regions. With potential fields, several potential fields are built that attracts the pathfinder towards the target destination and makes obstacles repulsive, such that gradient descent will lead the agent towards the destination.

In graph-based pathfinding approaches, a common technique is to construct a navigational mesh (navmesh). A navmesh is a triangular mesh with a polygonal perimeter. If the total number of vertices in the polygon is N_{total} and h is the number of boundaries , then the navmesh can be made of $N_{total} + 2h - 2$ triangles. This means that the graph will have at least $N_{total} + 2h - 2$ vertices and at least $N_{total} - 3$ edges. The performance of any graph-based search algorithms are mainly determined by these two metrics.

The main challenge in the initialization staging is building a problem that enables the search stage to be performed as efficiently as possible. For static environments, the initialization stage only needs to be performed once. In these cases, approaches that invest heavily in the initialization stage is favourable, as this usually means smaller expenses for computing paths in the second stage. One such algorithm is the Floyd-Warshall algorithm which computes the shortest paths between all pairs of points[2, p. 693]. The computational complexity of Floyd-Warshall on a graph $G = \{V, E\}$ where V are the vertices and E are the edges that connects the vertices, is $\Theta(V^3)$ and assuming a sparse representation, requires storing V^2 indices. With this algorithm we can, given the agents location A, and a destination B, look-up the next node to go to in constant time. This allows us to change paths frequently without having to compute a new path every time. The main downside with this approach is that the memory required to store the look-up table grows quadratically with the size of the graph, and changes to the graph requires a full re-computation of the table, which depending on the size of the graph, can prohibit execution in real-time.

2.1.2 The search stage

The search stage is finding a continuous path between the starting point and a target destination.

While there does exist pathfinding algorithms for finding paths in continuous domains, the focus of this project will be on discrete domains. If needed, a discrete path can be converted to a continuous one by use of various interpolation techniques.[3, p.266]

To find paths we use path-traversal algorithms. Given two positions, the algorithm attempts to traverse the navmesh from the vertex closest to one of the positions to the vertex closest to the other position. A nice property of graph-traversal algorithms is that they work well with arbitrarily many obstacles.

In most cases a pathfinding graph will be reused. Every time a new path is required between two endpoints, a new search operation is required. This means that having a cost-effective search stage usually is preferable to saving costs on the initialization stage.

2.1.3 The A* algorithm

The A* algorithm is a graph-based shortest path search algorithm that is widely used because of its optimality, efficiency and applicability to many problems.

The first version of pathfinding algorithm was proposed in the IEEE Transactions on Systems Science and Cybernetics by Hart in 1968 as a heuristic modification to speed up Dijkstra's algorithm [4]. It was later shown that no algorithm could find an optimal path by expanding fewer nodes than the A* algorithm [5]. The only way to outperform A* is to add additional layers of functionality into the algorithm by means of pre-processing for short-cuts or other artifacts that can aid the pathfinding.

The algorithm works by expanding a frontier of paths until the destination point is reached and the shortest from the start to the destination is found. The frontier is a priority-queue of the next edges to explore sorted by some cost function. The cost function is the combination of two values. One is the current cost of the path this far, the second is the estimated cost to reach the target from the current destination. The second value, usually referred to as the heuristic, must always have an estimate that is optimistic, i.e. the estimate must be no worse than the best possible outcome.

The worst-case computational complexity of A* on a graph $G : \{V, E\}$ where V is the vertices and E is the edges, is $O(|V| + |E|)$ and means that in the worst case, the optimal path is the last path explored [6, p.95]. However, the average performance is usually better.

The A* algorithm also allows for adding variable weights to edges. This can be used to simulate the quality of the terrain, allowing the pathfinder to account for this. E.g. it might be preferable to take a slightly longer path on a paved road instead of a "short-cut" through a swamp.

It is no way to know if any path in the frontier will reach the destination before the destination is reached through the search. This will be called the *termination-criterion*. The way A* finds the optimal path is through consistently maintaining optimal paths in its frontier, the optimal path to the target destination is found when the next node to be explored in the frontier is the target destination. This implies that if the shortest path between two points in the graph are a million nodes long, to find the path a minimum of a million computations are required, and because of the breadth-first nature of the algorithm, in most cases more computations than that.

There are many variants of A* using different graph representations, heuristics, and frontiers. Some of them compromise between the optimality of the path and computational efficiency. For very large graphs, shortcut edges can be inserted into the graph between distant nodes. This shortcut edge has a pre-computed optimal path between the two nodes and can be used to shorten the computational requirements significantly. This approach will make finding the shortest path impossible, but in many cases the trade-off between computational costs and optimality is worth it. In general, to get faster results than from A*, more information is required ahead of time that can be utilized in the search problem.

2.2 Conformal mappings

Conformal mappings are homeomorphic functions between two domains that preserves orientation and angles locally. They are common in complex analysis, where a conformal map of a domain is called an image of the domain, or the virtual domain. For conformal maps, if two curves intersect in the domain with some angle θ, they will intersect with the same angle θ in the image of the domain. In other words, if a path in one domain does not intersect any walls or obstacles; neither will the image of the path. This is the core property that makes conformal maps useful in pathfinding. We can map a domain which is computationally expensive to do pathfinding in, to a domain where pathfinding is easy.

2.2.1 Riemanns' mapping theorem

Riemanns' mapping theorem states that any simply connected open subset of the complex number-plane can conformally be mapped to the interior of the unit disk.

$$D = \{z \in \mathbf{C} : |z| < 1\}$$

A consequence of this theorem is that *any* two simply connected sub-domains of the complex number-plane are homeomorphic, e.g. can be mapped to each-other.

2.2.2 Mapping to simple domains

The implication of Riemanns' mapping theorem is that it is possible for any 2-dimensional domain to be mapped to any other 2-dimensional domain with the same connectivity. For a domain to be simple, all boundaries must have curvatures that has the same signing anywhere. In practice this implies that the region defined by each boundary is convex.

Within a convex region, any two points can be connected by a straight path without intersecting the boundary. This means that pathfinding is easy; the agent can simply move directly towards the target.

All domains considered in this project has one external boundary that limits the size of the graph to a finite number. If there exist no obstacles inside this domain, the graph is simply connected and can be mapped to a convex domain. However, most of the time there will exist obstacles. It is not possible to map domains containing obstacles to a convex domains, but it is possible map them to simpler geometries by mapping the boundaries of obstacles to circular objects, the pathfinding problem is simplified.

Dead-ends

Graph-based pathfinding has a termination condition. That means that the search for a path between the two endpoints has to complete before the agent can start to move. There is simply no way for the algorithm to know ahead of time if any path will actually lead the agent towards the target destination. A* pathfinding uses a greedy heuristic to optimistically refine the priority queue. In certain cases, this will lead to the algorithm spending significant time exploring dead-ends.

Dead-ends are a consequence of having boundaries with different signing to their curvatures at different points. If the external boundary has the same signing for its curvature all around, the boundary is convex and at no point will a path within the region intersect the boundary. For internal boundaries, the same property means that the agent at no point will have to backtrack if following the boundary.

With the help of conformal mappings, it is possible to create a mapping of any n-connected region to any other n-connected region. In this project we are interested in mapping any n-connected region into a n-connected region where all boundaries are convex.

Any point in the virtual domain corresponds to exactly one point in the original domain. And the local ordering of points will be unchanged. In other words; any pair of points that were neighbours in the original domain, are also neighbours in the virtual domain. The relative spacing between points will be different.

The properties mentioned above is important because they mean that any continuous path in the original domain, will have a bijective correspondence with a continuous path in the virtual domain.

This means that a complicated geometry in the original domain, like that of a maze, can be transformed into a virtual domain that is a perfect circle. Any points within the maze will be within the boundary of the circle, and since a circle is a convex domain, we know that any path between two points inside the circle can be a straight line. And because of the properties mentioned above, we know that this straight line will correspond to a valid path between the same two points in the labyrinth, effectively solving the maze without having to perform any search to connect the end-points.

2.2.3 Computing maps

There has been done a lot of research on conformal maps [7, 8, 9, 10, 1, 11, 12, 13]. While Riemanns' mapping theorem ensures the existence of a conformal map, it is difficult to find such mappings analytically. A recent paper suggests it is possible to accurately approximate conformal maps with rational functions [14]. Most of the time, numerical approximations are required to find the mappings. The methods used have different computational costs and accuracy.

The Schwarz-Christoffel mapping

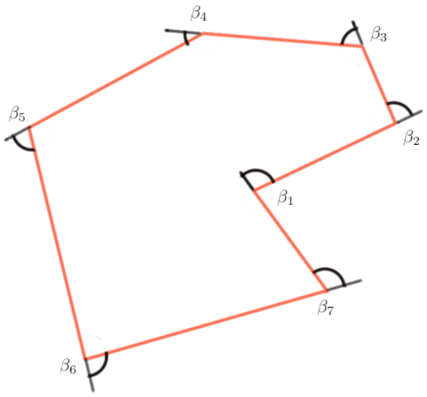
One of the most commonly used conformal mapping schemes today are the Schwarz-Christoffel mappings. They are named after Elwin Bruno Christoffel and Hermann Amandus Schwarz. While there exists several variations of them, the core idea is that there exist functions based on the geometry of the domain and image boundaries, that defines mappings between them. The core concept of the mapping is to take any domain in the complex plane represented by polygons, and map them to some simpler, virtual domain. The virtual domain can be any geometry with the same level of connectedness, and common domain include the upper half-plane, the unit disk, an infinite strip, a rectangle and so on. In this project, the circle is chosen as the virtual domain because this allows for arbitrarily many vertices without the pre-vertices going off into infinity which is a possibility in the upper half-plane case.

A numerical solver of the mapping was proposed by Lloyd Trefethen in 1979 [10]. Initially the mapping was only used with simply-connected domains, but in later years the algorithm has been extended to include multiply connected domains as well [15, 16].

Schwarz-Christoffel mappings between two domains can be found by knowing the perimeter of both domains. Let perimeter of some domain $\mathcal{D}$ be a polygon $\partial\mathcal{D} = \{z_1, z_2, ..., z_N\}$ where the points $z_i, i = \{1, 2, ..., N\}$ are vertices in the polygon and are connected by simple curves in a counter-clockwise fashion and z_N is connected to z_1. We call these points *pre-vertices*. Also, let the perimeter of some other domain Ω be a polygon $\partial\Omega = \{w_1, w_2, ..., w_N\}$ in the same fashion as for $\mathcal{D}$ with exterior angles $\{\beta_1, \beta_2, ..., \beta_n\}$ (see Figure 2.1) where

$$\sum \beta_i = 2\pi$$

Figure 2.1 A polygon with exterior angles



Then there exist a conformal map $f : \mathcal{D} \to \Omega$ between the domains such that $w = f(z)$.

$$f(z) = w_c + K \int_0^z \prod_{i=1}^{N} \left(1 - \frac{w}{w_i}\right)^{-\beta_i} \mathrm{d}w, z \in \mathcal{D}, w \in \Omega$$

Usually, the pre-vertices are not known in advance. The problem of finding pre-vertices is called the *parameter problem*. In addition to knowing the pre-vertices, a conformal center must also be chosen. The parameter problem is solved as an optimization problem, where the parameters given initial guesses, and then approximated through gradient descent. The inverse Schwarz-Christoffel mapping is the mapping from the original domain to the virtual. This can be solved the same way the forward mapping is solved and can be derived directly from the forward mapping.

The Schwarz-Christoffel mapping cannot usually be solved analytically. For any point in the virtual domain, the corresponding point in the original domain is found through numerically solving the function. The Gauss-Jacobi quadrature is very well suited numerical scheme for solving the integral because it is stable close to singularities. [10]

Notice that for the Schwarz-Christoffel mapping, a numerical algorithm including all vertices in the domain is executed. For the purpose of pathfinding, these mappings must be computed frequently and throughout the entire lifetime of the agent, and often several points has to be computed at a time. The benefit of the Schwarz-Christoffel mapping is that is compact and very precise, but at the downside it is quite expensive to use it.

Consider a case where the agent must navigate around a corner in an otherwise large and complicated map. With conventional A* pathfinding, a breath-first search will be performed. Since the endpoint is very close and only around a single obstacle, very few iterations are required to reach the endpoint. For a conformal pathfinder using the Schwarz-Christoffel scheme, first both endpoints must be translated to virtual coordinates. Already here two operations involving the entire boundary of the domain are performed, which would be much more expensive than the A* approach and probably slower as well.

To make the conformal pathfinder more efficient, it is possible to quantize the Schwarz-Christoffel formula into a graph which can act as a look-up table for pathfinder. This will circumvents the problem of having to compute the mapping every time, and allows for constant-time mapping between the two domains. The accuracy of the mapping will be significantly reduced, but sufficient for pathfinding purposes. I will come back to this later.

Discrete Ricci-flow

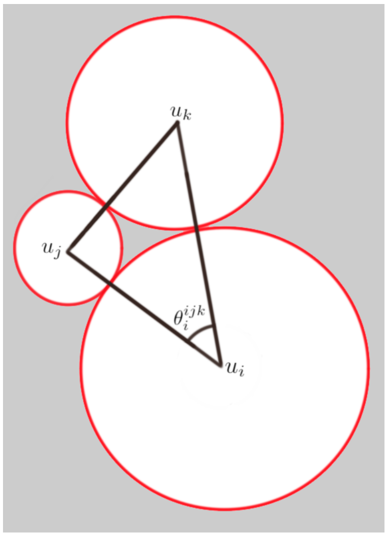
To perform discrete conformal mapping the discrete Ricci-flow algorithm can be used. By simulating the behaviour of heat-flow, the algorithm approximate a conformal mapping. The discrete Ricci-flow algorithm has been used to perform routing problems before. [17], In that case, the goal also was to avoid dead ends by conformal mapping. In their case, the basis was a network of connected wireless sensors where data-packets could get stuck due to dead ends. They used connection information between sensors, prior knowledge about positions and landmark-based triangulation to create the initial graph. The main difference between their use and mine is that their nodes was physical objects that the signal must go through.

Assuming a connected graph that represents a circle-packing, the algorithm works by tuning the radius of each circle while requiring all neighbouring circles to stay neighbours and kiss each other.

Given three neighbouring circles u_i, u_j and u_k that for a triangle, with radii r_i, r_j and r_k. Let $l_{ij} = r_i + r_j$, $l_{ik} = r_i + r_k$ and $l_{jk} = r_j + r_k$ be the side lengths of the triangle. Then the corner angle at u_i is given by

$$\theta_i^{ijk} = acos\left(\frac{l_{ij}^2 + l_{ik}^2 - l_{jk}^2}{2 l_{ij} l_{ik}}\right)$$

Figure 2.2 Three neighbouring circles' radius can be used to compute the angles of the triangle they form.



By summing up the corner angles for all triangles a node is a part of, we get the a measure of the overlap of the neighbours. For an interior node, the angle-sum should be exactly 2π and corresponds to a discrete curvature of 0. For the boundary nodes, we require the curvature they form together to generate circle. We require the target curvature to be $-2\pi/n_\gamma$ for all boundary nodes that belong to the same interior boundary γ, where n_γ is the number of boundary nodes for that boundary. For the exterior boundary, we require the target curvature to be $2\pi/n_\gamma$. The discrete curvatures can be computed as

$$k_i = \begin{cases} \pi - \sum \theta_i^{ijk} & u_i \in \partial M \\ 2\pi - \sum \theta_i^{ijk} & u_i \notin \partial M \end{cases}$$

We denote the target curvature for a node as $\bar{k_i}$ and the difference between the current curvature and the target curvature is thus $\bar{k_i} - k_i$.

Since radii of circles cannot be negative, we don't update the radius directly. Instead, we let nodes have a Ricci energy u_i and let the radius be given as $r_i = e^{u_i}$. For each iteration, we update the energy by $u_i \leftarrow u_i + \delta(\bar{k_i} - k_i)$.

At the last steps of the algorithm, the target curvature for boundary nodes can be computed

Algorithm 1 Discrete Ricci-flow

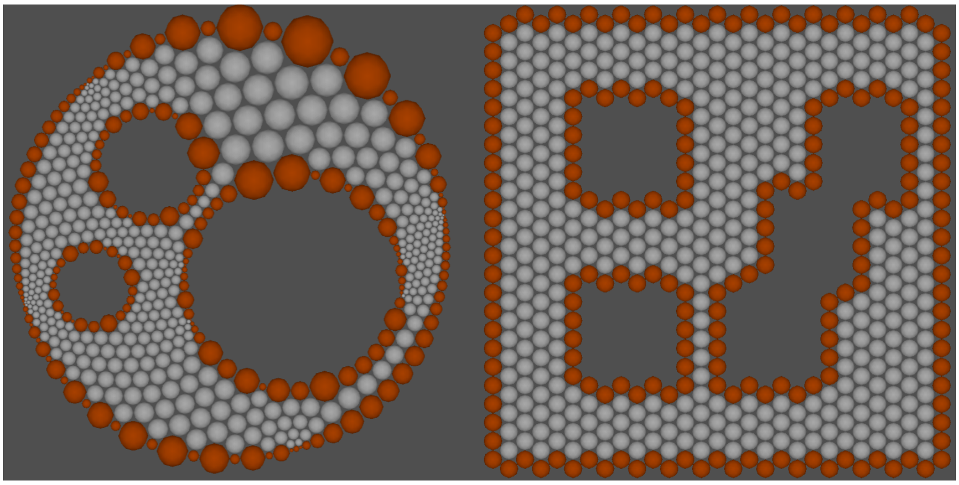
```
Require: Triangular mesh M, target curvature for each vertex $\bar{k}_i$. Error threshold $\epsilon$. Step
    length $\delta$
Ensure: Discrete metric (edge lengths) satisfying the target curvature.
  For all vertex $v_i$, $u_i \Leftarrow 0$
  while $true$ do
    Compute edge length $l_{ij}$ for edge $[v_i, v_j]$
        $$l_{ij} = e^{u_i} + e^{u_j}$$
    Compute corner angle $\theta_i^{ijk}$ in triangle $[v_i, v_j, v_k]$
        $$\theta_i^{ijk} = acos\left(\frac{l_{ij}^2 + l_{ik}^2 - l_{jk}^2}{2l_{ij}l_{ik}}\right)$$
    Compute curvature $k_i$ at $v_i$
        $$k_i = \begin{cases} \pi - \sum \theta_i^{ijk} & u_i \in \partial M \\ 2\pi - \sum \theta_i^{ijk} & u_i \notin \partial M \end{cases}$$
    if $\max |\bar{k}_i - k| < \epsilon$ then
      return The discrete metric $\{l_{ij}\}$
    end if
    Update $u_i$
        $$u_i \Leftarrow u_i + \delta(\bar{k}_i - k)$$
  end while
```

with

$$\bar{k_i} = -2\pi \frac{l_{i,i-1} + l_i}{\sum_M l_j}$$

. which results in a more circular result. This extra computation is computationally expensive and used with care.

Figure 2.3 Circle-packed rasterization and resulting mapping



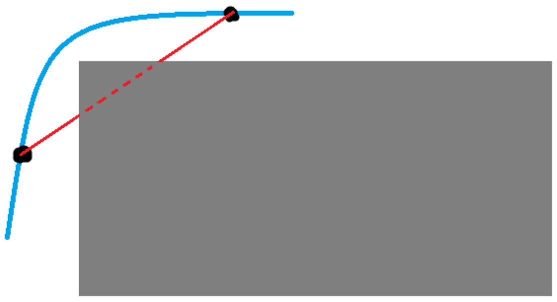
When the position of all nodes are computed, the conformal map is computed. An example of a initial circle packing and the result can bee seen if Figure 2.3. If a shortest path result isn't required, the graph can be used as-is, but because of how the algorithm warps the domain, additional pre-processing is required before the graph can be used to shortest path pathfinding.

2.3 Considerations of conformal pathfinding

2.3.1 Bend-compression

When bending the region to be convex, the spatial density of the map changes. This means that three equidistant points in the original domain will not be equidistant in the virtual domain. This is a problem because it is not possible to follow a curve with infinitesimal precision. It has to be quantized into a set of way-points to be used. To quantize that path is a non-trivial problem. Since the straight line in the virtual domain is curved to avoid the boundary in the original domain, the quantization must have sufficiently high resolution to not intersect with the boundary. We want to find a quantization such that the poly-line generated has the shortest possible length and consists of as few points as possible.

Figure 2.4 The quantization problem. If the step-length when quantizing the continuous path is too big, the resulting poly-line can intersect the boundary of the domain.



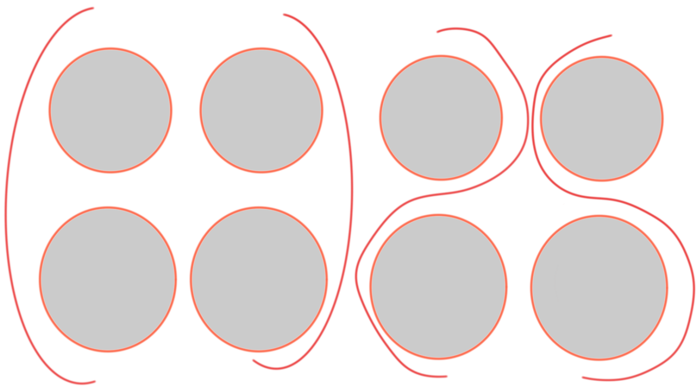
There are several ways to attack the quantization problem. We can attempt to find conformal maps that minimize the variance in spatial dilation, and find mapping procedures that allows for efficient computation of local dilations so that finding good quantization is cost-effective. Another approach is to use discrete variants for conformal mappings.

2.3.2 Path configuration

For every obstacle, the path between two points can move move relative to that obstacle in either a clock-wise or counter clock-wise fashion. This means that for n obstacles, there are n^2 possible configurations of the path relative to the obstacles, and most likely only one of the configurations can be continuously transformed into the shortest path. Figure 2.5 shows this for two obstacles.

Given three points in the pathfinding region, A,B and C. If $|AB| < |AC|$, then we want their images to have the same *relative ordering* $|A^*B^*| < |A^*C^*|$. This property is required for straight lines to be mapped to straight lines, which would be a requirement for the image of a shortest path to be a shortest path as well. However, this is property does not hold for conformal maps because they distort relative distances to attain simpler geometries, which means that some auxillary algorithms are required to guarantee that the path taken around obstacles in the virtual domain can be continuously transformed into the shortest path in the original domain.

Figure 2.5 With two obstacles, there are 4 different configurations of paths between end-points



2.3.3 Shortest-path

While the conformal map simplifies finding a feasible path between two points, it is highly unlikely to be the shortest path will be found by mapping the virtual path into the original domain directly. If a direct line is infeasible between two points, the shortest path will go through at least one corner-point on the obstacle that separates them. This property can be used to find more optimal paths in the conformal domain. This will be discussed in 3.1.2

2.3.4 Spatial dilation

Conformal mappings distorts relative sizes. It is not possible to asses the size or form of an obstacle in the original domain by considering its image. The same goes for relative distances between obstacles. There is no way for any pathfinding agent to determine if it will fit between two obstacles by considering the virtual geometry, the agent can't even directly determine it's own geometry.

This issue can be remedied by dilating all edges and boundaries, also called Minkowski addition [18]. Generally, navigating a body through a space with boundaries is the same as navigation a point through the same space with boundaries, where all boundaries are dilated with the size of the body. An added benefit of this is that some boundaries might overlap when dilated, which essentially merge the overlapping boundaries and simplifies the map.

2.4 Discrete conformal mappings

There are many approaches to create a conformal mapping. In this project I ended up with using a circle-packing methodology combined with discrete Ricci-flow to approximate the conformal mapping. This is not the most precise approach and have quite slow convergence-rate compared to other approaches, but has certain advantages:

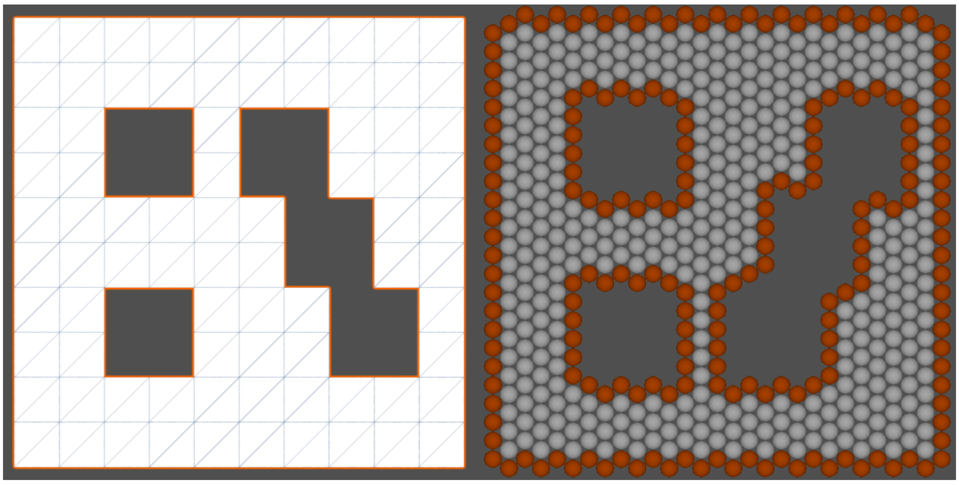
- It is simple to understand. The approach is intuitive, simple to understand and relatively easy to implement.
- It is invariant to the degree of connectedness. It allows many interior obstacles without any additional complexity to the algorithm.
- It's basis in circle-packing allows for keeping important information during the transformation. If the agent can fit within a circle in the original domain, the agent will also fit within the circle in the virtual domain. So any path of connected circles in the virtual domain will result in a path in the original domain that the agent can transverse. Without this property, there would be difficult to determine if the agent would be able to fit between gaps in boundaries because of the distortion of space. The hexagonal packing of circles also informs us of the curvature of the path. From the center of the circle, the angle between neighbouring circles will always be 60 degrees.
- It generates a look-up table. Other conformal mapping approaches approximates an analytic function based on the boundary of the original domain and the corresponding boundary in the virtual domain. This means that for any mapping we wish to compute, a function that scales with the complexity of the boundary has to be computed. Look-up tables have in average constant look-up times.

2.4.1 The circle packing algorithm

William Thurston conjectured that the conformal mapping of a simply connected plane domain $\mathcal{M}$ to the unit disc could be approximated by manipulating hexagonal circle configurations lying in $\mathcal{M}$ [13]. This was later proved by Rodin and Sullivan in 1987 [19]. It is suggested that discrete conformal mappings are well suited when the domain has a high level of connectedness[12]. Because of this, discrete conformal maps approximated by circle packings has been chosen in this project. The circle packing is generated by a hexagonal rasterization of the polygon defining the domain. By chosing all cicles to have the same radius intially, the radius can be used to encode spatial information and makes it possible to do spatial analysis in the virtual domain.

Given a two-dimensional polygon, with optional interior boundaries. We first generate a bounding-box that encompass the entire polygon, with extra padding for possible boundary circles. Then, given the desired radius of the circles, the point of origin for each circle is

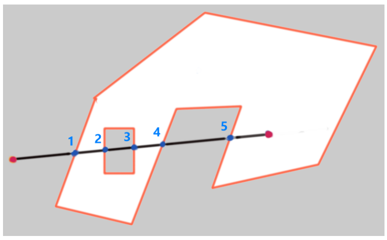
Figure 2.6 Example of a circle-packing rasterization



computed, assuming a hexagonal packing. The hexagonal packing is chosen because it is the most dense circle-packing possible, which will allows for the maximum number of connections between nodes in the graph and thus the most accurate approximation of a conformal mapping.

To determine if a circle is inside the polygon we use ray-casting and the even-odd-rule. Starting from a point outside of the circle, draw a line to the center of the circle. If the line intersects with the polygon an odd number of times, the point center of the circle is inside the polygon, otherwise it's outside. This operation can be done in parallel for all circles within the bounding box.

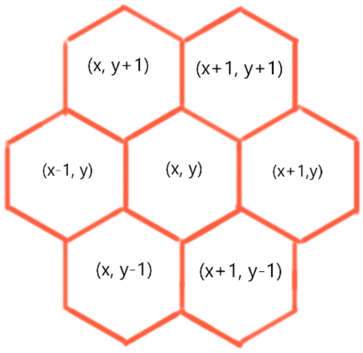
Figure 2.7 The even-odd rule.



When all circles with center within the polygon is found, the next stage is to connect neighbouring circles together to form a connected graph. In this graph, the center of the circles are nodes. Interior nodes will have six neighbours because of the hexagonal packing, and all nodes on the perimeter will have five or less neighbours. Each node is given a two dimensional index that represent its coordinate in the hexagonal grid. During the connection phase, each nodes compute the potential indices of its neighbours based on its own index. All the nodes' indices are pooled to a hash-map that can be queried by each node to find their neighbours. The distance between all neighbouring nodes will be the same, so the actual position of the nodes are irrelevant. The topology of the domain is preserved in the connections between the nodes.

Since the discrete conformal mapping handles domains with many holes quite easily, it is also possible to use the discrete conformal mapping to do a rough estimation of the parameters in the parameter-problem of the Schwarz-Christoffel mapping. To do this, the nearest circle to a vertex in the polygon is labeled as a "corner circle". The corner circles' virtual position can then later be used as the initial guesses for the pre-vertices for the parameter-problem solver.

Figure 2.8 Hexagonal grid indices. In this simple graph, the center hexagon has six neighbours and each perimeter node has three neighbours.



For the discrete conformal mapping to work, there are certain requirements to the graph. First, boundaries can't share nodes. This is obvious because it would lead to the two boundaries to compete for the nodes' radius to approximate their respective circle. Consequently, any node can only be part of one boundary. Secondly, boundary nodes must always have two neighbours that also are boundary nodes, so that we can form a simply connected circular boundary. An additional consideration is the complexity of the domain. If there are many dead ends, sharp turns, and other difficult configurations, the conformal mapper will have to perform severe warping of the domain to produce a virtual domain with the desired properties. The higher resolution the rasterization has, in other words the smaller the radius of the circles used, the more freedom the mapper has to warp the domain.

It is important to remember that the graph is to be used by a agent for pathfinding. A the size of the graph scales with the resolution of the rasterization. High-resolution graphs take a longer time to be processed by the conformal mapping algorithm and results in higher memory-requirements. It also results in the agent having to more frequently querying the pathfinder for new waypoints, resulting in higher computation costs. Because of this, it is desirable to keep the resolution as low as possible.

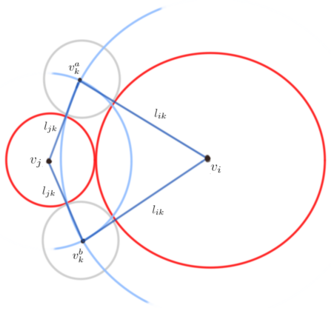
2.5 Flattening algorithm

After the Ricci-flow algorithm terminates, the virtual coordinates of the nodes needs to be computed. Since all nodes have the same neighbours, and the ordering of the neighbours are preserved, it is possible to compute relative coordinates for all nodes given position of at least two nodes.

By choosing an initial node and defining its position to be zero, and choosing one of its neighbours and giving it the position (0, l_{ij}, the position of all other nodes can be computed by a flooding algorithm.

Given two neighbours v_i with radius r_i and position p_i and v_j with radius r_j and position p_j and their common neighbour v_k with radius r_k. Then p_k will be the intersection of the circles with centers at p_i and p_j with radii $l_{ik} = r_i + r_k$ and $l_{jk} = r_j + r_k$ respectively. This gives two potential positions. To solve for one unique position we use the fact that v_k will lie clockwise to v_j from v_i's perspective, and counter-clockwise to v_i from v_j's perspective.

Figure 2.9 The position of v_k can be derived from its radius and its neighbours radius and position



When the new node has been given a position, it is added to the visited collection and its neighbours that haven't been visited is added to the frontier. The process continues to all nodes have been given a position. If a node in the frontier doesn't yet have two neighbours that have been visited, it is added to the back of the frontier.

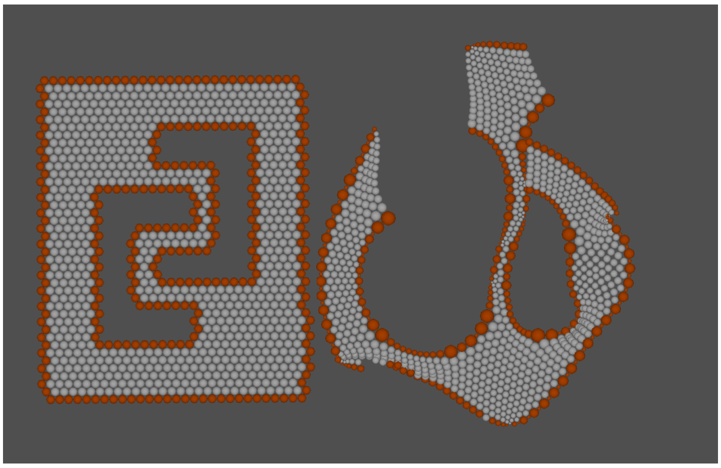
The flattening algorithm is quite sensitive to errors in the graph. If the accuracy of the radii in the graph are too low, a cascading effect will result in misalignment of the nodes distant from the seed nodes. An example of a such misalignment can be seen in Figure

Algorithm 2 Flattening of circle-graph

```
Require: Triangular mesh M, with vertices $v_i$ that has a radius $r_i$
Ensure: All vertices $v_i$ has a coordinate $p_i$
  Initialize an empty collection $visited$. Initialize an empty queue $frontier$
  Take two arbitrary nodes that are neighbours to eachother $v_i$ and $v_j$. Set the position of $v_i$, $p_i = (0,0)$ and the position of $v_j$, $p_j = (0, r_i + r_j)$. Add their neighbours to $frontier$.
  while $frontier \neq$ do
    Take the next node $v_k, v_k \in frontier$.
    if $v_k$ has two common neighbours in $visited$ then
      Compute radii $l_{ik} = r_i + r_k$ and $l_{jk} = r_j + r_k$
      Compute the intersection point $v_k$ between the circles with centers $p_i$ and $p_j$ with radii $l_{ik}$ and $l_{jk}$ respectively.
      Add $p_k$ to $visited$
      Add all neighbours of $v_k$ that is not visited to $frontier$
    else
      Add $v_k$ to the back of $frontier$
    end if
  end while
```

2.10. The required accuracy is dependent on the number of nodes, as the error accumulate through the flooding algorithm. The neighbourhood close to the seed nodes usually are well-formed, with the error manifesting further out and get especially visible around obstacles.

Figure 2.10 Premature termination of the discrete Ricci-flow algorithm can lead to arbitrary graphs



1. Given two points in the original domain, we want to compute a path between them

2. Find the nearest circle-nodes to the two points
3. We map the coordinates of the two circle-nodes to the virtual domain
4. We compute a line between the two virtual coordinates
5. Given the starting point and the line, find the neighbouring node of the starting point that the line intersects. This will be the next node in the path towards the destination
6. Map the virtual coordinate of the next node back to the original domain
7. The agent can move towards this coordinate on a straight line and get closer to the destination

By using discrete conformal pathfinding, we don't have to compute the entire path between two locations up front. At any time we can look up the next point in the path towards the destination. This has significant advantages in terms of computational costs in certain situations.

- If the target destination is only temporary; that the destination will most likely be changed before the agent has reached it.
- If the target is moving.
- If the number of agents requiring pathfinding simultaneously is high enough,

While this approach is extremely efficient at finding a path between two points, a straight line in the virtual domain will essentially never result in any straight lines in the original domain, and by extension never result in the shortest path, a few exceptions.

For a simply connected region, the shortest path can be found by a continuous deformation of any other simple path in the region. Intuitively, we can imagine that, given any simple path between two point, the shortest path is found by "tightening" the path, much like tightening a rope. At any time, we also only require the algorithm to "tighten" the path between its current location and the first tangent with a boundary. This will be a straight line toward that point.

For the multiply connected domain, for the "tightening" approach to be valid, we require the shortest path to be a continuous deformation from the proposed path. In other words, we require the proposed path to go navigate between the boundaries similarly to the shortest path.

It's not the case that the shortest path found in the virtual domain is a continuous deformation away from being a shortest path in the original domain, to find the shortest path we will have to calculate the lengths for the various proposed paths and choose the shortest from this list.

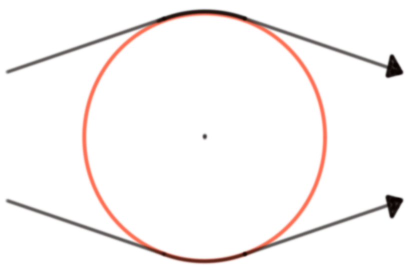
One could argue that we actually still do the search, we only do it ahead of time in the form of computing the conformal mapping. In fact, the Floyd-Warshall algorithm also computes the shortest path between any two points ahead of time, and the only operation required online is performing a look-up. The Floyd-Warshall algorithm has a $O(n^3)$ computational complexity and generates a $O(n^2)$ look-up table [2, p.693]. The conformal mapping approach has a $O(nlog(n))$ computational complexity and generates a $O(n)$ look-up table.

2.6 A* algorithm with circular obstacles

When all obstacles are circles, a more specialized version of A* can be implemented. While the classical A* works over a graph where each node represents an segment of area feasible for navigation, the circular A* algorithm have nodes that represent the perimeter of obstacles. The underlying algorithm is similar, the main difference is in how the graph is constructed.

The optimal path when navigating in a forest of circular objects will be a combination of bi-tangents between circles and arcs hugging the obstacles. The circular pathfinding graph will for the use of conformal pathfinding be superimposed on the conformal graph. The arcs and line-segments generated by pathfinding on the circle graph will serve the same purpose as the simple straight lines in conformal pathfinding without obstacles.

Figure 2.11 Clockwise nodes connects and and counter-clockwise node connects

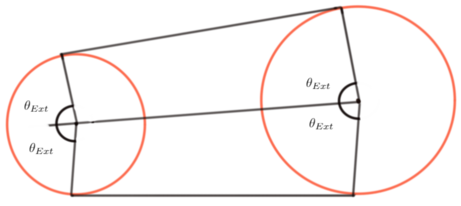


The circle graph is constructed by connecting the circles with bi-tangents. There are two different types of bi-tangents, namely internal and external bi-tangents. All connection points on the same circles are then connected with each-other. All clockwise input points will be connected to all clockwise output points and the same goes for the counter-clockwise case.

External bi-tangents

External bi-tangents are line segments that tangent both circles and does not cross the mid-line that connect the center points of the circles. When a agent follow an external bi-tangent, it will move around both obstacles in the same turning direction. That is, if it goes around obstacle A clockwise, it will also go around obstacle B clockwise. The angle relative to the mid-line in which the connecting line tangent the circle is given by

Figure 2.12 External bi-tangents



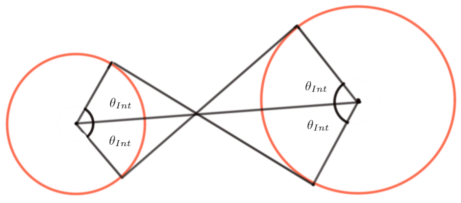
$$\theta_{Ext} = \sin^{-1}\left(\frac{r_A + r_B}{d}\right)$$

Where r_A and r_B are the radii of the circles and d is the length of the mid-line

Internal bi-tangents

External bi-tangents are line segments that tangent both circles and crosses the mid-line that connect the center points of the circles. When moving along an internal bi-tangent, the turning direction of the agent changes. The angle i which the connecting line tangent the circles relative to the mid-line is given by

Figure 2.13 Internal bi-tangents



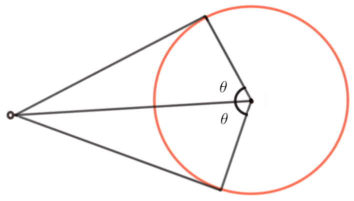
$$\theta_{Int} = \sin^{-1}\left(\frac{|r_A - r_B|}{d}\right)$$

Where r_A and r_B are the radii of the circles and d is the length of the mid-line

Connecting to the circle graph

When the agent want to do pathfinding on a circle graph, it must first connect to the graph. In classical A*, the agent is located on a node. In circular A*, nodes represent obstacles, so agents cannot be located on them. Instead, the agent must create a temporary node that connects to the circle graph. This is done by computing the bi-tangent between the point where the agent is located and all circles. All tangent-lines that are not intersected by other circles serve as initial connections to the graph for the agent. The same procedure must be performed for the endpoint. When the algorithm terminates, the temporary nodes and connections are deleted again.

Figure 2.14 bi-tangents to a point



The angle in which the connecting line tangents with the circles is the same as for the other bi-tangents, with the radius of a point being zero.

$$\theta = \sin^{-1}\left(\frac{r}{d}\right)$$

Where r is the radius and d the length of the mid-line

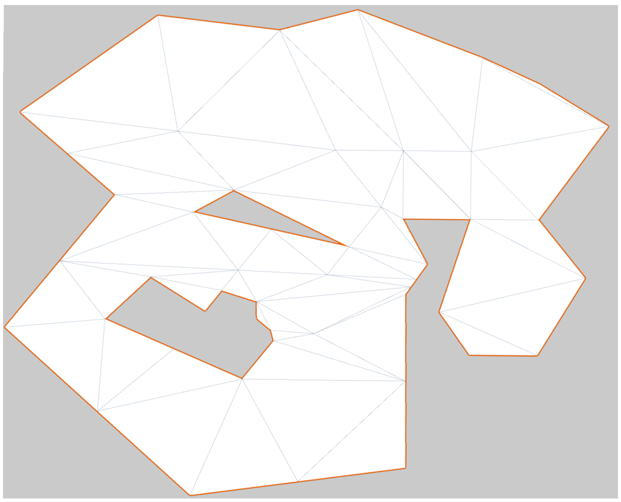
In the case of intersecting circular obstacles, the internal bi-tangents would be removed. In conformal pathfinding, however, the obstacles will never intersect or even tangent. If two obstacles intersect or touch in the original domain, they will be considered a single obstacle from the pathfinders perspective and form a single circle.

Chapter 3

The conformal pathfinding algorithm

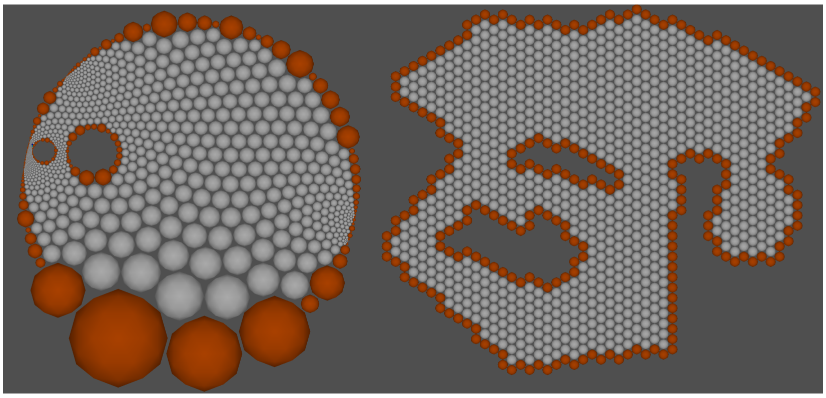
While the essence of the conformal pathfinding is pretty straight forward, it doesn't compute shortest paths. To do that, extra steps needs to be applied.

Figure 3.1 A map that represent the movable area for a navigation agent



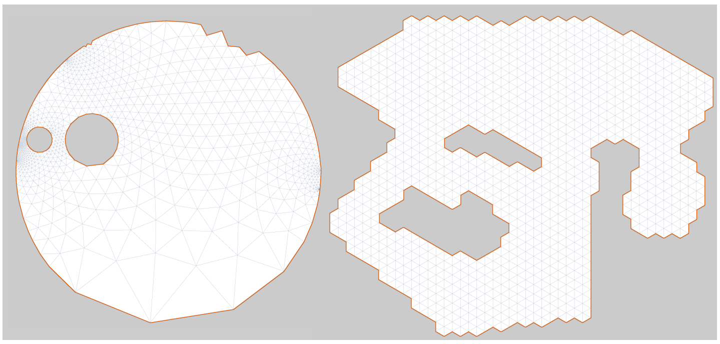
Given a map that's composed of polygons; one for the boundary and the rest for interior obstacles. Before pathfinding can be executed, the map needs to be processed. The first step is to find the conformal mapping of the original region to a domain consisting of a circular boundary and only circular interior obstacles. To find this mapping, the original domain is first rasterized into a graph consisting of equilateral triangles and then processed with the discrete Ricci-flow algorithm.

Figure 3.2 Circle packing of map and the results of the Ricci flow algorithm



From the generated virtual domain, all the interior boundaries are then represented as circles in an circular graph which will be used for circular pathfinding. The circles found by taking the virtual coordinates of the circular boundaries and using the linear least-squared algorithm for circle-fitting.

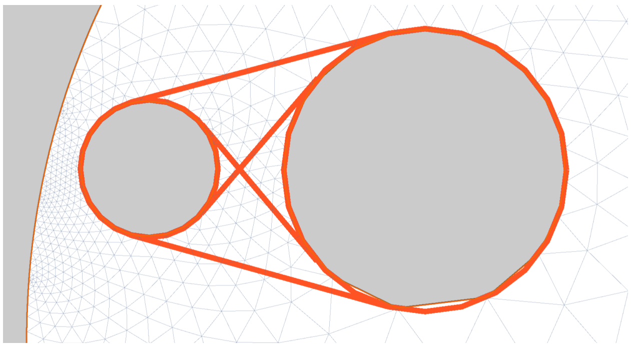
Figure 3.3 Conformal mapping between original and virtual domain



The weight of the edges in the circular graph will be the length of the shortest path between the nodes in the edge in the original domain. An optional extension to the map is also a repulsion field describing the virtual node-density which can be used for path optimization later on.

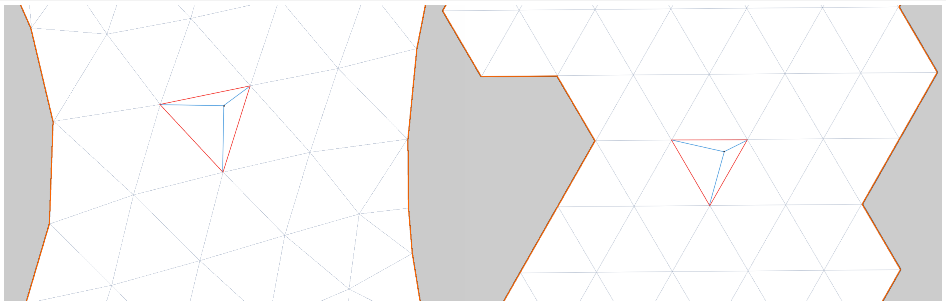
When the map is processed, it is ready for pathfinding requests. The requests will be a pair of coordinates representing the to -and from destination on the map. The first step is to find the virtual coordinates of the pair. This is done by finding the triangle of nodes

Figure 3.4 The circular pathfinding graph that is extracted



encompassing the coordinates and then representing the endpoints by their barycentric coordinates given the encompassing triangles. Then in the virtual domain, the virtual coordinates are found by converting back from barycentric to global coordinates given the virtual coordinates of the nodes as seen in 3.5

Figure 3.5 Mapping between original and virtual coordinates through a barycentric transform. Virtual map on the left and original map on the right.

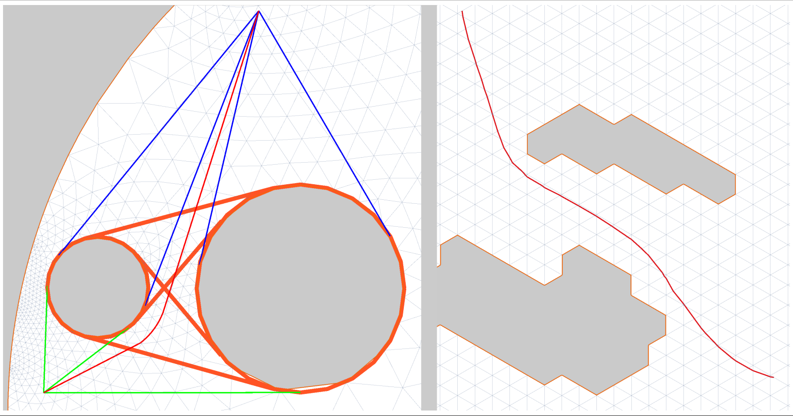


The virtual coordinates are used to generate start and end nodes which are connected to the circular pathfinding graph. For the best result, the generated edges should be weighted with the real-life shortest-path distance these edges would have had. A less expensive optional optimization could be to estimate the path-length of the virtual path based on the intersection the path has with the triangles in the virtual graph. Now, circular pathfinding can be performed and a virtual path between the endpoints are found.

3.1 Multistage pathfinding

With conformal pathfinding, the pathfinding process can be divided into two stages. One stage determines the path configuration, and the other stage computes the actual path.

Figure 3.6 The circular path in the virtual domain and its original dual. The green and blue lines represent the extended graph that includes the start and endpoint as nodes. The red line represents the path. The reason for it not hugging the circles is because a radial padding has been added to avoid the path visiting dead ends.



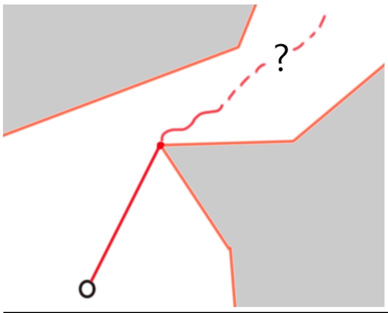
3.1.1 Macro-scale pathfinding

This stage involves finding the correct path configuration as described in 2.3.2. This can be regarded as a required rough outlining of the complete path between the two endpoints. It is only necessary when obstacles are involved because which direction the agent will go around obstacles has to be decided.

3.1.2 Micro-scale pathfinding

When the rough outline of the path is determined, additional optimization of the path can be done continuously while the agents move along the path. If the path configuration with the shortest possible path is chosen at the macro-scale phase, the shortest path can be found by look-ahead to the nearest corner, one corner at a time. This can be achieved with look-ahead algorithms that optimize the path around the next corner. The reason this is true is because of the optimal substructure property of shortest paths, stating that the shortest path between two points contains all shortest paths between the points it is a superset of [2, p.644], and the property that the shortest path between two points is a straight line.

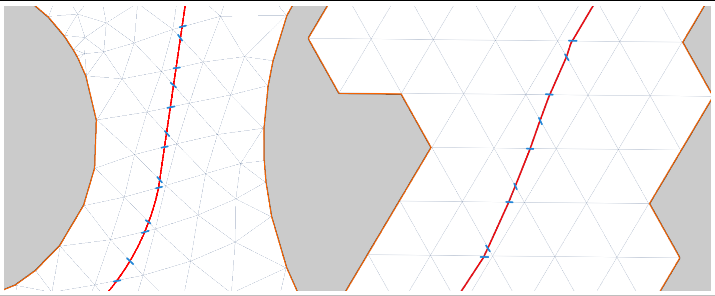
Figure 3.7 Micro-scale pathfinding can be done to find the shortest path online



Optionally before transforming the virtual path back into the original domain, the path can be smoothed with the repulsion field. This will result in a shorter original path. A much less expensive and less accurate smoothing is to just not let the virtual path hug to the obstacles, but to radially push the path away from the obstacle. This kind of smoothing can be seen in 3.6

Now, given the virtual path, the navigation agent can start moving. The way the virtual path intersects with the triangles on the virtual graph is used to find barycentric coordinate points that given the original coordinates of the edge-nodes are transformed into way-points in the original domain. When the navigation agent reach a way-point, the next way-point can be computed on-demand. In other words, the original path is computed online.

Figure 3.8 Path is transformed by barycentric mapping based on where the virtual path intersect with triangle edges



This is close to the shortest path, but not actually the shortest path. To get the shortest path, it is possible to compute the path ahead around the first obstacle, and then snap the path to be the direct line to the corner of that obstacle. To detect when the path has ”gone around” an obstacle, the algorithm search for the first time the line-of-sight intersects with that boundary.

In the case of pathfinding with no obstacles, the circular pathfinding step is omitted. The virtual path will be a straight line and the same repulsion-field post-processing of the virtual path can be performed to get a more optimal shortest path.

Chapter 4

Experiments

To test the conformal pathfinding algorithm, several experiments have been conducted. The goal with the experiments is to answer the following questions.

1. What is the runtime performance of the conformal pathfinder compared to the current standard, the A* algorithm?
2. How does the conformal pathfinder deal with various geometries, and are there any limitations to what maps it can handle?
3. Is it possible to use the conformal pathfinder to find the shortest path?

The experiments are conducted on a computer running 64-bit Windows 10 Home. The processor is a Intel(R) Core(TM) i5-7600K CPU @ 3.80GHz, 3792 Mhz, with four cores and 8 GB physical memory.

The Unity-engine used as reference is the 2019.1.3f1 version.

Factors that can interfere with the experiments includes:

1. Other processes running in the background
2. The graphics rendering required to conduct the experiments easily
3. The test-code is non-optimized and largely object oriented, which can affect the performance

4.1 Spatial warping

To convert a domain into a convex representation, the original map goes through a process of local contractions or dilations. This results in an uneven spatial resolution over the virtual domain, where nodes form local clusters. From the various maps studied, the clusters always seem to have their epicenters located along the perimeter of the boundary and obstacles. The same goes for local dilations.

To investigate the behaviour of spatial warping, several maps are tested. There are several factors that can effect the warping of the domain.

4.1.1 Protrusions into the domain

Parts of the perimeters that protrudes into the domain creates obstacles that the agent has to go around. Protrusions manifest when the boundary has a negative turn going counter-clockwise, or when the perimeter of obstacles take positive turns. The conformal transform removes such obstacles by dilating the area close to the perimeter of the obstacle.

To study the behaviour of the dilation, a rectangular region with a triangular protrusion is considered. The effect of changing the sharpness of the protrusion is measured and can be seen in Figure 4.1. By making the protrusion sharper, the node density was more evenly distributed over the domain. Two paths are also plotted. The red curve represent the shortest path in the virtual domain, and the blue curve represent the shortest path in the original domain. Notice that a more evenly distributed node density leads the shortest path in the virtual domain more closely approximates the shortest path in the original domain.

If a straight line goes through a dilated region, the number of triangles intersected is small, as opposed to going through a cluster. This means that going through dilated regions results in shorter paths. This is works well with the fact that the shortest path when bypassing obstacles involves touching their boundaries. In the virtual domain, however, it is not straight forward to know if a path actually has to go around an obstacle since this information is obfuscated by spatial warping. In Fig 4.2, the shortest path in the original domain does not cross an obstacle. Notice that the mapping of the virtual path into the original domain seem to get pulled downwards close to the endpoints, and pushed upwards towards the middle. Also notice this effect is more prominent in with the sharper protrusion. In the virtual domain, the mapping of the shortest path in the original domain still gravitates towards the protrusion.

The observations from this experiment suggests that to make the conformal pathfinder approximate the shortest path in the virtual domain, local dilations should act as an attractor for the virtual path. Even if the path does not intersect with the obstacle, the protrusion generates a pull of the entire field in the virtual domain.

Figure 4.1 Domains with sharp protrusions. The left domain has a sharper protrusion than the right. The blue curve in the two upper rows represent the shortest path between the two endpoints, the red line shows the virtual shortest path.

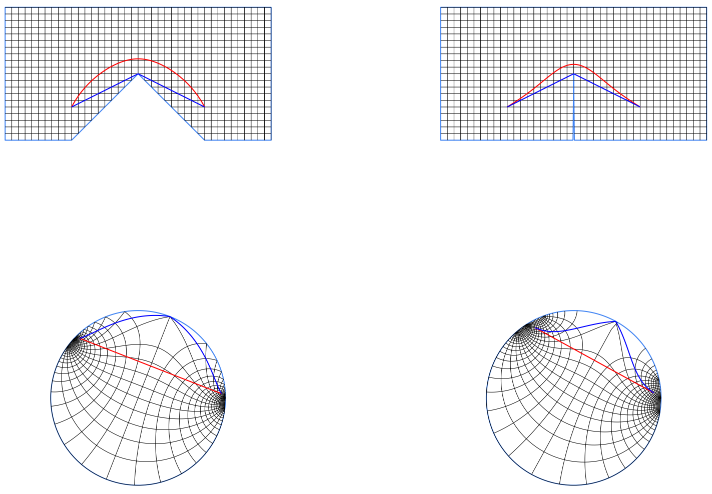
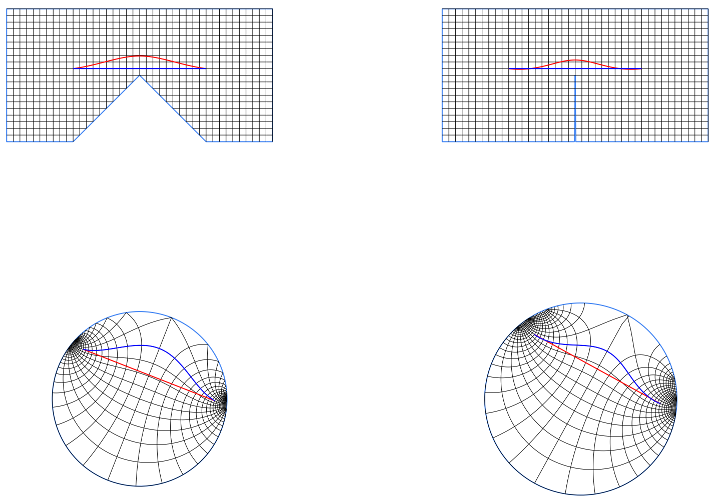
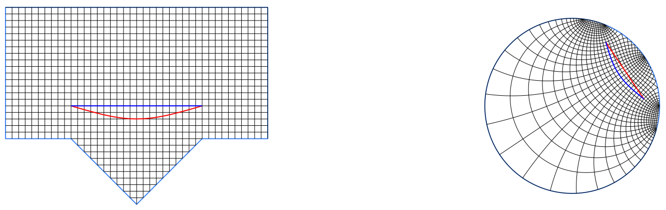


Figure 4.2 Domains with sharp protrusions. The left domain has a sharper protrusion than the right. The blue curve in the two upper rows represent the shortest path between the two endpoints, the red line shows the virtual shortest path.



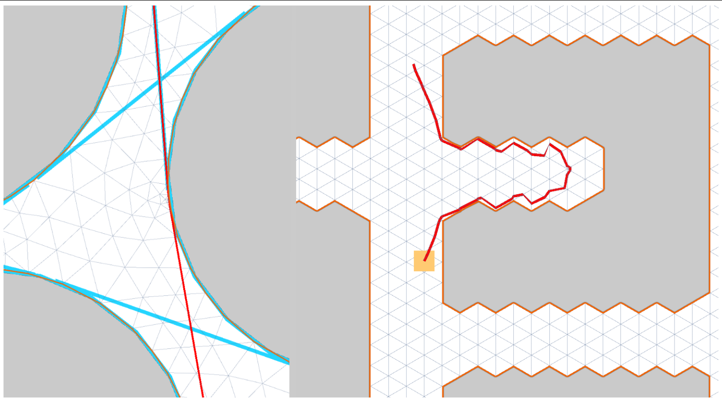
4.1.2 Retrusions out of the domain

Figure 4.3 Domain with sharp retrusion.



Retrusions can be regarded as ”pockets” of the domain. When an obstacle must be bypassed as part of a path, the agent wants to go ”around” the obstacle. All retrusions into that obstacle are dead ends and unnecessary extensions to the path. To smooth out dead ends in the virtual domain, the conformal transform contracts the pocket into the rest of the domain. This results in the pocket being represented by a clustering of nodes in the virtual domain. This can be seen in 4.3. Notice that the shortest path here behaves opposite of how it behaved with retrusions. The shortest path in the original domain is repulsed by the cluster.

Figure 4.4 Retrusions out of the domain results in clusters that will ”suck in” the path if the virtual path goes through the cluster.



Since the resulting obstacle is a circle, this means that the shortest path around that obstacle is the arc of that circle, which in the original domain translates to visiting all ”pockets” of that obstacle, and example of this can be seen in Figure 4.4. The simple approach to this problem is to just let the path around the obstacles not go along the edge by increasing the radius of the arc. The problem is that while the agent should avoid retrusions, we have already observed that the agent should visit protrusions, which will be avoided by just increasing the radius of the arc. Generally, increasing the radius will make the longest paths shorter and the shortest path longer, which might be an acceptable compromise.

A more involved optimization would be to use potential fields. Clusters could be used to create repulsive fields while dilations could create attractive fields. The force would be dependent on the density. Gradient descent could then be used to find an approximation of the shortest path.

Since the value of conformal pathfinding lies in its ability to ignore the complexity of local geometry when doing pathfinding in the virtual domain, the more algorithms that goes into do that analysis, the less the value of the conformal pathfinder. The least expensive variant of the pathfinder produce a path that consists of straight line segments. This approach does not consider the quality of the resulting path in the original domain.

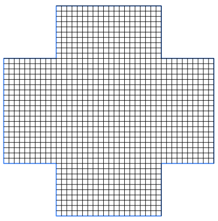
4.1.3 Elongated and narrow domains

The warping of the domain is not only dependent on local dead ends and obstacles, but also on the relative distribution of them in the whole region. For instance, a region with an even distribution of obstacles and dead ends that also are equally sized results in a symmetric and relatively evenly distributed virtual domain. An example of this can be shown in Fig. 4.5.

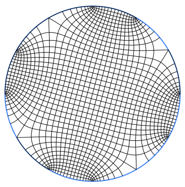
When domains have long or narrow regions, the distribution becomes much more uneven. This is known as the crowding phenomenon, and results in a reduction of precision in the conformal mappings. Essentially, very small regions in the virtual domain comes to represent very large parts of the original domain. If the domain is has regions that are too narrow or too elongated, some regions becomes vanishingly small, and compromise the conformal pathfinder. This will be further investigated in 4.5.

In Fig. 4.6 the problem with elongated regions becomes apparent. The uneven distribution results in sub-optimal pathfinding solutions because of a decreased precision.

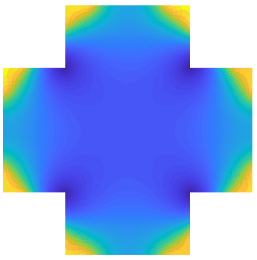
Figure 4.5 Example of a domain with a even distribution. The heatmap in (c) and (d) show the local node-density of the virtual domain. The yellow corners have a high density of nodes, while the dark blue corners have a very low density, but overall the density is evenly distributed throughout the domain.



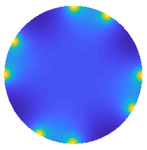
(a) A domain with evenly distributed protrusions and retrusions



(b) The virtual mapping of the domain

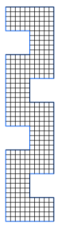


(c) The node density in the virtual domain projected back on the original domain

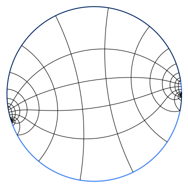


(d) The node density in the virtual domain

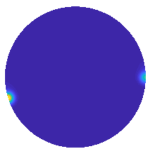
Figure 4.6 Example of a domain with a uneven distribution. The heatmap in (c) and (d) show the local node-density of the virtual domain. In this instance, the spatial warping is severe, and large regions of the original domain is represented by very small virtual regions, making pathfinding difficult.



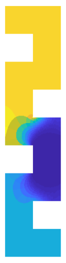
(a) A domain with evenly distributed protrusions and retrusions



(b) The virtual mapping of the domain



(c) The node density in the virtual domain projected back on the original domain



(d) The node density in the virtual domain

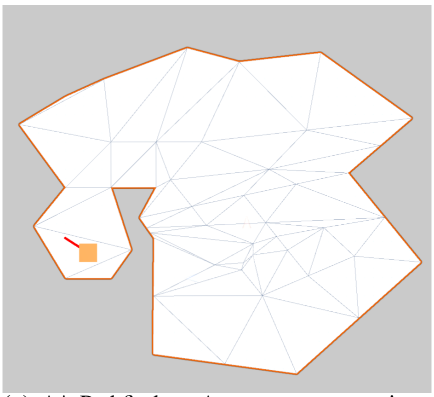
4.2 Simply-connected domains

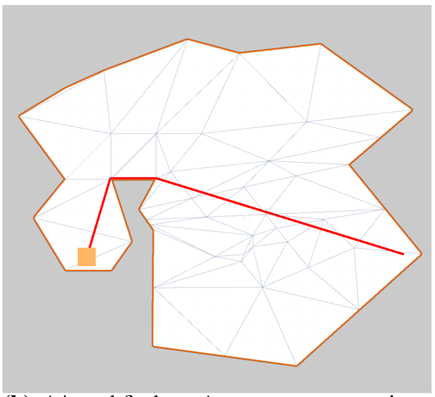
When no obstacles are present in the map, conformal pathfinding is straight forward. This experiment focus on the responsiveness of the conformal algorithm when no circular pathfinding is required.

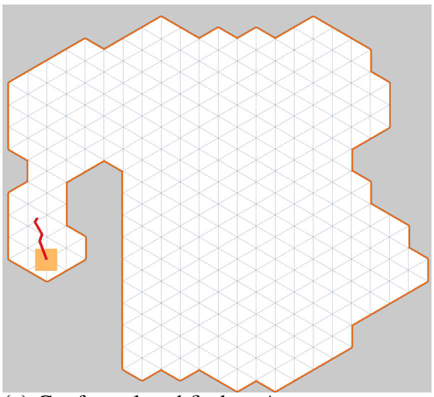
In this case, the response-time of the pathfinder should be close to constant. However, before the path can be computed, the barycentric triangle for both endpoints must be computed from the coordinates requested. This time is included in the time estimate.

The test is conducted by requesting the agent to move between arbitrary endpoints. Some endpoints will be close while others will be as far away from each other as possible. Several hundred requests are given and the minimum, maximum and average response times are measured.

Figure 4.7 Comparison of pathfinders in a simply-connected domain


(a) A* Pathfinder: Average response time: 0.01 ms


(b) A* pathfinder: Average response time: 0.04 ms


(c) Conformal pathfinder: Average response time: 0.04 ms

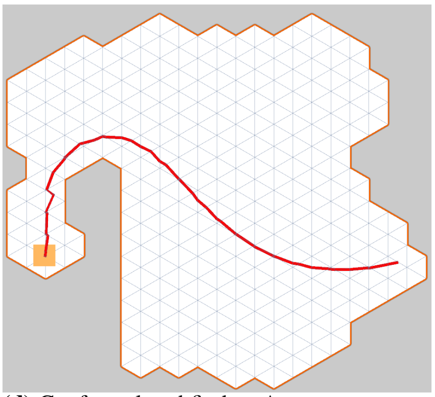

(d) Conformal pathfinder: Average response time: 0.04 ms

Table 4.1: Comparison of the A* pathfinder and the conformal pathfinder

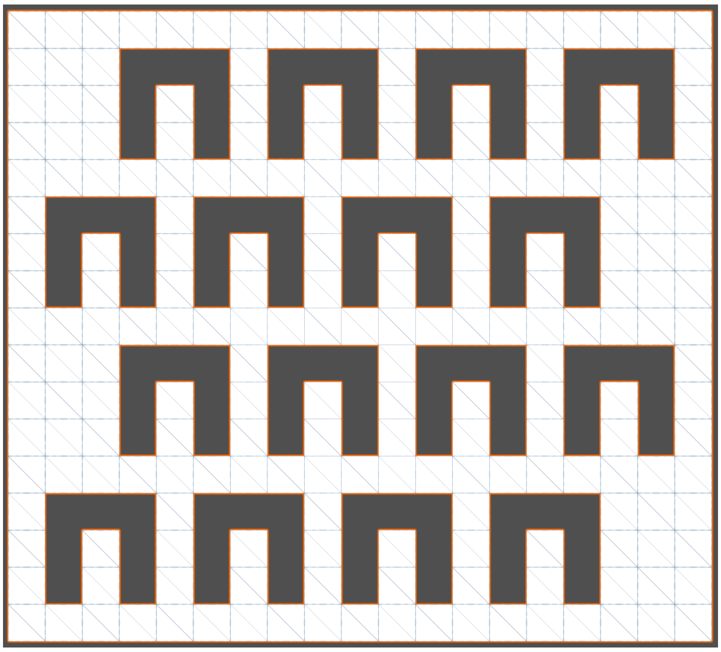
	A* pathfinder	Conformal pathfinder
Shortest response time (ms)	0.01	0.02
Longest response time (ms)	0.12	0.12
Average response time (ms)	0.03	0.04

In the tested cases, the response time of the conformal pathfinder seemed to be invariant to the distance between the endpoints and had an average of 0.04 milliseconds (Fig 4.7c and 4.7d). Deviations are attributed to the triangulation procedure and the process sharing the CPU with other processes. The A* pathfinder had a response time that varied based on the distance between the endpoints, with points being inside the same node of the mesh having an average response time of 0.01 milliseconds (Fig 4.7a) while the average response time when the endpoints where as far away as possible was measured to 0.04 milliseconds (Fig 4.7b).

The observed jitter in the paths generated by the conformal pathfinder is because the conformal mapping is only an approximation. Increasing the resolution of the rasterization reduces this problem, but it generally the path should go through post-processing after the virtual path is mapped to the original domain. The path generated by the A* pathfinder is run through a funnel modifier before it is presented [20]. Without this, it would also be jittery.

4.3 Pathfinding with many obstacles

Figure 4.8 A tessellation of non-convex obstacles



With many obstacles, the playing-field is somewhat leveled between conventional A* and conformal pathfinding as the circular graph becomes large. An example of one such domain is shown in Fig. 4.8. This results in the conformal pathfinder spending a significant time performing circular A* computations. The benefit of the conformal pathfinder in this scenario is that it disregards dead-ends. When going ”upstream”; for every obstacle in the domain, the A* pathfinder will consider the dead-end in that obstacle. In the down-stream case it will not.

In 4.2, the response-time of the A* pathfinder and the conformal pathfinder is compared. When given a clear path, the conformal pathfinder performed almost as well as the A* pathfinder. The extra time is suspected to be because the conformal pathfinder must calculate if it intersects with any circle before the straight virtual path is computed. The conformal pathfinder has a response-time that is invariant to the path-request is downstream or upstream, while the A* pathfinder shows a slightly longer time when going upstream than when going downstream.

Figure 4.9 The virtual domain of the domain in Fig. 4.8

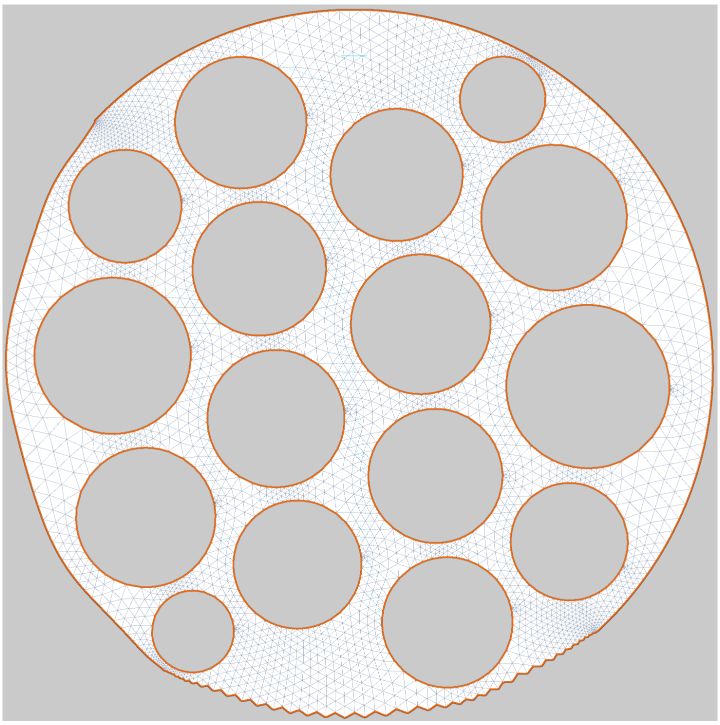
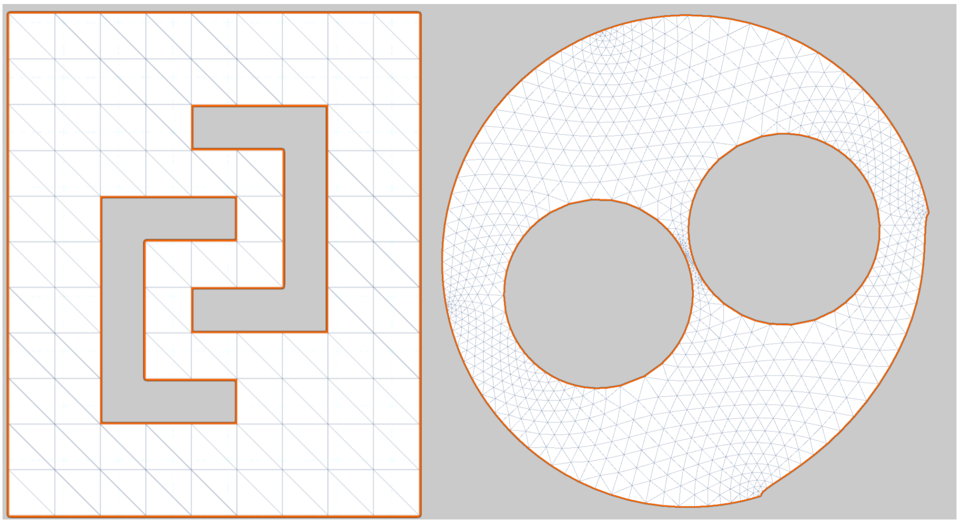


Table 4.2: Comparison of the A* pathfinder and the conformal pathfinder

	A* pathfinder	**Conformal pathfinder**
Shortest response time (ms)	0.02	0.05
Longest response time (ms)	0.21	5.23
Average response time (ms)	0.62	2.23

Figure 4.10 Entangled obstacles doesn't look entangled in the virtual domain



4.4 Entangled obstacles

The shortest path in the original domain doesn't necessarily have the same path around obstacles as the shortest path in the virtual domain. This becomes evident with entangled obstacles. The way obstacles can intertwine is obfuscated in the virtual domain. This means that even though there doesn't exist a straight path between two obstacles in the original domain, it will exist in the virtual domain. If the two endpoints are aligned correctly, they might get connected by a straight line between two obstacles that are entangled in the original domain. An example of an entangled domain is shown in Fig. 4.10

Table 4.3: Comparison of the A* pathfinder and the conformal pathfinder

	A* pathfinder	**Conformal pathfinder**
Shortest response time (ms)	0.02	0.07
Longest response time (ms)	0.08	0.53
Average response time (ms)	0.04	0.15

In 4.3, the response time of the conformal pathfinder and the A* algorithm is compared for the entangled domain. For direct paths, the result is comparable, but when navigating through obstacles, the A* pathfinder outperforms the conformal pathfinder. This suggests that it is the circular A* pathfinder that is the bottleneck for the conformal pathfinder. The conformal pathfinder also consequently wanted to go through the entangled region, resulting in paths that could not have been reduced to the shortest path.

When obstacles are entangled, the node density is high in the area between them. This can be used as an indication on that the path between those obstacles are long. If this information can be embedded into the pathfinder, it is possible to avoid them. One possible solution is to add costs to the edges in the circle-graph that represent the real-life cost of traversing those paths.

An alternative approach to using node densities, is to adjust the weights on the edges in the virtual domain to correspond to the real-life shortest distances of those edges. This would give the pathfinder an indication of which configuration that contains the shortest path. The use of weight adjustment will not change the computational cost of the pathfinder and can result in much better results. The problem with this strategy is that the bi-tangent connections in the circle-graph won't be placed relative to any corner on the obstacle, resulting in the length provided being somewhat arbitrary with respect to the need to know which configuration to choose.

4.5 Infeasible maps

A well known problem in with conformal mappings is the crowding phenomenon [11]. This is when the pre-vertices in the virtual domain gets really close and results in extreme warping of the domain and loss of precision. The crowding phenomenon happens when the original domain have narrow or elongated regions. Several methods have been proposed to solve this. One approach is to have a virtual domain that can have a shape that is reactive to the crowding phenomenon, like a rectangle that can be elongated to compensate for the crowding. Another proposed solution involves adding synthetic vertices that ”break-up” elongated regions [1].

The crowding phenomenon has resulted in the conformal pathfinding algorithm failing for regions that are elongated or have narrow sections.

The zig-zag domain

The zig-zag domain (see Fig 4.11) is designed to be beneficial for the conformal pathfinder. While the classical A* algorithm has to explore many dead ends, the conformal pathfinder only has to navigate around a circle, yielding a very short response-time. However, the domain was too demanding to map for the Ricci-flow algorithm when the difference in radius between the smallest circle and the largest circle was too high, resulting in precision errors.

Nested obstacles

It is not only elongated domains that the conformal pathfinder struggles with. When the original domains have narrow gaps that separate the domain, the conformal map gets extremely warped. In Fig. 4.12, a domain with nested boundaries is considered. The expected result when mapping it to the virtual domain was a ”recursive” behaviour, where the circular obstacles would be generated along one axis, but on alternating sides with respect to eachother becoming increasingly smaller. The main interest was by how much each obstacle would become down-scaled.

The expected result was confirmed with testing. The conformal mapping is shown in Fig. 4.13. The conformal mapping breaks for the innermost obstacle because the radii becomes too small.

Figure 4.11 The zig-zag domain

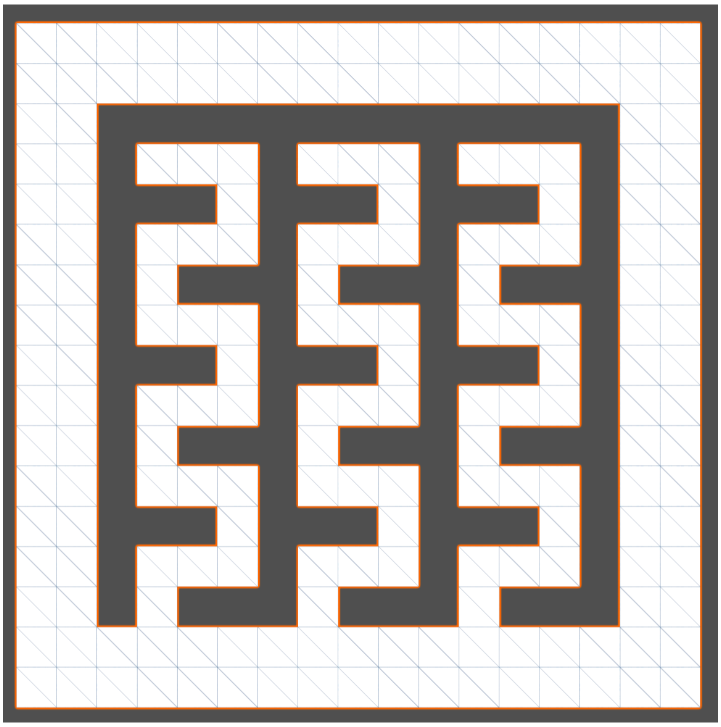


Figure 4.12 Domain with nested obstacles. The narrow openings into the ”next level” causes extreme warping

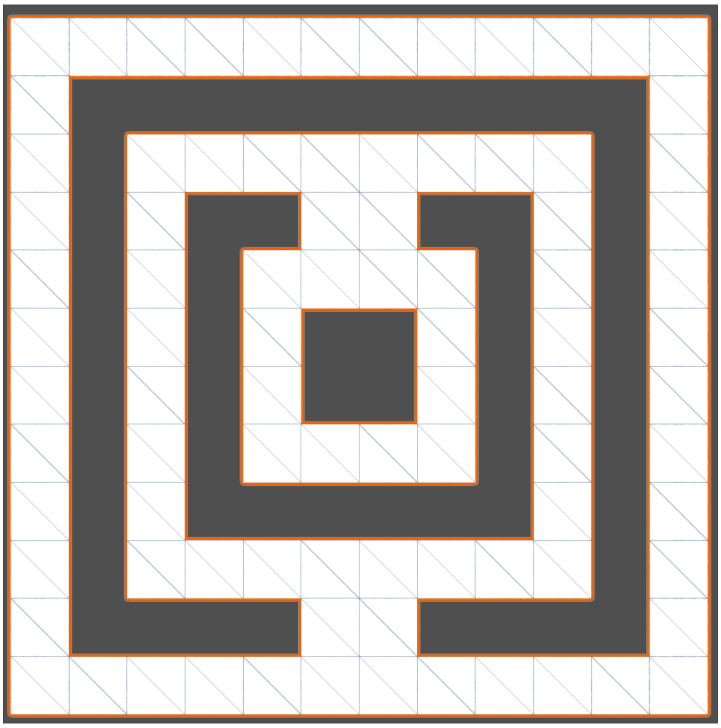
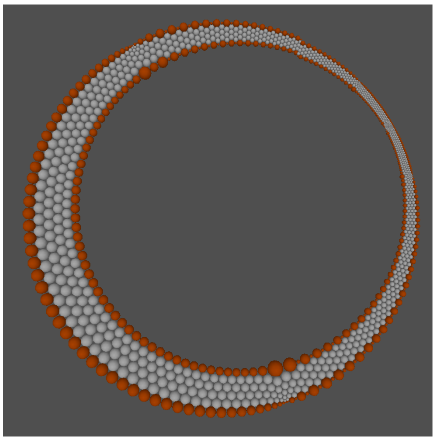
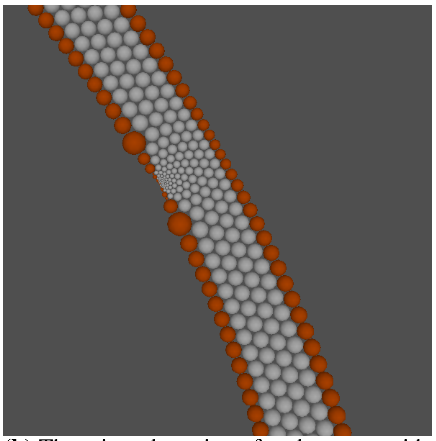


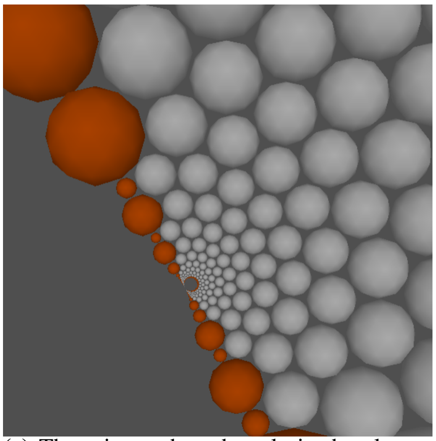
Figure 4.13 Nested obstacles result in extreme warping and breaks the conformal mapper



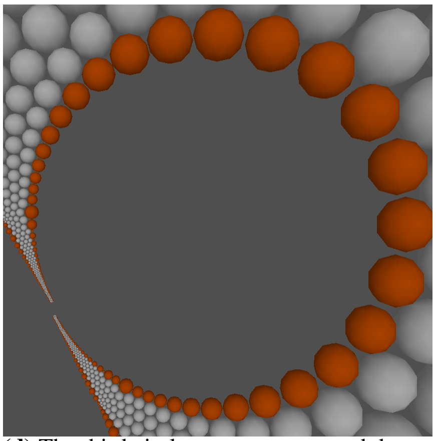
(a) Only one obstacle is seen



(b) There is a clustering of nodes at one side of the circle



(c) There is another obstacle in the cluster, much smaller than the first



(d) The third circle was not generated due to the extreme warping

Chapter 5

Discussion

Conformal pathfinding is a technique where the geometry of a two dimensional domain is simplified by a conformal mapping before pathfinding is performed. This mapping lets the agent ignore the local complexities of obstacles and the boundary of the map during pathfinding, resulting in a faster response-time. In its most general form, conformal pathfinding does not allow for shortest-path pathfinding.

The mapping from arbitrary domains to the circle-domain was found to be too demanding for domains with complicated domains. If the domain contained either too elongated regions or too narrow openings, the spatial distortion of the domain became too extreme, resulting in the failure to generate the virtual coordinates required to perform pathfinding in the virtual domain. This limits the applicability of the conformal pathfinder to domains with feasible geometries.

In this project, I have studied the effects of conformal mapping on the geometry of several different domains with focus on the application of pathfinding. If any path to the goal is sufficient, then conformal pathfinding outperforms conventional A* pathfinding in response-time. The resulting path will be smooth and end up at the target destination, but it will most likely not be the shortest path. Even if the resulting paths aren't the shortest, they usually are pretty good.

With a slight modification to the circular pathfinder, the paths around obstacles won't hug the perimeter, and dead-ends are mostly avoided at the expense of ”good” corners not being hugged either, but this is generally a good trade-off. Several optional techniques have been proposed to produce more optimal paths based on the computational budget of applications.

There are many applications today where optimal paths are not required as long as the

Figure 5.1 Real-time games with many units on large maps with many obstacles can benefit significantly from conformal pathfinding techniques

paths are sufficiently good. One such application is real-time strategy games. In these games, many units are moving simultaneously over potentially large domains with many obstacles. In this application, the processing unit of the computer are required to handle real-time rendering of graphics, audio and simultaneously provide a responsive user interface to the player and many other operations. With such high demand, not having to compute full paths when pathfinding is a significant benefit.

The nature of real-time strategy games are also so that a major portion of pathfinding requests will be terminated before the unit has reached the requested destination. Players of these games often provide new path-requests for their units multiple times a second. With conventional A* every request must be fully processed before the unit can respond to the new request. With conformal pathfinding the unit can start to move almost immediately, and the path can refined over time as needed.

The value of this is especially apparent for multiplayer games that are hosted on dedicated servers. In this setting, the game publisher must pay for the server load, which includes the cost of pathfinding. This means that computational resources that are spent on computing paths that only will be overwritten seconds later is a significant cost, especially when considering games with a high number of units on large maps played by many players.

For these kind of applications, the shortest path isn't usually required as long as the path is sufficiently good, and as such, conformal pathfinding is a very promising alternative.

The code used in this project has not been optimized. Many data-structures are stored on the memory-heap as opposed to on the stack, resulting in much higher response times.

The source-code has also suffered from the experimental nature of the project, where techniques and methods have been come up with, and changed during the life-time of the project. The code has several redundant steps and lacks polishing. Despite all these disadvantages, the algorithm still competes with highly optimized A* code. The reference algorithm has been Unity3D's built-in pathfinding system which is written natively in C++, while the code written for this thesis is written in C#. In some cases, the response time of the conformal pathfinding code was faster than that of the built-in pathfinder, which speaks to the potential of conformal pathfinding.

5.1 Further work

5.1.1 Using continuous functions to generate the conformal graphs

While the implementation of circle packings and discrete Ricci-flow is simple, it has some limitations. Because of the nature of conformal mappings, complicated domains often require extreme warping to become circular. As a consequence, the difference between the largest and smallest circle radii can be computationally infeasible to realize. This is especially prominent in domains with elongated or narrow regions. Solutions to this problem has been found for the Schwarz-Christoffel mapping [1, 10] for simply-connected domains.

5.1.2 Modifying graphs in real-time

In some applications, the domain will change frequently. Obstacles might be added, modified or removed. Conventional A* algorithms have the possibility of updating the graph in real-time by adding, modifying or removing nodes to adapt to the changing environment. To compete with classical pathfinding techniques, conformal pathfinding should also have this option.

In many cases, it is not sustainable to do all the pre-processing steps all over again, so it should be a way of modifying the existing graph. Similarly to conventional A* it is possible to add and remove nodes from the graph representing the original domain. Since the discrete Ricci-flow is a stable algorithm, it is then possible to compute additional iterations of the algorithms to account for the changes. When the algorithm converge, the circle graph can be updated and the other pre-processing steps can also be done. It should be possible to do start using the new graph before all pre-processing steps are done. This will manifest in the application as paths gradually get better. As for the intermediate time while the new conformal map is processed, it is possible to use the original coordinates of the graph to perform conventional A*, might be noticed by the player as a slightly longer response time, but with modern computer the delay is barely noticeable, but if modern games adopt conformal pathfinding, it might also allow developers to create more complicated maps that is not sustainable by conventional A* algorithms.

5.1.3 More accurate modifications to support shortest paths

In this project I have discussed several methods to improve conformal pathfinding to get more optimal paths without using more resources than conventional pathfinding. Further research should focus on more precise and cost-efficient extensions to improve the quality of the paths generated.

5.1.4 Composite maps

Instead of having one map covering the entire map, it might be possible to divide the map into regions and have local conformal maps, that later is stitched together. This could allow for having more even distributions and avoiding the crowding problem. This would require the sub-maps to have a geometry in the virtual domain that allows stitching.

Bibliography

[1] T. A. Driscoll and S. A. Vavasis, “Numerical conformal mapping using cross-ratios and delaunay triangulation,” *SIAM J. Sci. Comput.*, vol. 19, pp. 1783–1803, nov 1998.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[3] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control*. Chichester, England: Wiley,, 2011.

[4] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, July 1968.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, “Correction to ”a formal basis for the heuristic determination of minimum cost paths”,” *SIGART Bull.*, pp. 28–29, Dec. 1972.

[6] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson Education Limited,, 2016.

[7] D. Marshall and S. Rohde, “Convergence of a variant of the zipper algorithm for conformal mapping,” *SIAM Journal on Numerical Analysis*, vol. 45, no. 6, pp. 2577–2609, 2007.

[8] H. Hakula, T. Quach, and A. Rasila, “Conjugate function method and conformal mappings in multiply connected domains,” *arXiv e-prints*, feb 2015.

[9] H. H. T. Q. A. Rasila, “Conjugate function method for numerical conformal mappings,” *Journal of Computational and Applied Mathematics*, vol. 237, no. 1, pp. 340 – 353, 2013.

[10] L. Trefethen, "Numerical computation of the schwarz–christoffel transformation," *SIAM Journal on Scientific and Statistical Computing*, vol. 1, no. 1, pp. 82–102, 1980.

[11] L. Howell and L. Trefethen, "A modified schwarz–christoffel transformation for elongated regions," *SIAM Journal on Scientific and Statistical Computing*, vol. 11, no. 5, pp. 928–949, 1990.

[12] K. Stephenson, "The approximation of conformal structures via circle packing," in *In Computational Methods and Function theory 1997, Proceedings of the third CMFT conference*, pp. 551–582, World Scientific, 1997.

[13] W. P. Thurston, "Three dimensional manifolds, kleinian groups and hyperbolic geometry," *Bull. Amer. Math. Soc. (N.S.)*, vol. 6, pp. 357–381, 05 1982.

[14] A. Gopal and L. N. Trefethen, "Representation of conformal maps by rational functions," *Numerische Mathematik*, pp. 1–24, 2019.

[15] D. Crowdy, "The schwarz–christoffel mapping to bounded multiply connected polygonal domains," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 461, no. 2061, pp. 2653–2678, 2005.

[16] T. K. DeLillo, A. R. Elcrat, and J. Pfaltzgraff, "Schwarz-christoffel mapping of multiply connected domains," *Journal d'Analyse Mathématique*, vol. 94, no. 1, pp. 17–47, 2004.

[17] J. Gao, X. D. Gu, and F. Luo, *Discrete Ricci Flow for Geometric Routing*, pp. 1–8. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.

[18] M. Naderan-Tahan and M. T. Manzuri-Shalmani, "Efficient and safe path planning for a mobile robot using genetic algorithm," in *2009 IEEE Congress on Evolutionary Computation*, pp. 2091–2097, May 2009.

[19] B. Rodin, D. Sullivan, *et al.*, "The convergence of circle packings to the riemann mapping," *Journal of Differential Geometry*, vol. 26, no. 2, pp. 349–360, 1987.

[20] L. J. Guibas and J. Hershberger, "Optimal shortest path queries in a simple polygon," *Journal of Computer and System Sciences*, vol. 39, no. 2, pp. 126–152, 1989.

Appendix

Digital appendix

The digital appendix contains the Matlab code used to create the conformal mappings used for spatial analysis. The Schwarz-Christoffel Toolbox provided by Tobin. A. Driscoll is required to run the code. [1]