Path Planning in Weighted Regions with Clearance Information

Angelos Kremyzas
GMT Capita Selecta, Utrecht University
a.kremyzas@students.uu.nl

March 25, 2014

Abstract

This work addresses the problem of navigating a disc-shaped character towards its destination within weighted planar polygonal regions while following smooth paths. We tackle this problem by extending the *Modified Indicative Routes and Navigation (MIRAN)* method to consider disc-shaped characters. First, we introduce a method for computing the cost of a linear disc traversal within a weighted planar triangulation of the scene. This traversal-cost computation method exploits the DCEL representation of the scene in order to efficiently detect the traversed weighted regions and compute the exact *overlap intervals* of those regions. An overlap interval between a moving disc and a region is defined by the position of the disc center during first and last occurring intersections between the disc and that specific region. Next, we propose modifications to the strategy MIRAN uses in order to compute candidate attraction points. Finally, we utilize the proposed traversal-cost computation method to select the attraction point among the candidates. This way, we introduce a method for path planning in weighted regions with clearance information. At the same time, we present a cost computation method for an arbitrary linear disc traversal in weighted regions. This method can be used as a black box by weighted region path planning algorithms.

Contents

Chapter 1

Introduction

Virtual environments often contain a variety of terrain types such as roads, sidewalks, grass and mud. For a simulation to be plausible it is necessary to employ path planning algorithms that take the nature of the underlying terrain type into account. A binary partitioning of the virtual scene into traversable and non-traversable regions might be inadequate in these cases. Individual preferences of a navigating character might dictate that the followed path should be more likely to avoid certain terrain types that are discomforting or unsafe, such as mud and dark alleys at night, compared to safer or more interesting ones, such as busy streets and recreational parks. In addition, the size of the character should also be considered since there should be enough clearance from obstacles throughout its navigation.
In this work, we address the problem of navigating a disc-shaped character through a weighted planar polygonal subdivision. Such a subdivision can resemble a heterogeneous virtual environment with different terrain types, and the character can have individual preferences for these terrain types. First, we introduce a method for computing the traversal cost of a disc that moves linearly between two points in the scene. Afterwards, we extend the work presented in [JCIG13] by using the new cost computation to account for disc-shaped characters.

1.1 Related Work

1.1.1 Path Planning With Clearance Information

Geraerts [Ger10] has introduced a data structure called *Explicit Corridor Map (ECM)* which can be used for path planning with an arbitrary amount of clearance from static obstacles. The ECM can also be employed for achieving local collision avoidance. The ECM is a generalized Voronoi diagram, with the Voronoi sites being the obstacles of the virtual scene. Each edge consists of a collection of bisectors, i.e. line segments and circular arcs, depending on the closest obstacles. Each part of an ECM edge that lies between two obstacle edges can be defined as a straight line segment. Parabolic arcs describe an ECM edge near the corners of the obstacles. At the endpoints of each bisector of a Voronoi edge, the ECM structure stores the closest points of the obstacles to the right and left of the edge. The ECM structure also stores the corresponding clearance value at these endpoints. The generalized Voronoi diagram is approximated by utilizing graphics hardware based on the technique by Hoff et al. [HKL$^+$99] that has been also used by Overmars and Geraerts [OG08]. Explicit Corridors extracted from the ECM are defined as a sequence of discs whose radius equals the maximum clearance from the obstacles. The left and right closest points that correspond to these discs serve as an explicit representation of the corridor's boundaries. The paper also describes how these corridors can be shrunk to allow finding shortest paths with an arbitrary minimum amount of clearance from obstacles.
Karamouzas et al. [KGO09] present a method for efficient real-time path planning for multiple virtual characters. The proposed technique extends previous work on corridor maps by Overmars and Geraerts [OG08] and addresses the limitation of the latter to generate convincing paths. Overmars and Geraerts [OG08] steer the virtual character using attraction points that lie on the medial axis of the environment. As a result, their method causes characters to keep equal distance from obstacles, which is not realistic. Karamouzas et al. [KGO09] tackle this problem. Initially, a path from the character's start position to its destination is either manually defined or generated using a planner. Such a path is referred to as an *Indicative Route*. It serves as a guide for the

virtual character. Next, the corridor map method is employed in order to compute the corridor containing this guiding path. At each step of the simulation, an attraction point is identified as the point of the indicative route that lies within the current clearance disc and has minimum curve distance from the goal. The curve distance is measured along the indicative route. The attraction point exerts an attraction force on the virtual character to steer it towards the goal. Additionally, repulsive forces are exerted from the boundaries of the corridor and from the dynamic obstacles on the character to avoid collisions. Random noise can further increase naturalness by inducing path variation. The computed positions and forces are numerically integrated using Verlet integration under a small time step to provide the final path. Experiments conducted on a cluttered virtual scene show that the method can be used to efficiently plan the motion of thousands of characters in real-time. A limitation of this method is that the character might take undesired shortcuts along the route. This happens when the current clearance disc intersects a part of the route that lies further ahead towards the goal. In such cases, the intersection point that is furthest along the route is picked as an attraction point. As a result, the attractive force causes the character to progress towards that point and to skip the intermediate part of the route.

Kallmann [Kal10] proposes a method for computing shortest paths with an arbitrary clearance from static obstacles. Kallmann introduces a navigation mesh called *Local Clearance Triangulation* (LCT). Initially, the navigable space is represented as a *Constrained Delaunay Triangulation (CDT)* of the segments that define the obstacles. This means that all segments that define the static obstacles are edges in the CDT, and the circumcircle of each triangle in the CDT contains no vertex in its interior that is visible from the edges of that triangle. For any traversed triangle that does not contain the start and goal position, the disc is considered to traverse the triangle by entering it through an edge and exiting it from another edge. Therefore, the LCT navigation mesh is defined as a CDT of the environment that stores on every edge the maximum allowed clearance for exiting that edge. The LCT is the outcome of incremental refinements of the original CDT by adding vertices to the triangulation whenever disturbances are detected to triangle traversals. The refinement process is repeated until no disturbances exist and the last generated CDT along with edge-exit clearance values defines the LCT of the environment. By employing A* on the LCT, a triangle sequence (*channel*) is found for connecting start and goal positions. Several metrics are proposed for guiding an A* search, which are then compared to each other. Next, Kallmann enhances a funnel algorithm to also account for clearance. This method is then used to compute shortest paths within the channel that A* produces. This method results in a path that is locally optimal, i.e. the generated path is shorter or equal to any other feasible path within the same channel. Should global optimality be required, Kallmann describes a process that gradually improves the locally optimal path by allowing multiple expansion fronts and running the enhanced funnel algorithm to each front. In this way, when all expansion fronts have terminated, the generated path that reaches the goal with minimum cost is guaranteed to be the globally optimal path.

Wein et al. [WvdBH05] propose a method for planning short paths with an arbitrary minimum amount of clearance from polygonal obstacles in a planar environment. Their method relies on a hybrid between the visibility graph and the Voronoi diagram of the scene, namely the $VV^{(c)}$-diagram, where c is the preferred clearance value. The authors present a straightforward method for computing the $VV^{(c)}$-diagram for a fixed c value. $VV^{(0)}$-diagram corresponds to the visibility graph. As c approaches ∞ the $VV^{(c)}$-diagram turns into the Voronoi diagram. In addition, the authors introduce a a data structure that is called the visibility-Voronoi complex. The visibility-Voronoi complex is constructed in a preprocessing phase and stores information on all edges that compose the $VV^{(c)}$-diagram along with their validity range in terms of clearance. The visibility-Voronoi complex can be queried for a pair of start and goal positions, and a clearance value. The query returns a short path that has been computed by performing a Dijkstra search on the implicitly generated graph. This way, an explicit computation of the $VV^{(c)}$-diagram is avoided, thus improving the overall running time of the method when multiple queries of different clearance values are performed.

All methods mentioned above consider homogeneous environments. The methods assume a binary subdivision of the virtual environment into traversable and non-traversable regions. Therefore, the cost of an arbitrary path is only dependent on its curve length. Thus, these methods cannot be employed in simulation scenarios or games where the scene is partitioned in weighted regions.

1.1.2 The Weighted Region Problem

The *Weighted Region Problem* (WRP) was introduced by Mitchell and Papadimitriou [MP91]. Given a subdivision of the navigation space into regions that are associated with weight values, a start position s and a goal position g, the WRP is defined as the problem of computing an optimal path from s to g. The measure of optimality in this case is the weighted length of a path, which in turn is defined by the sum of arc lengths of the path weighted by the weight value of the region that each arc overlaps. The path that connects s and g and has minimum weighted length is the optimal path. The authors use optimality concepts from optics and base their method on Snell's Law of Refraction. An optimal path should always consist of straight line segments within each intersecting region. The bending points of the path are located at the boundaries separating regions of different weight. However, the time complexity of the first approximated solution with respect to the number of vertices n that define the environment makes the application of the method impractical for real-time simulations ($O(n^8)$).
Aleksandrov et al. [ALMS98] propose a graph-search method that computes an approximation of the optimal path of a point in a weighted triangulated planar subdivision. Intermediate traversals of the path are discretized by placing Steiner points along the edges of the scene triangles. Steiner points are placed along the edges following a logarithmic distribution to capture the geometric aspects of the problem. Since this discretization approach would require an infinite number of Steiner points near the vertices of the triangulation, the concept of *vertex vicinity* is introduced. The vertex vicinity is a circle that is void of Steiner points, centered at each vertex. The radius of the vertex vicinity is determined by the ϵ-value of the approximation of the optimal path. The ϵ-approximation of the optimal path is computed by running Dijkstra's algorithm on the constructed graph. Further research by Aleksandrov et al. [AMS05] improves the running time of the Steiner point graph-search method by adding Steiner points on the bisectors of the triangulation.
Sun and Reif [SR01] propose an alternative algorithm called *BUSHWHACK*, that employs Steiner points for computing shortest paths. BUSHWHACK implements a lazy strategy for adding Steiner points to the graph. As a result, once a path connecting the start position s to the goal position g has been found, the algorithm selects this path as the optimal path. At the same time, the lazy nature of BUSHWHACK implies that all possible paths from s to any destination that have less weighted length than the optimal path from s to g are computed prior to the termination of the algorithm.
The *Modified Indicative Routes and Navigation (MIRAN)* method, as introduced by Jaklin et al. [JCIG13], addresses the problem of steering a *point* character towards its destination within a weighted planar scene while avoiding obstacles. The path that the character follows uses an *indicative route* as a guideline based on previous work by Karamouzas et al. [KGO09]. Jaklin et al. [JCIG13] follow a novel approach by allowing for individual preferences related to the types of terrain that a virtual character can have. Each region of the scene is assigned a weight value that describes the preference of the virtual character in traversing that region. The corresponding paths are planned by taking these preferences into consideration. The MIRAN method borrows from Overmars and Geraerts [OG08] the representation scheme of corridor maps with an arbitrary amount of clearance to allow for local collision avoidance. At the same time it utilizes an indicative route similar to Karamouzas et al. [KGO09] as a guide during the real-time navigation of the character. The indicative route is computed by running an A* algorithm on a grid representation of the scene. Guidance along the route is achieved by specifying and potentially adjusting in real-time a pair of parameter values: a) a shortcut value that describes the extent to which the next intermediate goal (attraction point) of the character can progress along the indicative route compared to a current reference position (reference point), and b) a sampling distance to determine the density of the candidate attraction points over this extent. The MIRAN method improves upon the work of Karamouzas et al. [KGO09] in taking weighted terrains into account and in letting the user control the smoothness of the final path.
All methods mentioned above consider point-character navigation. Therefore, these methods cannot be employed in cases where the character is modeled using a shape that bounds its silhouette, such as a disc. In the frame of this work, we investigate how the MIRAN method [JCIG13] can be adjusted so as to take clearance into account during path computation.

1.1.3 Collision Detection

Another important aspect of crowd simulations and games is to efficiently determine whether a moving character collides with static or dynamic obstacles. In the disc variant of the WRP,

collision detection is useful in determining whether the disc-shaped character overlaps a weighted region throughout its navigation. Extensive research has been conducted in the field of collision detection. An overview of the underlying concepts and current approaches can be found in the book of Ericson [Eri05] where emphasis is put on the real-time requirements of the problem.
A basic theorem that is frequently used in collision detection approaches is the *Separating Axis Theorem*. Given two non-colliding objects A and B, a separating axis of A and B is any vector $\vec{v}$ such as the projections of A and B on $\vec{v}$ do not overlap. For planar problems involving polygonal objects, the Separating Axis Theorem is stated as follows:

Separating Axis Theorem. *For any pair of non-intersecting polygons, there exists a separating axis that is orthogonal to an edge of each polygon.*

Current collision detection approaches often tackle the problem in three phases: the broad phase, the middle phase, and the narrow phase. During the broad phase, the objects of the virtual scene are modeled using primitive shapes that bound their silhouette. Besides model partitioning of the objects, space partitioning might also be performed during this phase in order to group objects together that are physically close to each other. During the middle phase, the primitive shapes are checked against each other to detect possible intersections. A commonly used primitive shape is an *Axis-Aligned Bounding Box (AABB)* since overlap tests between AABBs are the fastest to compute compared to other primitive shapes. However, AABBs can be a crude approximation of the bounded objects and can thus lead to false-positive collision detections. For this reason, the narrow phase employs finer approximations of the objects that were found during the middle phase and checks whether they collide. During the narrow phase, more expensive algorithms are employed to determine whether a collision between two objects occurs. Furthermore, during the narrow phase, algorithms can be employed to compute the penetration depth and use it later in collision resolution. A common choice for the narrow phase is the GJK algorithm [GJK88], which determines whether two objects collide based on their Minkowski sum (see Section 1.1.4). If the objects collide, the GJK algorithm computes the penetration depth. If the objects do not collide, the GJK algorithm computes their separating distance.
Furthermore, when the virtual environment includes objects or characters that consist of more complex shape combinations, then the *Bounding Volume Hierarchies (BVH)* structure can be used during model partitioning. BVH consists of a tree structure where leaves correspond to primitive shapes, and each node corresponds to a volume that bounds the union of its children. By following this hierarchical modeling approach, collision detection tests between children of BVH trees are only performed if their parents collide with each other.
In the frame of our work, the weighted regions that define the environment are triangulated. We introduce a cost computation method for an arbitrary linear traversal within weighted regions. Our method, requires collision detection tests in order to compute when a disc enters or exits a scene triangle. For this reason, we define two collision detection tests. The first test determines whether a disc intersects a triangle edge. The second test determines whether a box intersects an edge and is similar to the AABB collision detection test.

1.1.4 Minkowski Sums

Given a set of points A and a set of points B, the *Minkowski Sum* $A \oplus B$ is defined by the following equation:

$$A \oplus B = \{\mathbf{a} + \mathbf{b} : \mathbf{a} \in A, \mathbf{b} \in B\}, \tag{1.1}$$

where $\mathbf{a}$ and $\mathbf{b}$ are the position vectors of a point in A and B, respectively. A visualization of the Minkowski sum $A \oplus B$, where A is a disc and B is a triangle, is given in Figure 1.1. A basic property is that the Minkowski sum of two convex shapes is also convex. For the weighted region problem that also considers clearance, the Minkowski sum operation $A \oplus B$ defines the set of all possible points on a plane where a shape A can be positioned in order to intersect shape B. As an example, in Figure 1.1 when disc A is centered at any point of $A \oplus B$, it intersects triangle B.

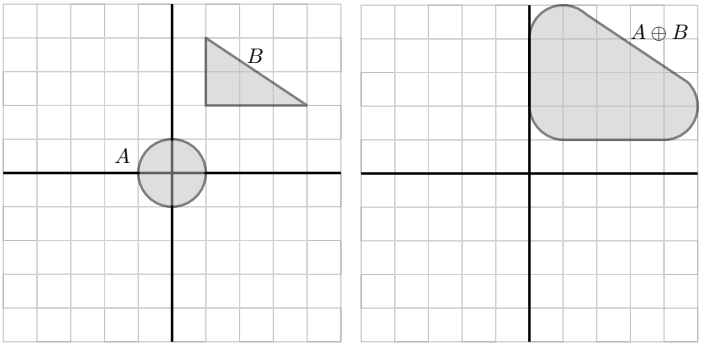


Figure 1.1: (a) A disc A and a triangle B on a plane, and (b) The Minkowski sum $A \oplus B$.

If the Minkowski sum operation between a disc-shaped character and every weighted subdivision of the environment is performed, then a WRP that considers disc navigation can be restated as a WRP that considers point navigation. As Figure 1.1 indicates, conic arcs are created for every vertex of a polygon. For the disc variant of the WRP, the disc-offsetting operation should be performed against every polygon that represents a weighted region. Next, the overlay of the inflated regions must be computed based on a weighting scheme. Finally, a WRP path planner that considers point navigation can be applied on the overlay of the weighted scene. Following this approach also requires approximating every conic arc before applying a WRP path planning method that assumes a polygonal representation of the environment. The method we propose does not follow this approach. As a result, our method does not rely on a representation of the environment that is dependent on the disc radius. More information on the Minkowski sum operation can be found in the book of Ericson [Eri05].

1.2 Report Structure

The rest of the report is structured as follows: In Section 2.1, the details of the MIRAN method are provided. This method serves as a basis in our analysis. In Section 2.2, the DCEL representation scheme of a polygonal planar subdivision is discussed. This representation is used by our method. In Section 3, we propose a method for computing the cost of an arbitrary linear traversal of a disc within a weighted triangulated planar environment. This method is then used in Section 4, where the MIRAN method is extended to consider navigation of a character that is modeled as a disc. In Section 5, the proposed methods and future work are discussed.

Chapter 2

Background

2.1 The MIRAN Method

This section discusses the details of the MIRAN method, as introduced by Jaklin et al. [JCIG13]. This method serves as a basis for our analysis. The problem of path planning in weighted regions with clearance information is tackled by extending the MIRAN method so as to consider disc-shaped characters. The MIRAN method borrows from Overmars and Geraerts [OG08] the representation scheme of *corridor maps* with an arbitrary amount of clearance to allow for local collision avoidance. At the same time it utilizes an *indicative route* similar to Karamouzas et al. [KGO09] as a guide during the real-time navigation of the character.

Algorithm 1 The MIRAN Method

Input: Start s, goal g, indicative route from s to g
Output: Smooth terrain-dependent path from s to g

```
i ← 0
x_0 ← s
while x_i ≠ g do
    r_i ← ComputeReferencePoint(x_i)
    A_i ← ComputeCandidateAttractionPoints(r_i, x_i)
    α_i ← PickBestCandidate(A_i, x_i)
    x_{i+1} ← MoveCharacterTowardsAttractionPoint(r_i, x_i)
    i ← i + 1
end while
```

Figure 2.1: The MIRAN Method Overview [JCIG13]. At each time step i let: r_i denote the reference point, $\mathcal{A}_i$ the set of candidate attraction points, α_i the selected attraction point and x_i position of the character.

Figure 1 presents an overview of the algorithm that implements the MIRAN method. The MIRAN method requires that an indicative route has been either sketched or computed by another method. The indicative route, $\pi_{ind} : [0, 1] \rightarrow \mathbb{R}^2$, consists of a continuous curve path that has the starting character position s and the goal character position g as its endpoints. Given this route, the MIRAN method uses it as a guideline and is responsible for moving the character throughout the simulation. The simulation terminates when the goal has been reached.
In order to keep track of the character's progress along the indicative route, a *reference point* r_i is computed on the route for each simulation step i. Furthermore, in order to attract the character towards its goal, an *attraction point* a_i along the indicative route is selected from a *list of candidate attraction points* $\mathcal{A}_i$. The reference point r_i is defined as the point on the indicative route that minimizes the Euclidean distance from the actual character position x_i while lying between the previous reference point r_{i-1} and previous attraction point α_{i-1}. This way, the character is prevented from taking undesired shortcuts along the route.
For the computation of the candidate attraction points, a *shortcut parameter* σ and a *sampling distance* d are employed. The shortcut parameter σ is defined as the maximum curve length distance between the reference point r_i and a candidate attraction point. The sampling distance d is defined as the maximum curve length distance between two consequent candidate attraction

points. Both curve length distances are measured along the indicative route π_{ind}. In order to compute the list of candidate attraction points $\mathcal{A}_i$, the method considers the curve segment of the indicative route that starts at reference point r_i and ends at a curve length distance σ towards the goal, at point σ_i. The parts of this curve segment that are not visible by the actual position of the character x_i are discarded. Non-visible parts of the route are discarded, since attracting the character towards an arbitrary point on those parts would cause the character to collide with scene obstacles. Next, for each of the remaining curve segments j, candidate attraction points are placed at its endpoints $\pi_{ind}(a_j)$ and $\pi_{ind}(b_j)$. The interior of each curve segment j is sampled: starting from the endpoint $\pi_{ind}(a_j)$, candidate points are placed every d curve length distance until endpoint $\pi_{ind}(b_j)$ has been reached. Also, the reference point r_i is not considered a candidate in case more candidates exist. This is done to prevent the character from remaining stationary. Figure 2.2 gives an example of the candidate attraction points that are computed during the i-th step of a simulation.

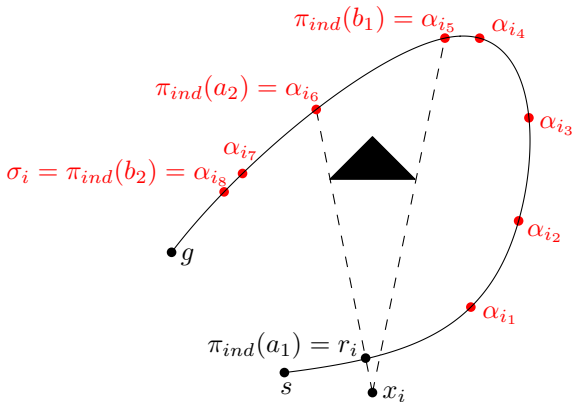


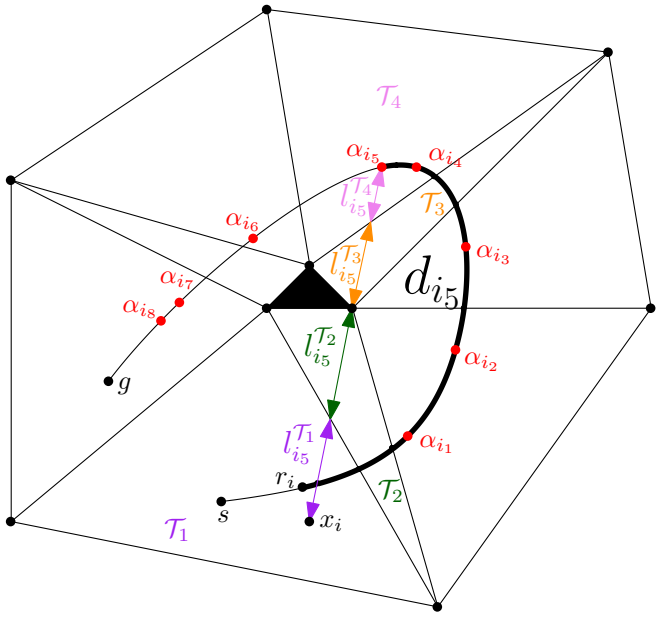
Figure 2.2: Example of candidate attraction points computed by MIRAN method [JCIG13]. The *black* triangle represents a static obstacle of the scene.

The next step is to select an attraction point α_i from the k candidates that have been computed during simulation step i. For this reason, a weight function ω is defined for estimating the cost of traversing from the actual character position x_i to each candidate attraction point α_{i_j}, where $1 \leq j \leq k$. For each line segment $l(\alpha_{i_j}, x_i)$, let $\mathcal{T}_{i_j}$ denote the set of all terrain types that $l(\alpha_{i_j}, x_i)$ intersects. Also, $w(\mathcal{T}) > 0$ defines the character's traversal preference over any terrain type $\mathcal{T} \in \mathcal{T}_{i_j}$. Let d_{i_j} be the curve length distance along π_{ind} from the reference point r_i to the candidate attraction point α_{i_j}. For each terrain type $\mathcal{T}$, let $l_{i_j}^{\mathcal{T}}$ be the total summed length of the parts of $l(\alpha_{i_j}, x_i)$ that lie on $\mathcal{T}$. Then the cost $\omega(l(\alpha_{i_j}, x_i))$ of a candidate attraction point α_{i_j} is defined as:

$$\omega(l(\alpha_{i_j}, x_i)) = \frac{1}{d_{i_j}} \sum_{\mathcal{T} \in \mathcal{T}_{i_j}} w(\mathcal{T}) l_{i_j}^{\mathcal{T}} \ . \tag{2.1}$$

The candidate attraction point α_{i_m} with the minimum ω cost value is chosen as an attraction point α_i, i.e. $\alpha_i = \alpha_{i_m}$, where $m = \underset{1 \leq j \leq k}{\operatorname{argmin}} \ \omega(l(\alpha_{i_j}, x_i))$. Figure 2.3 shows the computation of the ω value of the fifth candidate attraction point α_{i_5} for the example scene of Figure 2.1.

Finally, a force-based model similar to Karamouzas et al. [KGO09] is used to update the position of the character. This involves three force factors: a) an attractive term that steers the character towards the attraction point, b) a repulsive term that pushes the character away from the closest scene boundary, and c) another repulsive term that achieves local collision avoidance between the character and other characters or small obstacles. For further details the reader is referred to Jaklin et al. [JCIG13], where the MIRAN method is introduced, as well as its prior related work by Overmars and Geraerts [OG08], and Karamouzas et al. [KGO09].



$$\omega(l(\alpha_{i_5}, x_i) = \frac{1}{d_{i_5}}\left(w(\mathcal{T}_1)l_{i_5}^{\mathcal{T}_1} + w(\mathcal{T}_2)l_{i_5}^{\mathcal{T}_2} + w(\mathcal{T}_3)l_{i_5}^{\mathcal{T}_3} + w(\mathcal{T}_4)l_{i_5}^{\mathcal{T}_4}\right)$$

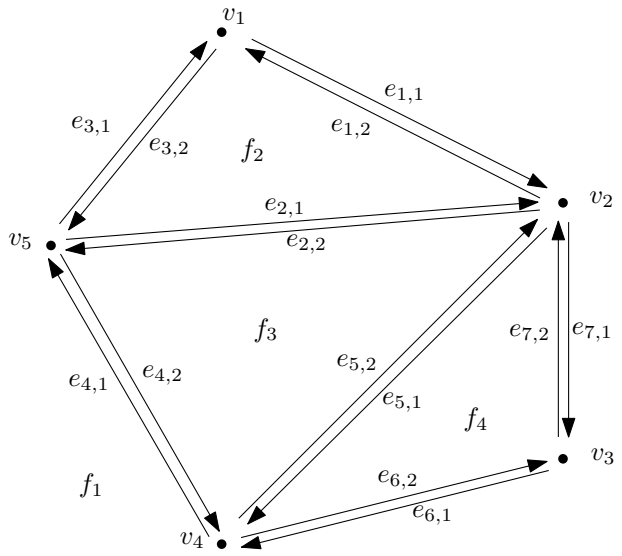
Figure 2.3: Computing $\omega(l(\alpha_{i_5}, x_i))$. In this example, each of the four triangles that $l(x_i, \alpha_{i_5})$ intersects is associated with a different terrain type and is illustrated using a different color (*purple*, *green*, *orange* and *violet*). The *thick* curve segment has length d_{i_5}.

2.2 The DCEL Representation of a Planar Subdivision

A scene in a planar WRP typically consists of a set of non overlapping polygons where each polygon is associated with a weight value. It is convenient to employ a planar subdivision representation that allows for efficient queries to the components of the scene. A *vertex* corresponds to a node in a graph subdivision of the scene. An *edge* corresponds to an arc connecting two nodes. A *face* corresponds to a maximal subset of the plane that does not intersect any edge or vertex. De Berg et al. [DBVKOS08] define a *doubly connected edge list* (DCEL) as a representation that contains a record for each vertex, edge and face of a planar subdivision. We consider each edge as a pair of opposite-directed *half-edges*, with each half-edge bounding a different face of the subdivision. A pair of half-edges that correspond to the same edge are considered *twins* to each other. For consistency, a half-edge is considered to bound the face on its left when walking along the direction of the half-edge.
The DCEL representation allows for storing additional information that does not deal with the geometry or topology of the subdivision. This is possible by employing an extra record field called *attribute information*. For the weighted region problem, it is meaningful to store in the attribute information field the terrain type or its corresponding weight value for each face.
The DCEL represantation of a planar subdivision stores the following information:

- For each vertex v, a record stores the planar *coordinates* of v, and a pointer $IncidentEdge(v)$ to an arbitrary half-edge that has v as its origin.
- For each face f, a record stores a pointer $OuterComponent(f)$ to an arbitrary half-edge that lies on the outer boundary of f. If f is the outer face of the subdivision, then $OuterComponent(f)$ is **nil**. For each hole of f, the face record also stores a pointer, $InnerComponent(f)$ to an arbitrary half-edge that bounds f with that hole. If f has no holes, then $InnerComponents(f)$ are **nil**.
- For each half-edge e, a record stores a pointer to its origin, $Origin(e)$, a pointer to its twin half-edge, $Twin(e)$, and a pointer to the face it bounds, $IncidentFace(e)$. The record also stores a pointer to the next half-edge along the boundary of the incident face, $Next(e)$, and a pointer to the previous half-edge along the boundary of the incident face, $Prev(e)$.

Figure 2.4 shows the information stored in the vertex, face and half-edge records of the DCEL representation of an example scene. In this example, the scene is a weighted triangulated planar subdivision where a different region type is associated with each face of the subdivision. More information on the DCEL representation can be found in the book of De Berg et al. [DBVKOS08].



Vertex	**Coordinates**	**IncidentEdge**
v_1	(x_1, y_1)	$e_{1,1}$
v_2	(x_2, y_2)	$e_{7,1}$
v_3	(x_3, y_3)	$e_{6,1}$
v_4	(x_4, y_4)	$e_{4,1}$
v_5	(x_5, y_5)	$e_{3,1}$

face	**OuterComponent**	**InnerComponents**	**AttributeInformation**
f_1	**nil**	$e_{1,1}$	$TerrainType1$
f_2	$e_{3,2}$	**nil**	$TerrainType2$
f_3	$e_{4,2}$	**nil**	$TerrainType3$
f_4	$e_{6,2}$	**nil**	$TerrainType4$

Half-Edge	**Origin**	**Twin**	**IncidentFace**	**Next**	**Prev**
$e_{1,1}$	v_1	$e_{1,2}$	f_1	$e_{7,1}$	$e_{3,1}$
$e_{1,2}$	v_2	$e_{1,1}$	f_2	$e_{3,2}$	$e_{2,1}$
$e_{2,1}$	v_5	$e_{2,2}$	f_2	$e_{1,2}$	$e_{3,2}$
$e_{2,2}$	v_2	$e_{2,1}$	f_3	$e_{4,2}$	$e_{5,2}$
$e_{3,1}$	v_5	$e_{3,2}$	f_1	$e_{1,1}$	$e_{4,1}$
$e_{3,2}$	v_1	$e_{3,1}$	f_2	$e_{2,1}$	$e_{1,2}$
$e_{4,1}$	v_4	$e_{4,2}$	f_1	$e_{3,1}$	$e_{6,1}$
$e_{4,2}$	v_5	$e_{4,1}$	f_3	$e_{5,2}$	$e_{2,2}$
$e_{5,1}$	v_2	$e_{5,2}$	f_4	$e_{6,2}$	$e_{7,2}$
$e_{5,2}$	v_4	$e_{5,1}$	f_3	$e_{2,2}$	$e_{4,2}$
$e_{6,1}$	v_3	$e_{6,2}$	f_1	$e_{4,1}$	$e_{7,1}$
$e_{6,2}$	v_4	$e_{6,1}$	f_4	$e_{7,2}$	$e_{5,1}$
$e_{7,1}$	v_2	$e_{7,2}$	f_1	$e_{6,1}$	$e_{1,1}$
$e_{7,2}$	v_3	$e_{7,1}$	f_4	$e_{5,1}$	$e_{6,2}$

Figure 2.4: DCEL example on a weighted triangulated planar scene.

Chapter 3

Traversal-Cost Computation

3.1 Introduction

In this chapter, we propose a method for computing the cost of an arbitrary linear traversal of a disc in a weighted triangulated planar subdivision. A basic assumption of the proposed method is that at any location of the disc, the current traversal cost is determined by the overlapping triangle with the maximum weight. When a *point*-character moves linearly from a starting point s towards an ending point g, then the traversal cost $c_{\vec{sg}}$ can be computed by the following equation:

$$c_{\vec{sg}} = \sum_{i=1}^{n} w_i * l_i, \tag{3.1}$$

where t_i denotes the i-th triangle that the line segment sg intersects, w_i denotes the weight of t_i, l_i denotes the length of the *overlap interval* of the moving point with t_i throughout $\vec{sg}$ traversal, and n is the number of overlapping triangles (see Figure 3.1). An overlap interval between a point-character and a triangle is defined by the character position during first and last occurring intersections between the character and that specific triangle.

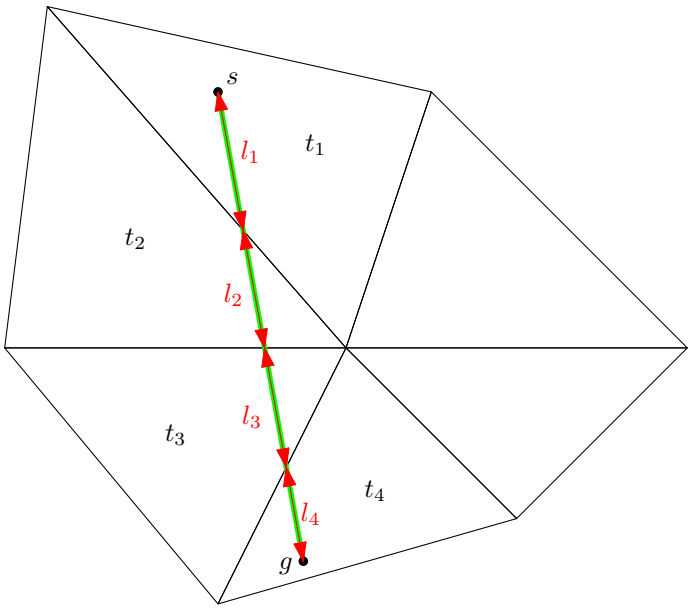


Figure 3.1: Overlap intervals l_1, l_2, l_3 and l_4 (*red*) of a moving point with triangles t_1, t_2, t_3 and t_4 respectively throughout $\vec{sg}$ traversal(*green*).

However, Equation (3.1) does not hold when clearance is considered. Figure 3.2 presents an instance of a *disc*-shaped character performing the $\vec{sg}$ traversal of Figure 3.1. In this case, the disc also overlaps with t_2 although the disc center lies within t_1. Therefore if $w_2 > w_1$, then our basic assumption dictates that t_2 currently determines the traversal cost of the disc. As a result, a

different approach is required when computing the traversal cost of a disc in a weighted scene.

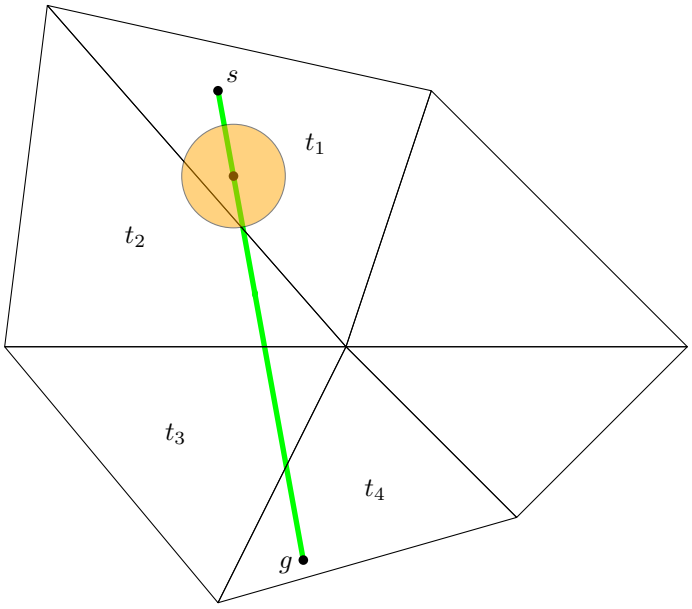
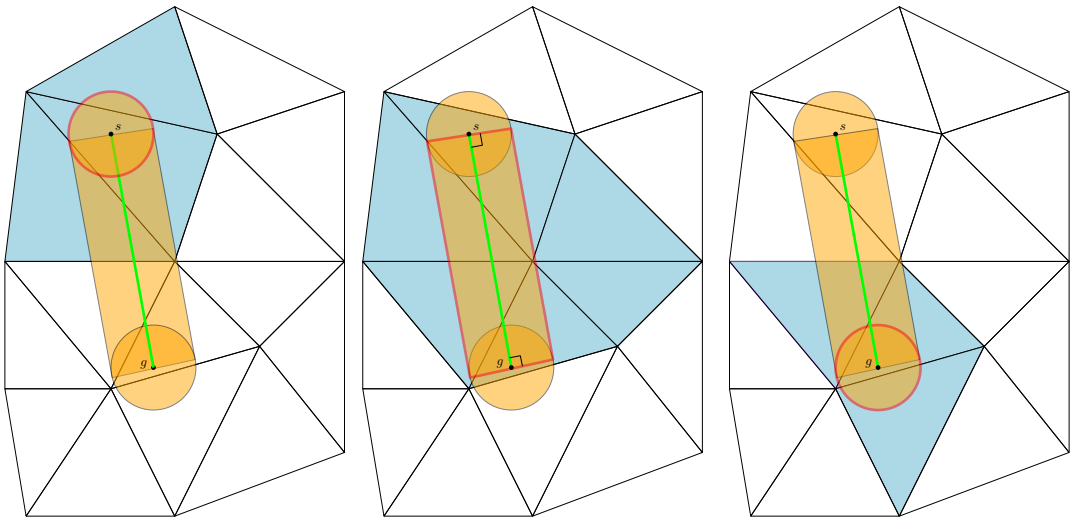


Figure 3.2: An instance of a disc (*orange*) moving along the $\vec{sg}$ line segment (*green*) in the same scene as Figure 3.1.

In order to identify all intersecting triangles during a linear disc traversal, we follow a sweeping disc approach that is split into three phases: the starting phase, the intermediate phase, and the ending phase. During the starting phase, we find all triangles that the disc intersects when centered at start position s (Figure 3.3a). During the intermediate phase, we find all triangles that the intermediate box of the sweeping disc intersects (Figure 3.3b). During the ending phase, we find all triangles that the disc intersects when centered at goal position g (Figure 3.3c). The union of the triangles that are found during the three phases equals the set of triangles that the sweeping disc intersects. It should be noted that there is an overlap between the three detection phases. In addition, the box of the intermediate phase does not correspond to an exact bounding shape of the sweeping disc during that phase. Nevertheless, this scheme is expected to be more efficient than a more sophisticated segmentation of the sweeping disc, while still guaranteeing that no intersecting triangle will be ignored during the cost computation.



(a) Starting phase **(b)** Intermediate phase **(c)** Ending phase

Figure 3.3: Sweeping disc (*orange*) in a triangulated scene. Triangles in *light blue* intersect the disc throughout the three phases of its traversal (*red*).

The rest of the chapter is divided as follows: In Section 3.2, we present the geometrical aspects that describe disc-triangle intersections during a linear traversal from an arbitrary starting point s to an arbitrary ending point g. Relevant equations are formulated and general solutions are devised for computing disc-triangle overlap intervals. In Section 3.3, we provide the details of the algorithm that is responsible for computing the $\vec{sg}$ traversal cost. The solutions devised in Section 3.2 are employed by the internal functions of the algorithm (see Algorithm 5 in Section 3.3.2).

3.2 Problem Description

A disc of radius r moves linearly from an arbitrary starting point s towards an arbitrary goal point g. The *primary axis* of our analysis is defined by vector $\vec{sg}$. The *secondary axis* is perpendicular to the primary axis and points to the left when following $\vec{sg}$, i.e. when moving along the direction of traversal in 2D space. The starting point s defines the *origin* of the coordinate system. Let $\mathbf{p}$ denote the position vector $\mathbf{p} = (x_P, y_P)^T$ of an arbitrary point P, where x_P is the coordinate of that point on the primary axis and y_P is the coordinate of that point on the secondary axis. Figure 3.4 gives an illustration of the coordinate system and the position vector $\mathbf{p}$.

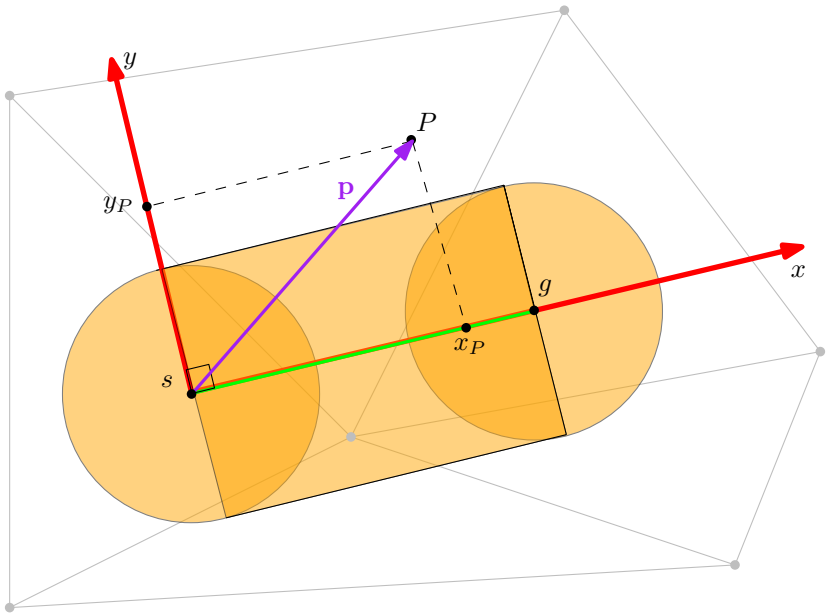


Figure 3.4: Coordinate system (*red*) and position vector $\mathbf{p}$ (*purple*) of an arbitrary point P during linear traversal (*green*) from starting point s to ending point g of a disc (*orange*) in a triangulated scene (*grey*).

If we assume a sweeping disc with its center moving across the line of motion sg, then each intersecting triangle edge or vertex can have at most two single point intersections with the disc along its transition. For each triangle ABC that is intersected by the sweeping disc, the disc enters the triangle by touching one of its vertices A,B,C or one of its edges AB,BC,AC. For each edge or vertex intersection with the sweeping disc, the *first* occurring intersection point P_{in} should lie on the perimeter of the disc centered at $P_{d,in}$. Similarly, the *last* occurring intersection point P_{out} should lie on the perimeter of the disc centered at $P_{d,out}$. Therefore:

$$\|\mathbf{p}_{in} - \mathbf{p}_{d,in}\| = r \tag{3.2}$$

and

$$\|\mathbf{p}_{out} - \mathbf{p}_{d,out}\| = r \tag{3.3}$$

Figure 3.5 gives an example of first and last disc-edge intersections.

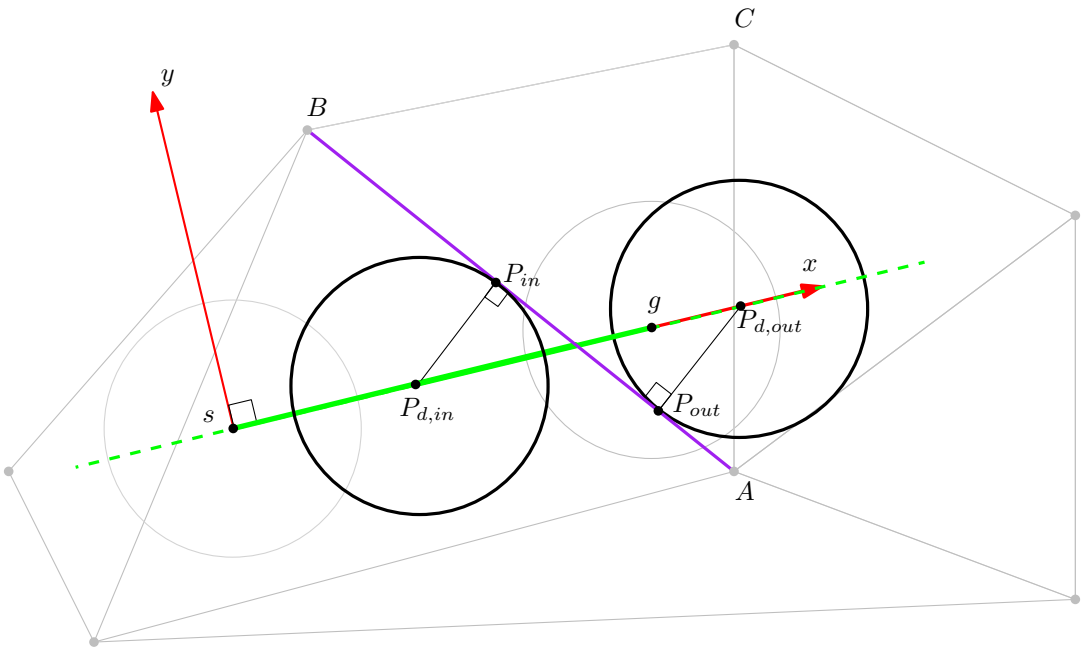


Figure 3.5: First and last occurring intersections of the disc with edge AB (*purple*) along line of motion sg (*green*). The disc first intersects edge AB at P_{in} when centered at $P_{d,in}$ and it last intersects AB at P_{out} when centered at $P_{d,out}$.

The x-projection of the first occurring intersection point should be larger than that of the disc center:

$$x_{in} > x_{d,in} \tag{3.4}$$

The x-projection of the last occurring intersection point should be smaller than that of the disc center:

$$x_{out} < x_{d,out} \tag{3.5}$$

During a disc-edge or disc-vertex intersection, the disc center is constrained to lie within the transition line segment sg. Therefore:

$$\mathbf{p}_{d,in} = (1 - t_{sg})\mathbf{p}_s + t_{sg}\mathbf{p}_g, \text{ where } t_{sg} \in [0, 1] \tag{3.6}$$

and

$$\mathbf{p}_{d,out} = (1 - t_{sg})\mathbf{p}_s + t_{sg}\mathbf{p}_g, \text{ where } t_{sg} \in [0, 1] \tag{3.7}$$

Since $\mathbf{p}_s = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and $\mathbf{p}_d = \begin{pmatrix} x_d \\ 0 \end{pmatrix}$, equations (3.6) and (3.7) can be rewritten as:

$$x_{d,in} \in [0, x_g] \tag{3.8}$$

and

$$x_{d,out} \in [0, x_g] \tag{3.9}$$

Equations (3.6), (3.7), (3.8) and (3.9) should be regarded as constraints. These equations can be used as validation checks to determine if a detected intersection occurs within the traversal segment sg or if it occurs on one of its extensions. In the latter case, the overlap interval $[x_{in}, x_{out}]$ should be pruned. As an example, in Figure 3.5 equations (3.6) and (3.8) apply for the first occurring disc-edge intersection, but equations (3.7) and (3.9) do not apply for last occurring disc-edge intersection. Therefore, we conclude that only the first occurring disc-edge intersection is valid with respect to an sg traversal.

At the first and last disc-edge intersections, the disc either touches one of the edge endpoints or the edge is tangent to the disc. We treat each case separately below.

In case of first or last occurring intersection between the disc and vertex A at P_A, this point should satisfy equation (3.2) or (3.3), respectively. The same equations apply to first and last occurring disc-triangle intersection with the rest of the triangle vertices, B and C, if such intersections exist.

In case of edge AB being tangent to the disc, the following pair of equations is obtained for describing first and last occurring edge intersections:

$$(\mathbf{p}_{in} - \mathbf{p}_{d,in}) \cdot (\mathbf{p}_B - \mathbf{p}_A) = 0 \tag{3.10}$$

and

$$(\mathbf{p}_{out} - \mathbf{p}_{d,out}) \cdot (\mathbf{p}_B - \mathbf{p}_A) = 0 \tag{3.11}$$

See Figure 3.5 for an example of first and last disc-edge intersections where the edge is tangent to the disc.
In case of an intersection with edge AB, the first and last occurring disc-edge intersection points should lie on edge AB, thus we obtain:

$$\mathbf{p}_{in} = (1 - t_{AB})\mathbf{p}_A + t_{AB}\mathbf{p}_B, \text{ where } t_{AB} \in [0, 1] \tag{3.12}$$

and

$$\mathbf{p}_{out} = (1 - t_{AB})\mathbf{p}_A + t_{AB}\mathbf{p}_B, \text{ where } t_{AB} \in [0, 1] \tag{3.13}$$

Similar equations describe first and last occurring intersections with the rest of the triangle edges, AC and BC, if such intersections exist.

3.2.1 Disc-Vertex First and Last Occurring Intersection Points

Let the disc intersect vertex A of triangle ABC. Then, the first and last occurring intersections between the disc and vertex A are computed.
The position $\mathbf{p}_{d,in}$ of the disc center during the first occurring intersection with vertex A is obtained by solving the system of equations (3.2), (3.4), (3.8) when $\mathbf{p}_{in} = \mathbf{p}_A$.
In detail, Equation (3.2) for $\mathbf{p}_{in} = \mathbf{p}_A$ gives:

$$\|\mathbf{p}_A - \mathbf{p}_{d,in}\| = r$$

The equation above can be rewritten in matrix form:

$$\| \begin{pmatrix} x_A \\ y_A \end{pmatrix} - \begin{pmatrix} x_{d,in} \\ 0 \end{pmatrix} \| = r$$

Solving last equation for $x_{d,in}$ gives:

$$\| \begin{pmatrix} x_A - x_{d,in} \\ y_A \end{pmatrix} \| = r \Leftrightarrow$$

$$(x_A - x_{d,in})^2 + y_A^2 = r^2 \Leftrightarrow$$

$$x_A^2 + {x_{d,in}}^2 + 2x_A x_{d,in} + y_A^2 = r^2 \Leftrightarrow$$

$${x_{d,in}}^2 + (2x_A)x_{d,in} + (x_A^2 + y_A^2 - r^2) = 0$$

The above quadratic equation has real solutions for $x_{d,in}$, if and only if the discriminant is non-negative:

$$(2x_A)^2 - 4 * 1 * (x_A^2 + y_A^2 - r^2) \geq 0 \Leftrightarrow$$

$$-4y_A^2 + 4r^2 \geq 0 \Leftrightarrow$$

$$r^2 \geq y_A^2 \Leftrightarrow$$

$$-r \leq y_A \leq r$$

If $y_A \in [-r, r]$, then the roots of the equation are:

$$x_{d,in} = -\frac{x_A}{2} \pm \sqrt{r^2 - y_A^2} \tag{3.14}$$

The smallest root of equation (3.14) that satisfies equations (3.4) and (3.8) is accepted for the first occurring intersection of the disc with vertex A.
The position $\mathbf{p}_{d,out}$ of the disc center during the last occurring intersection with vertex A is obtained by solving the system of equations (3.3), (3.5), (3.9) when $\mathbf{p}_{out} = \mathbf{p}_A$. Similar to the computation of $\mathbf{p}_{d,in}$ described above, it can be proven that if $y_A \in [-r, r]$, then:

$$x_{d,out} = -\frac{x_A}{2} \pm \sqrt{r^2 - y_A^2} \tag{3.15}$$

The largest root of equation (3.15) that satisfies equations (3.5) and (3.9) is accepted for the last occurring intersection of the disc with vertex A.

3.2.2 Disc-Edge First and Last Occurring Intersection Points

Let the disc intersect edge AB of triangle ABC. Then, the first and last occurring intersections between the disc and edge AB are computed.
The position $\mathbf{p}_{d,in}$ of the disc center during the first occurring intersection with edge AB is obtained by solving the system of equations (3.2), (3.4), (3.8), (3.10) and (3.12).
In detail, Equation (3.2) written in matrix form gives:

$$\begin{gathered}
\| \begin{pmatrix} x_{in} \\ y_{in} \end{pmatrix} - \begin{pmatrix} x_{d,in} \\ 0 \end{pmatrix} \| = r \Leftrightarrow \\
\| \begin{pmatrix} x_{in} - x_{d,in} \\ y_{in} \end{pmatrix} \| = r \Rightarrow \\
(x_{in} - x_{d,in})^2 + {y_{in}}^2 = r^2 \Leftrightarrow \\
|x_{in} - x_{d,in}| = \sqrt{r^2 - {y_{in}}^2}
\end{gathered} \tag{3.16}$$

Also, Equation (3.10) written in matrix form gives:

$$\begin{gathered}
\begin{pmatrix} x_{in} - x_{d,in} \\ y_{in} \end{pmatrix} \cdot \begin{pmatrix} x_B - x_A \\ y_B - y_A \end{pmatrix} = 0 \Rightarrow \\
(x_{in} - x_{d,in})(x_B - x_A) + y_{in}(y_B - y_A) = 0 \Leftrightarrow \\
(x_{in} - x_{d,in})(x_B - x_A) = -y_{in}(y_B - y_A) \Leftrightarrow \\
(x_{in} - x_{d,in})^2 (x_B - x_A)^2 = y_{in}^2 (y_B - y_A)^2 \Leftrightarrow \\
|x_{in} - x_{d,in}|^2 (x_B - x_A)^2 = y_{in}^2 (y_B - y_A)^2
\end{gathered} \tag{3.17}$$

By substituting Equation (3.16) in Equation (3.17) and solve for y_{in} we obtain:

$$\begin{gathered}
(r^2 - y_{in}^2)(x_B - x_A)^2 = y_{in}^2 (y_B - y_A)^2 \Leftrightarrow \\
[(x_B - x_A)^2 + (y_B - y_A)^2] y_{in}^2 = r^2 (x_B - x_A)^2 \Leftrightarrow \\
y_{in}^2 = \frac{r^2 (x_B - x_A)^2}{(x_B - x_A)^2 + (y_B - y_A)^2} \Leftrightarrow \\
y_{in} = \pm \frac{r|x_B - x_A|}{\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}} \Leftrightarrow \\
y_{in} = \pm \frac{r|x_B - x_A|}{|AB|}
\end{gathered} \tag{3.18}$$

As a result, y_{in} is computed using Equation (3.18).
In order to compute x_{in}, Equation (3.12) is written in matrix form:

$$\begin{pmatrix} x_{in} \\ y_{in} \end{pmatrix} = (1 - t_{AB}) \begin{pmatrix} x_A \\ y_A \end{pmatrix} + t_{AB} \begin{pmatrix} x_B \\ y_B \end{pmatrix}, t_{AB} \in [0, 1]$$

From the matrix equation above, the following system of equations is obtained:

$$\begin{cases} x_{in} = (1 - t_{AB}) x_A + t_{AB} x_B \\ y_{in} = (1 - t_{AB}) y_A + t_{AB} y_B \\ t_{AB} \in [0, 1] \end{cases} \Leftrightarrow$$

$$\begin{cases} x_{in} = (1 - t_{AB}) x_A + t_{AB} x_B \\ y_{in} - y_A = t_{AB} (y_B - y_A) \\ t_{AB} \in [0, 1] \end{cases} \Leftrightarrow$$

$$x_{in} = (1 - t_{AB}) x_A + t_{AB} x_B, \tag{3.19a}$$

$$t_{AB} = \frac{y_{in} - y_A}{y_B - y_A}, \tag{3.19b}$$

$$t_{AB} \in [0, 1], \tag{3.19c}$$

In order to find x_{in}, we first replace y_{in} from (3.18) in (3.19b). The obtained t_{AB} values are validated against (3.19c) and those accepted are used in (3.19a) to compute x_{in}. Thus, $\mathbf{p}_{in} = (x_{in}, y_{in})^T$ is computed. Similar to Section 3.2.1, it can be proven that, if $y_{in} \in [-r, r]$, then:

$$x_{d,in} = -\frac{x_{in}}{2} \pm \sqrt{r^2 - y_{in}^2} \tag{3.20}$$

The smallest root of equation (3.20) that satisfies equations (3.4) and (3.8) is accepted for the first occurring intersection of the disc with edge AB.
The position $\mathbf{p}_{d,out}$ of the disc center during the last occurring intersection with edge AB is obtained by solving the system of equations (3.3), (3.5), (3.9), (3.11) and (3.13). Similar to the computation of $\mathbf{p}_{d,in}$ it can be proven that:

$$y_{out} = \pm\frac{r|x_B - x_A|}{|AB|} \tag{3.21}$$

and

$$x_{out} = (1 - t_{AB})x_A + t_{AB}x_B, \tag{3.22a}$$

$$t_{AB} = \frac{y_{out} - y_A}{y_B - y_A}, \tag{3.22b}$$

$$t_{AB} \in [0, 1], \tag{3.22c}$$

In order to find x_{out}, we first replace y_{out} from (3.21) in (3.22b). The obtained t_{AB} values are validated against (3.22c) and those accepted are used in (3.22a) to compute x_{out}. Thus $\mathbf{p}_{out} = (x_{out}, y_{out})^T$ is computed.
Finally, similar to Section 3.2.1 it can be proven that if $y_{out} \in [-r, r]$, then:

$$x_{d,out} = -\frac{x_{out}}{2} \pm \sqrt{r^2 - y_{out}^2} \tag{3.23}$$

The largest root of equation (3.23) that satisfies equations (3.5) and (3.9) is accepted for the last occurring intersection of the disc with edge AB.
The computation of any other first or last occuring disc intersection with edges AC, BC or vertices B, C can be reduced to one of the cases above (see Sections 3.2.1, 3.2.1).

3.3 Traversal-Cost Computation Method

This Section proposes a method for computing the cost of a linear traversal of a disc moving between two arbitrary points in a weighted triangulated planar subdivision. The details of the method are presented in a top-to-bottom level approach.

3.3.1 Traversal Cost Between Points

Algorithm 2 ComputeTraversalCost: Computes the $\vec{sg}$ traversal costs.

Input: point s, point g, radius r
Output: Returns the traversal cost from node s to node g

```
1: box middleBox := ComputeIntermediateBox( s, g , r )
2: float totalCost := 0
3: overlapIntervalList ovIntList := ComputeOverlapIntervals( s , g , r, middleBox )
4: for all overlapIntervals ovInt in ovIntList do
5:     totalCost += Weight( ovInt.T ) * ( ovInt.end - ovInt.start )
6: end for
7: return totalCost
```

Algorithm 2 outlines the overall method for computing the traversal cost when a disc of radius r moves from an arbitrary point s of the weighted scene towards an arbitrary point g. The algorithm computes the traversal cost as the sum of lengths of the overlap intervals between the sweeping disc and the scene triangles weighted by the corresponding triangle weights. An overlap interval between a moving disc and a triangle is defined by the position of the disc center during the first

and last occurring intersections between the disc and that specific triangle. A member variable $\mathcal{T}$ also stores the region type that is associated with an overlap interval. The intervals are computed using the function ComputeOverlapIntervals which is discussed in Section 3.3.2. The overlap intervals do not intersect each other, which is in compliance with our basic assumption that only the maximum weighted triangle is considered for each disc position (see Section 3.1). In addition, the three detection phases through which ComputeOverlapIntervals computes the overlap intervals guarantee that all relevant triangles are detected. As a result, the overlap intervals have a summed total length of $|\vec{sg}|$. The function ComputeIntermediateBox computes the box that corresponds to the intermediate phase of the sweeping disc approach (see Figure 3.3b). Therefore, *middleBox* is a rectangle with its top-left corner at $(\ 0,\ r)$ and its bottom right corner at $(\ x_g,\ -r)$. The coordinates of the rectangle corners are expressed in the local coordinate system as defined in Section 3.2 (see Figure 3.4).

3.3.2 Compute Sweeping Disc Overlap Intervals

Algorithm 3 ComputeOverlapIntervals: Computes all disc-triangle overlap intervals that contribute to the $\vec{sg}$ traversal costs.

Input: point s, point g, radius r, box *middleBox*
Output: Returns all overlap intervals from node s to node g that contribute to the $\vec{sg}$ traversal costs as an overlap interval list

```
 1: triangleList tlTotal = FindAllIntersectingTriangles( s , g , r, middleBox )
 2: overlapIntervalList ovIntlList := ∅
 3: ovIntlList.Add( [0, x_g] , 0 )
 4: for all triangles t in tlTotal do
 5:    overlapInterval currentInterval = FindTriangleOverlapInterval( s , g , r , middleBox, t )
 6:    for all overlapIntervals ovInt in ovIntList do
 7:       if (currentInterval ∩ ovInt ≠ ∅) &&
          ( Weight( currentInterval.T ) > Weight( ovInt.T ) ) then
 8:          ovIntList.Remove( ovInt )
 9:          for all overlapIntervals aInt in ovInt ∩ currentInterval do
10:             ovIntList.Add( aInt, currentInterval.T )
11:          end for
12:          for all overlapIntervals bInt in ovInt∩Complement( currentInterval ) do
13:             ovIntList.Add( bInt , ovInt.T )
14:          end for
15:       end if
16:    end for
17: end for
18: return ovIntList
```

Algorithm 3 provides the details on how the list of the contributing overlap intervals is computed. First, we detect all triangles that intersect the sweeping disc using function FindAllIntersectingTriangles (see Algorithm 4). Next, Algorithm 3 returns a list of overlapping intervals by keeping only the maximum weighted triangle for each disc position in the list. Initially, all overlap intervals are computed using function FindTriangleOverlapInterval (see Algorithm 5). Then, the intervals are added sequentially while guaranteeing that the maximum weighted triangle so far is attributed to each position of the disc along $\vec{sg}$ (see lines 6-16). The function Complement returns the complement $(-\infty, x_{start}) \cup (x_{end}, +\infty)$ of an interval $[x_{start}, x_{end}]$.

Algorithm 4 FINDALLINTERSECTINGTRIANGLES: Finds all triangles that intersect the sweeping disc and returns them as a triangle list

Input: point s, point g, radius r, $middleBox$
Output: Returns all triangles that intersect with sweeping disc as a triangle list.

```
1: triangleList tlStart := ∅
2: triangleList tlEnd := ∅
3: triangleList tlTotal := ∅
4: triangle startTriangle = FINDTRIANGLE( s )
5: triangle goalTriangle = FINDTRIANGLE( g )
6: TRIANGLEDISCWALK( tlStart , startTriangle , s , r )
7: TRIANGLEBOXWALK( tlTotal , startTriangle , middleBox )
8: TRIANGLEDISCWALK( tlEnd , goalTriangle , g , r )
9: tlTotal.Add( tlStart )
10: tlTotal.Add( tlEnd )
11: return tlTotal
```

Algorithm 4 detects all triangles that intersect the sweeping disc. First, the function FINDTRIANGLE is employed to locate the triangles that the disc center lies in during the start and the end of its traversal. Next, the detection of all overlap triangles is performed in three phases as shown in Figure 3.3. In detail, when the disc is centered at starting point s (Figure 3.3a) or goal point g (Figure 3.3c), all overlapping triangles are found using function TRIANGLEDISCWALK (see Algorithm 6). For the intermediate phase, function TRIANGLEBOXWALK (see Algorithm 7) is responsible for detecting all triangles that intersect the box (Figure 3.3b).

Algorithm 5 FINDTRIANGLEOVERLAPINTERVAL

Input: point s , point g, radius r, box $middleBox$, triangle t
Output: Returns the overlap interval of sweeping disc with triangle t

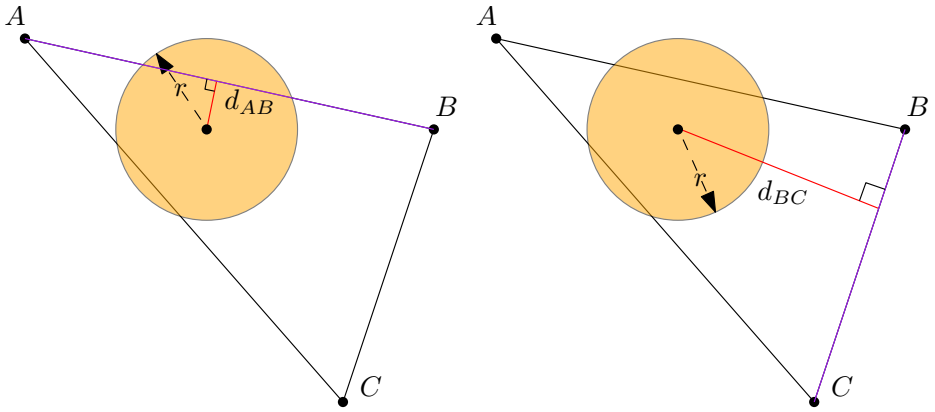
```
1: for all vertices v in t.V do
2:    p_{d,in}^{(v)} := COMPUTEFIRSTDISCVERTEXINTERSECTION( v , s , g , r )
3:    p_{d,out}^{(v)} := COMPUTELASTDISCVERTEXINTERSECTION( v , s , g , r )
4:    overlapInterval^{(v)} := [ p_{d,in}^{(v)}.x, p_{d,out}^{(v)}.x ] ∩ [ 0, x_g ]
5: end for
6: for all edges e in t.E do
7:    p_{d,in}^{(e)} := COMPUTEFIRSTDISCEDGEINTERSECTION( e ,s ,g , r )
8:    p_{d,out}^{(e)} := COMPUTELASTDISCEDGEINTERSECTION( e ,s ,g , r )
9:    overlapInterval^{(e)} := [ p_{d,in}^{(e)}.x, p_{d,out}^{(e)}.x ] ∩ [ 0, x_g ]
10: end for
11: overlapInterval^{(t)} := ⋃_{v∈t.V} overlapInterval^{(v)} ∪ ⋃_{e∈t.E} overlapInterval^{(e)}
12: overlapInterval^{(t)}.T = t.AttributeInformation
13: return overlapInterval^{(t)}
```

Algorithm 5 computes the overlap interval between the sweeping disc and a triangle that has been found to overlap. First, all disc-vertex overlap intervals are computed using functions COMPUTEFIRSTDISCVERTEXINTERSECTION and COMPUTELASTDISCVERTEXINTERSECTION, which implement the solutions devised in Section 3.2.1. Next, all disc-edge overlap intervals are computed using functions COMPUTEFIRSTDISCEDGEINTERSECTION and COMPUTELASTDISCEDGEINTERSECTION, which implement the solutions devised in Section 3.2.2. Pruning is also performed to discard any overlaps that occur outside the scope of $\vec{sg}$ traversal (see Equations (3.8) and (3.9) in Section 3.2). The union of all computed intervals consists the total overlap interval of the triangle with the sweeping disc. The triangle t consists a face in the DCEL representation. The attribute information of t is assigned as the region type $\mathcal{T}$ of the overlap interval. The algorithm is applicable to a triangle that has been found to overlap the disc during any of the three sweeping phases (Figure 3.3).

3.3.3 Detect all Disc-Triangle Overlaps (Triangle-Disc Walk)

We devise a walking scheme for detecting all triangles that a disc overlaps when centered at an arbitrary position. The triangle that contains the disc center is guaranteed to overlap the disc. By checking whether there exist edges of that triangle which overlap with the disc, it can be verified

whether the disc overlaps with the neighboring triangles that share those edges. We extend the search to all neighboring triangles that are found to overlap the disc and perform the same check to the edges of those triangles. It is guaranteed that, when this recursive walking scheme terminates, all overlapping triangles are detected. This is due to the fact that the disc can only overlap a list of triangles that are connected by shared edges. As a result, by initiating the walk from a triangle that overlaps the disc, the method guarantees that all overlapping triangles will be found.
The test for determining whether there is a disc-edge overlap consists of checking whether the distance of the disc center from the edge is smaller or equal to its radius (see Figure 3.6). If the distance is smaller or equal to the radius then the disc intersects the edge, otherwise no disc-edge intersection exists.



(a) Disc-edge intersection **(b)** Disc-edge non-intersection

Figure 3.6: A disc (*orange*) overlaps ABC triangle. The disc-edge overlap test compares the disc radius r (*dashed arrow*) with the distance d (*red*) of the disc center from the edge (*purple*). In (a) $d_{AB} < r$, therefore the disc intersects AB. In (b) $d_{BC} > r$, therefore the disc does not intersect BC.

Algorithm 6 TriangleDiscWalk: Adds *currentTriangle* to the *tlTotal* triangle list and examines neighboring triangles for overlap with disc.

Input: triangleList *tlTotal*, triangle *currentTriangle*, point *discCenter*, radius r
Output: *tlTotal* is recursively filled with all triangles that overlap with disc

```
1: tlTotal.Add( currentTriangle )
2: for all edges e in currentTriangle.E do
3:    if OverlapEdgeDisc( e , discCenter , r ) is true then
4:       triangle nextTriangle := DcelFindTriangle( Twin( e ) )
5:       TriangleDiscWalk( tlTotal , nextTriangle , discCenter , r )
6:    end if
7: end for
```

Algorithm 6 implements the walking scheme that is used to find all scene triangles that intersect a disc of radius r, centered at position *discCenter*. The algorithm starts from a triangle *currentTriangle*, that is found to intersect the disc, and adds it to the intersecting triangle list *tlTotal*. All three edges of *currentTriangle* are checked for overlap with the disc using OverlapEdgeDisc which implements the intersection test described above. Then for each intersecting edge e, the DCEL structure provides the *twin* of e as well as the corresponding neighbor triangle. This triangle intersects the disc, because edge e and its twin intersect it. As a result, the neighbor triangle that has been found to overlap the disc must be added in the intersection list. All the edges of this triangle should be further checked for disc intersection, repeating the procedure described above. Algorithm 6 is employed for the starting phase (Figure 3.3a) and ending phase (Figure 3.3c) of the sweeping disc approach by Algorithm 4.

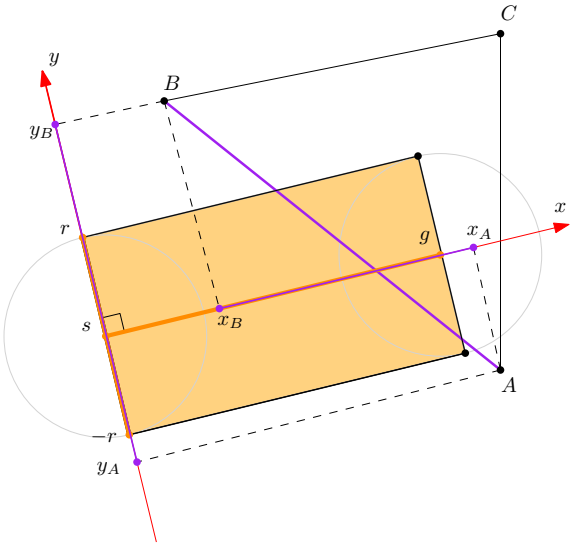
3.3.4 Detect all Box-Triangle Overlaps (Triangle-Box Walk)

Algorithm 7 TriangleBoxWalk: Adds *currentTriangle* to the *tlTotal* triangle list and examines neighbouring triangles for overlap with box.

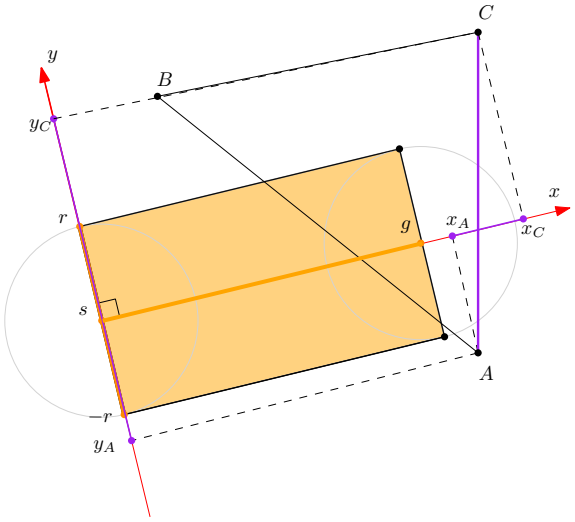
Input: triangleList *tlTotal*, triangle *currentTriangle*, box *middleBox*
Output: *tlTotal* is recursively filled with all triangles that overlap with box

```
1: tlTotal.Add( currentTriangle )
2: for all edges e in currentTriangle.E do
3:    if OverlapEdgeBox( e , middleBox ) is true then
4:       triangle nextTriangle := DcelFindTriangle( Twin( e ) )
5:       TriangleBoxWalk( tlTotal , nextTriangle , middleBox )
6:    end if
7: end for
```

Similar to function TriangleDiscWalk (see Section 3.3.3), a walking scheme is devised for detecting all triangles that intersect the box of the intermediate phase (Figure 3.3b) of the sweeping disc approach. TriangleBoxWalk, as detailed in Algorithm 7, differs from TriangleDiscWalk only in that it employs a function OverlapEdgeBox to determine whether an edge of a triangle overlaps the intermediate box. This test is performed by checking whether both the x- and y-projections of the edge on the current coordinate system (see Figure 3.4) overlap with the box intervals $[0, x_s]$ and $[-r, r]$, respectively. For an example of a box-edge overlap and a box-edge non-overlap see Figure 3.7. Algorithm 4 initiates the triangle-box walking scheme from the triangle that contains the disc center during the starting phase. Due to the manner in which the three sweeping phases overlap, the initial triangle will always intersect the intermediate box (see Figures 3.3a and 3.3b for an example). As a result, this algorithm guarantees that all triangles that overlap the intermediate box will be eventually detected when the recursive walk terminates.



(a) Box-edge intersection



(b) Box-edge non-intersection

Figure 3.7: An intermediate-phase box (*orange*) overlaps ABC triangle. The box-edge overlap test requires both x- and y-projections of the edge (*purple*) to overlap with the box. (a) AB overlaps the box in $[0, x_g]$ on the x-axis and in $[-r, r]$ on the y-axis. Therefore, there is a box-edge overlap . (b) AC overlaps the box in $[-r, r]$ on the y-axis but does not overlap the box on the x-axis. Therefore, there is no box-edge overlap.

Chapter 4

Modified MIRAN

In this chapter, we propose a modification to the MIRAN method so that it can be applied to smoothly steer a disc-shaped character towards its goal in a weighted triangulated planar scene. This chapter does not discuss the computation of an indicative route that accounts for disc clearance. Instead, our analysis assumes that the indicative route is already provided as an input to the modified MIRAN algorithm. In Section 4.1, we discuss the computation of the candidate attraction points for the modified MIRAN method. In Section 4.2, we modify the weight function that is used to select the attraction point among the candidates. The modified weight function employs our traversal-cost computation method (see Section 3.3) to account for the disc-shape of the character.

4.1 Computing Candidate Attraction Points

Let R be the radius of a disc character for which an indicative route has been provided from start point s to goal point g. At each simulation step i, the definition of the reference point r_i is the same as in the original MIRAN method. The sampling scheme along the indicative route π_{ind} is also the same and utilizes the shortcut parameter σ and the sampling distance d (see Section 2.1). In the original MIRAN method, during the computation of the candidate attraction points, we consider each curve segment that is visible from the character's actual position x_i. We place candidates at the endpoints $\pi_{ind}(a_j)$ and $\pi_{ind}(b_j)$ of each visible curve segment. However, in the modified version of MIRAN, each visible curve segment that lies within σ curve length distance from the reference point r_i should have its endpoints $\pi_{ind}(a_j)$ and $\pi_{ind}(b_j)$ adjusted to account for the clearance necessary for the disc-shaped character. Let p_{a_j} (p_{b_j}) be the point where the line segment $l(x_i, \pi_{ind}(a_j))$ ($l(x_i, \pi_{ind}(b_j))$) touches a static obstacle. A disc character centered at p_{a_j} (p_{b_j}) would intersect the static obstacle. In order to find the new position of the endpoint $\pi_{ind}(a'_j)$ ($\pi_{ind}(b'_j)$), the following adjustments are performed. First, we rotate the disc centered at p_{a_j} (p_{b_j}) by an angle θ_{a_j} (θ_{b_j}) with respect to an axis that is perpendicular to the plane and passes from x_i. Let p'_{a_j} (p'_{b_j}) be the disc center after the rotation . Angle θ_{a_j} (θ_{b_j}) is defined as the angle of rotation that is necessary for a disc of radius R centered at p'_{a_j} (p'_{b_j}) to touch the static obstacle at p_{a_j} (p_{b_j}).
After θ_{a_j} (θ_{b_j}) has been computed, a ray is cast from x_i towards p'_{a_j} (p'_{b_j}). Let $q_{a_j,u}$ ($q_{b_j,u}$) be an intersection point of the ray with the indicative route with $1 \leq u \leq k$, where k is the total number of intersections of the ray with the indicative route and u is the index of those intersection points in order of occurrence starting from s and moving towards g along the indicative route. For an arbitrary point p along π_{ind}, let t_p define the curve parameter of p, where $t_p \in [0, 1]$. The indicative route is defined as a curve $\pi_{ind} : [0, 1] \rightarrow \mathbb{R}^2$ with start point s and goal point g as its endpoints, i.e. $t_s = 0$ and $t_g = 1$. Let n be the number of visible curve segments computed by the original MIRAN method. For a modified endpoint $\pi_{ind}(a'_j)$ ($\pi_{ind}(b'_j)$) to exist, the ray must first intersect the indicative route before intersecting any scene obstacle.
The modified endpoint $\pi_{ind}(a'_j)$ is computed as follows:

- If $j = 1$, then the modified endpoint $\pi_{ind}(a'_1)$ is the reference point $r_i = \pi_{ind}(a_1)$.
- Else, for every $1 < j \leq n$ the modified endpoint $\pi_{ind}(a'_j)$ is the intersection point $q_{a_j,m}$ that satisfies the following properties:

1. $q_{a_j,m}$ has minimum curve length distance from $\pi_{ind}(a_j)$ along the indicative route, i.e. $m = \underset{1 \le u \le k}{\text{argmin}} \left(||t_{q_{a_j,u}} - t_{\pi_{ind}(a_j)}|| \right)$
2. $q_{a_j,m}$ lies between r_i and σ_i along the indicative route, i.e. $t_{r_i} \le t_{q_{a_j,m}} \le t_{\sigma_i}$.
3. $q_{a_j,m}$ lies between $\pi_{ind}(a_j)$ and $\pi_{ind}(b_j)$ along the indicative route, i.e. $t_{a_j} < t_{q_{a_j,m}} < t_{b_j}$.

The modified endpoint $\pi_{ind}(b'_j)$ is computed as follows:

- If $j = n$ and $\pi_{ind}(b_n) = \sigma_i$ then we also let $\pi_{ind}(b'_n) = \sigma_i$. It should be noted that a linear traversal from x_i to σ_i might not be free of collisions. Therefore σ_i must not be used as an attraction point, if obstacles occlude $x_i\vec{\sigma}_i$ traversal of the disc. Nevertheless, the modified weight function ω (see Section 4.2) will account for this by reporting a very large ω value for $l(x_i, \sigma_i)$, thus discouraging picking σ_i as the best candidate attraction point.
- Else for every $1 \le j \le n$ the modified endpoint $\pi_{ind}(b'_j)$ is the intersection point $q_{b_j,m}$ that satisfies the following properties:

  1. $q_{b_j,m}$ has minimizum curve length distance from $\pi_{ind}(b_j)$ along the indicative route, i.e. $m = \underset{1 \le u \le k}{\text{argmin}} \left(||t_{q_{b_j,u}} - t_{\pi_{ind}(b_j)}|| \right)$
  2. $q_{b_j,m}$ lies between r_i and σ_i along the indicative route, i.e. $t_{r_i} \le t_{q_{b_j,m}} \le t_{\sigma_i}$.
  3. $q_{b_j,m}$ lies between $\pi_{ind}(a_j)$ and $\pi_{ind}(b_j)$ along the indicative route, i.e. $t_{a_j} < t_{q_{b_j,m}} < t_{b_j}$.

However, if the ray does not intersect the route, then it is not possible to define a new curve segment endpoint $\pi_{ind}(a'_j)$ ($\pi_{ind}(b'_j)$). If the ray intersects a static obstacle before intersecting the route, then it is not possible to define a new curve segment endpoint $\pi_{ind}(a'_j)$ either $\pi_{ind}(b'_j)$ because not enough clearance exists between the obstacles. In these cases, both $\pi_{ind}(a'_j)$ and $\pi_{ind}(b'_j)$ are discarded and the method proceeds in modifying $\pi_{ind}(a_{j+1})$ and $\pi_{ind}(b_{j+1})$, if they exist. Figure 4.1 illustrates the computation of $\pi_{ind}(b'_1)$ for the disc variation of the problem that was presented in Figure 2.1 (see Section 2.1).

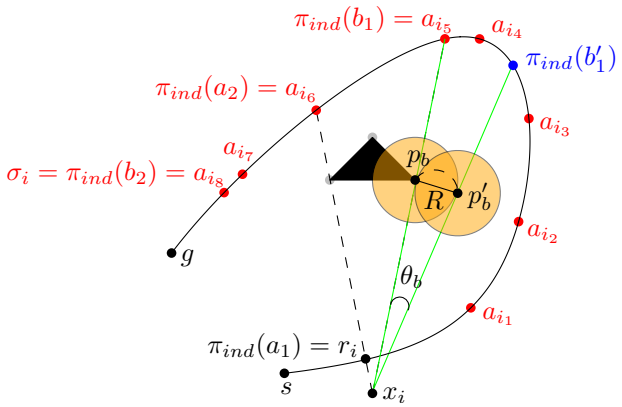


Figure 4.1: Computation of $\pi_{ind}(b'_1)$ (*blue*) for a disc (*orange*) of radius R on the same scene with Figure 2.1

Angle θ_{a_j} (θ_{b_j}) is computed by applying the *law of cosines* to the triangle $\tau_{x_i p_{a_j} p'_{a_j}}$ ($\tau_{x_i p_{b_j} p'_{b_j}}$). In detail, θ_{b_j} is computed as follows (see Figure 4.1).
Law of cosines on triangle $\tau_{x_i p_{b_j} p'_{b_j}}$:

$$\theta_{b_j} = \arccos\left(\frac{|x_i p_{b_j}|^2 + |x_i p'_{b_j}|^2 - |p_{b_j} p'_{b_j}|^2}{2|x_i p_{b_j}||x_i p'_{b_j}|} \right) \tag{4.1}$$

Since $|x_i p_{b_j}| = |x_i p'_{b_j}|$, Equation (4.1) is rewritten as:

$$\theta_{b_j} = \arccos\left(\frac{2|x_i p_{b_j}|^2 - |p_{b_j} p'_{b_j}|^2}{2|x_i p_{b_j}|^2} \right) \Leftrightarrow \tag{4.2}$$

Finally, since $|p_{b_j} p'_{b_j}| = R$, Equation (4.2) is rewritten as:

$$\theta_{b_j} = \arccos\left(1 - \frac{1}{2}\left(\frac{R}{|x_i p_{b_j}|}\right)^2\right) \tag{4.3}$$

Similarly, it can be proven that angle θ_{a_j} is given by:

$$\theta_{a_j} = \arccos\left(1 - \frac{1}{2}\left(\frac{R}{|x_i p_{a_j}|}\right)^2\right) \tag{4.4}$$

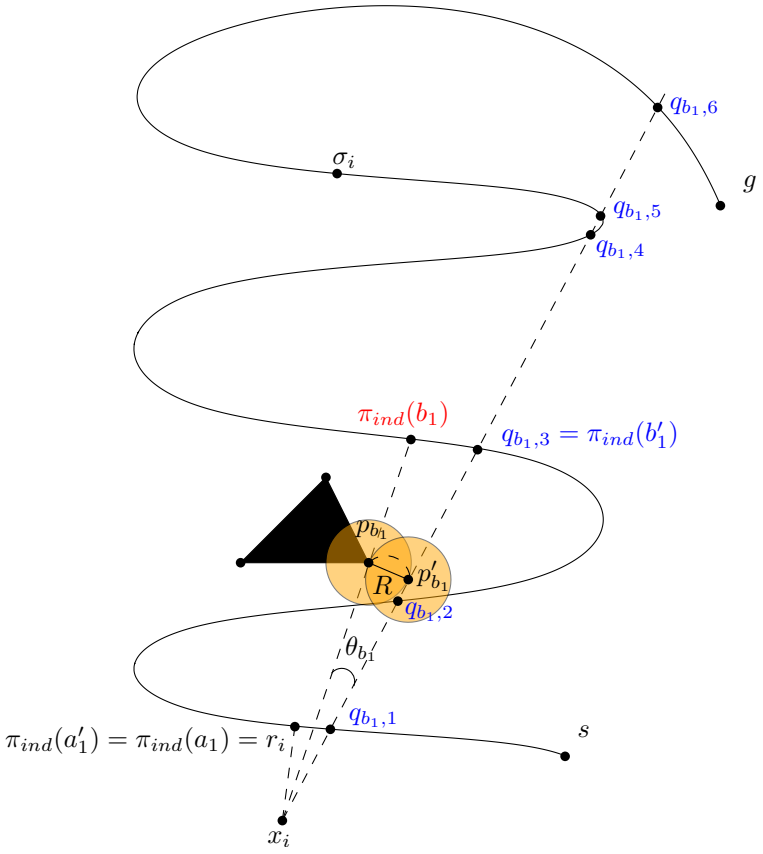


Figure 4.2: Computation of $\pi_{ind}(b'_1)$

Before registering a pair of endpoints $\pi_{ind}(a'_i)$ and $\pi_{ind}(b'_i)$ that were computed using the procedure described above, a final validation check occurs: $\pi_{ind}(b'_i)$ must succed $\pi_{ind}(a'_i)$ along the route, i.e. $t_{\pi_{ind}(a'_i)} < t_{\pi_{ind}(b'_i)}$. Figure 4.3 gives an example where the computed pair $\pi_{ind}(a'_2)$ and $\pi_{ind}(b'_2)$ is a valid pair. In this example, $\pi_{ind}(a'_2)$ preceeds $\pi_{ind}(b'_2)$ along the route. Therefore, $\pi_{ind}(a'_2)$ and $\pi_{ind}(b'_2)$ can be safely registered as the modified endpoints of the second visible curve segment of the route. Figure 4.4 gives an example where the computed pair $\pi_{ind}(a'_2)$ and $\pi_{ind}(b'_2)$ is not a valid pair. In this example, $\pi_{ind}(a'_2)$ is after $\pi_{ind}(b'_2)$ along the route. As a result, no visible curve segment exists for the disc case within the curve segment with endpoints $\pi_{ind}(a_2)$ and$\pi_{ind}(b_2)$ of the corresponding point case.

The computation of the candidate attraction points is performed in two stages. During the first stage, we modify all candidate attraction points $\pi_{ind}(a_j)$ and $\pi_{ind}(b_j)$ that the visibility test of the original method produces using the method described above. During the second stage, we apply the sampling scheme of the original method (see Section 2.1) on all curve segments with endpoints the modified candidate attraction points $\pi_{ind}(a'_j)$ and $\pi_{ind}(b'_j)$ that the first stage produces. Figure 4.5 shows the candidate attraction points that the modified MIRAN method produces on the same scene and simulation instance with Figure 2.1.

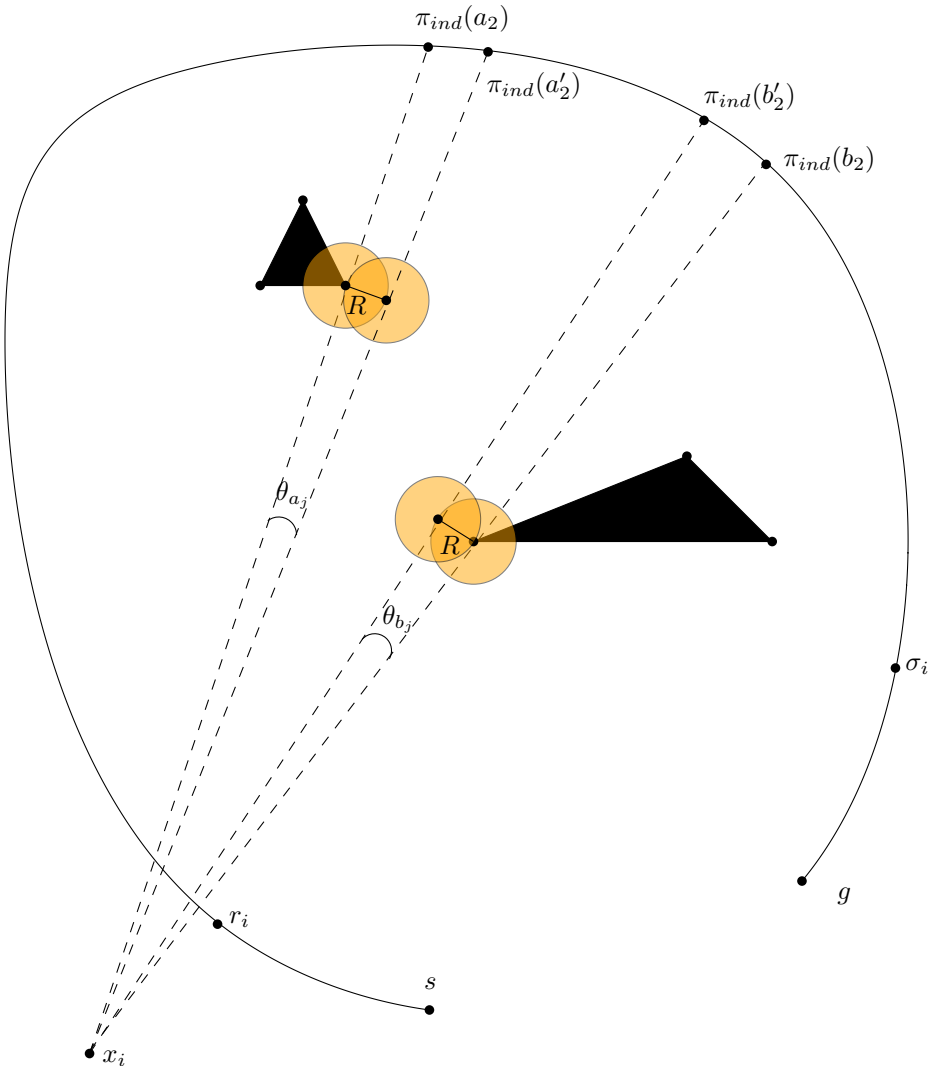


Figure 4.3: Valid computation of $\pi_{ind}(a'_1)$ and $\pi_{ind}(b'_1)$: Point $\pi_{ind}(a'_i)$ lies before point $\pi_{ind}(b'_i)$ along the indicative route π_{ind}.

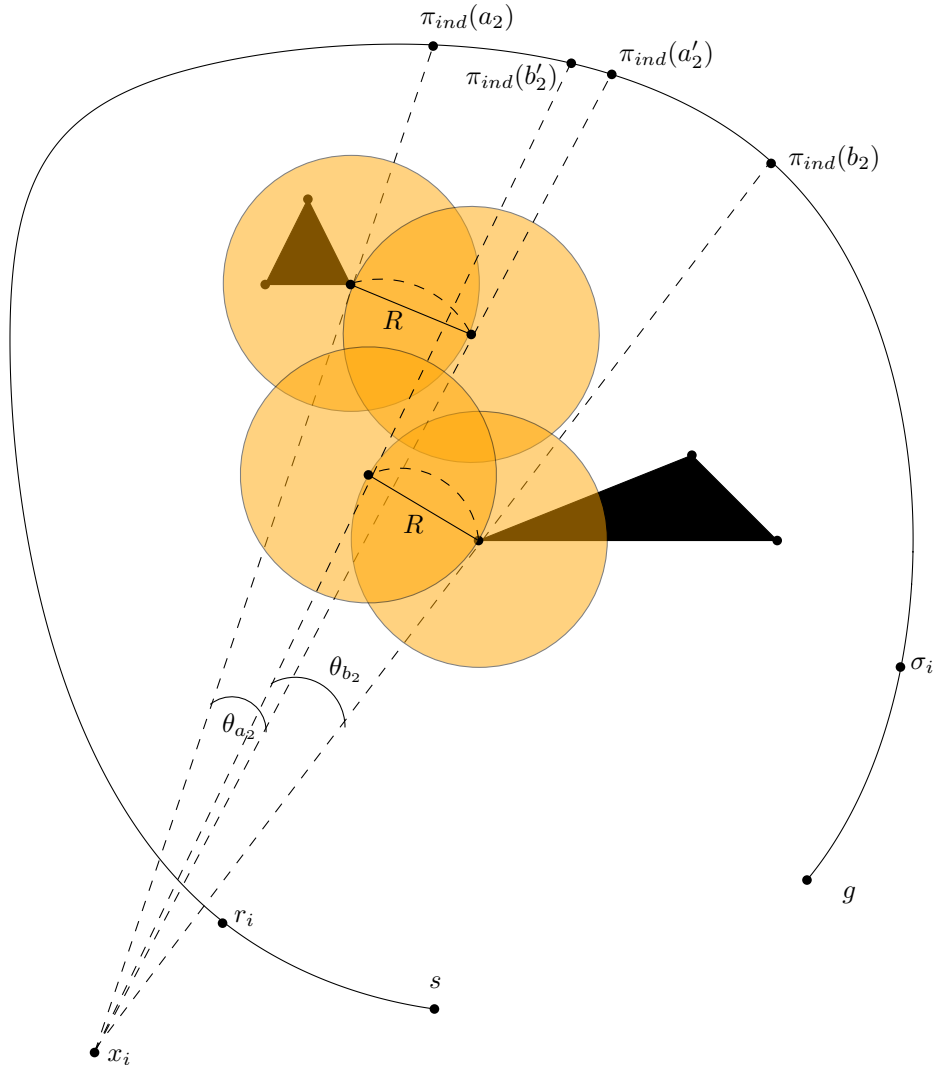


Figure 4.4: Invalid computation of $\pi_{ind}(a'_2)$ and $\pi_{ind}(b'_2)$: Point $\pi_{ind}(a'_i)$ lies after point $\pi_{ind}(b'_i)$ along the indicative route π_{ind}.

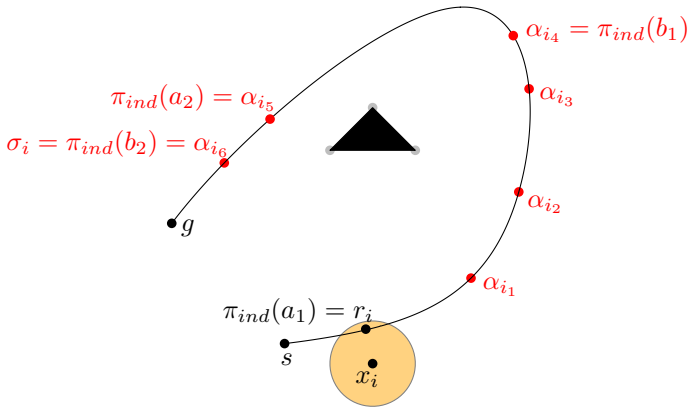


Figure 4.5: Candidate attraction points of the disc variant of Figure 2.1.

,

4.2 Picking Best Candidate Attraction Point

Figure 4.6 shows the candidate attraction points that the modified MIRAN method produces on the same scene and simulation instance with Figure 2.1 while also displaying the triangulatation.

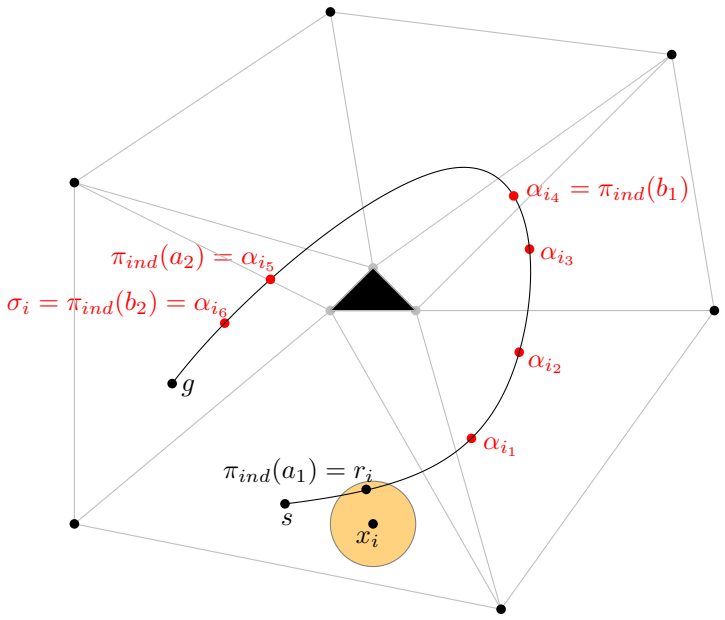


Figure 4.6: Candidate attraction points of the disc variant of Figure 2.1.

After computing all candidate attraction points α_{i_j} with $1 \leq j \leq k$, the original MIRAN method chooses the candidate that minimizes weight function ω as attraction point α_i for the i-th simulation step. The original method defines weight function ω in Equation (2.1) (see Section 2.1). Instead, the modified method selects as attraction point α_i the candidate α_{i_j} that minimizes the following weight function:

$$\omega(l(a_{i_j}, x_i)) = \frac{1}{d_{i_j}} \textsc{ComputeTraversalCost}(\ x_i,\ \alpha_{i_j},\ R\ ) \tag{4.5}$$

d_{i_j} is the curve length distance between r_i and α_{i_j} measured along the indicative route π_{ind}. In practice, the traversal-cost computation method that was presented in Algorithm 2 (see Section 3.3) is employed with the actual disc center position x_i as start point and each candidate attraction point α_{i_j} as goal point. Figure 4.7 shows the sweeping disc that is utilized by ComputeTraversalCost in order to compute $\omega(l(\alpha_{i_4}, x_i))$.

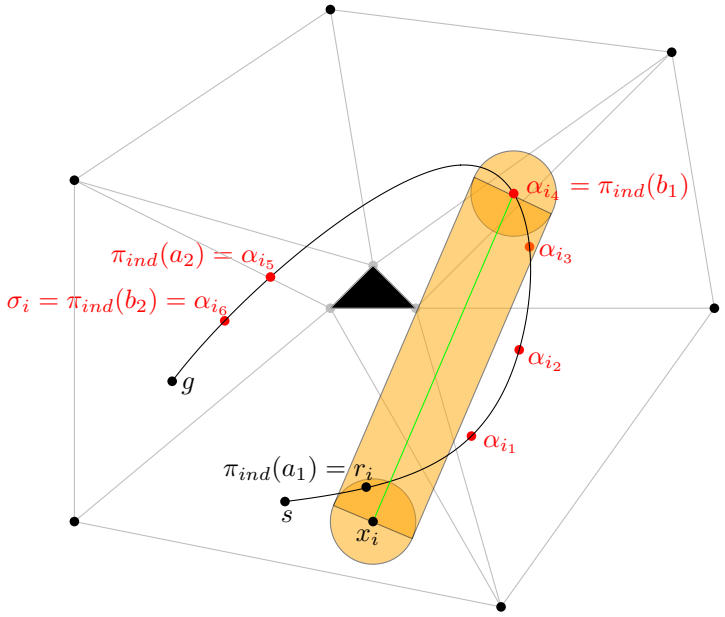


Figure 4.7: Computing the weight value ω for the fourth candidate attraction point α_{i_4}: $\omega(l(\alpha_{i_4}, x_i)) = \frac{1}{d_{i_4}}$ComputeTraversalCost(x_i, α_{i_4}, R).

Chapter 5

Discussion and Future Work

In this work, we have introduced a method for computing the cost of a linear traversal of a disc between two arbitrary points in a weighted triangulated planar environment. We have extended the MIRAN method to make it steer disc-shaped characters in planar weighted regions. This was achieved by modifying the candidate attraction points that the original method produces and by employing our traversal-cost computation method to select the attraction point among the candidates.

The proposed disc traversal-cost computation method (see Section 3) can be employed by any path planner for disc navigation in weighted regions. As a result, this method could be used within a global path planner in order to provide an indicative route with arbitrary clearance. This route can then be used as input to the extended MIRAN method we devised in Section 4.

A significant contribution of our approach is that both the traversal-cost computation method and the proposed extension of the MIRAN method can be applied to the path planning problem for characters with variable radii. If the indicative route provides a collision-free path for a disc radius R_{max}, then the radius R of the character disc may vary throughout the simulation within $(0, R_{max}]$. This is due to the fact that both the modification of the candidate attraction points (see Section 4.1) and the selection of the best candidate based on its ω value (see Section 4.2) do not rely on precomputed structures that are based on the character's radius. Instead, these methods employ all necessary geometry checks to account for current radius value and can have a different radius as input between the simulation steps.

As future work, the proposed methods should be evaluated in terms of quality and performance. Comparisons of the proposed disc variant of MIRAN with the original point navigation method is expected to provide useful insight. Since the proposed traversal-cost computation method relies on an exact computation of the disc-triangle overlaps, it remains to be seen whether this method is fast enough to be utilized in real-time applications such as simulations and games.

Our work relies on the assumption that at any location of the disc, the current traversal cost is determined by the overlapping triangle with the maximum weight. It would be interesting to investigate different weighting schemes and compare them with the proposed method in terms of plausibility of the generated paths.

Bibliography

[ALMS98] Lyudmil Aleksandrov, Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack. An ϵ-approximation algorithm for weighted shortest paths on polyhedral surfaces. In Stefan Arnborg and Lars Ivansson, editors, *Algorithm Theory SWAT'98*, volume 1432 of *Lecture Notes in Computer Science*, pages 11–22. Springer Berlin Heidelberg, 1998.

[AMS05] Lyudmil Aleksandrov, Anil Maheshwari, and Jörg-Rüdiger Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *J. ACM*, 52(1):25–53, January 2005.

[DBVKOS08] Mark De Berg, Marc Van Kreveld, Mark H. Overmars, and Otfried Cheong Schwarzkopf. *Computational Geometry*. Springer, 2008.

[Eri05] Christer Ericson. *Real-Time Collision Detection*. Taylor & Francis US, 2005.

[Ger10] Roland Geraerts. Planning short paths with clearance using explicit corridors. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1997–2004, May 2010.

[GJK88] Elmer G. Gilbert, Daniel W. Johnson, and Sathiya S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *Robotics and Automation, IEEE Journal of*, 4(2):193–203, Apr 1988.

[HKL+99] Kenneth E. Hoff, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 277–286, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[JCIG13] Norman Jaklin, Atlas Cook IV, and Roland Geraerts. Real-time path planning in heterogeneous environments. *Computer Animation and Virtual Worlds*, 24(3-4):285–295, 2013.

[Kal10] Marcelo Kallmann. Shortest paths with arbitrary clearance from navigation meshes. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 159–168, Aire-la-Ville, Switzerland, 2010. Eurographics Association.

[KGO09] Ioannis Karamouzas, Roland Geraerts, and Mark H. Overmars. Indicative routes for path planning and crowd simulation. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 113–120. ACM, 2009.

[MP91] Joseph S. B. Mitchell and Christos H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *J. ACM*, 38(1):18–73, January 1991.

[OG08] Mark H. Overmars and Roland Geraerts. Enhancing corridor maps for real-time path planning in virtual environments. In *COMPUTER ANIMATION AND SOCIAL AGENTS (CASA'08)*, pages 64–71, 2008.

[SR01] Zheng Sun and John Reif. Bushwhack: An approximation algorithm for minimal paths through pseudo-euclidean spaces. In Peter Eades and Tadao Takaoka, editors, *Algorithms and Computation*, volume 2223 of *Lecture Notes in Computer Science*, pages 160–171. Springer Berlin Heidelberg, 2001.

[WvdBH05] Ron Wein, Jur P. van den Berg, and Dan Halperin. The visibility–voronoi complex and its applications. In *Proceedings of the 21st Annual Symposium on Computational Geometry*, SCG '05, pages 63–72, New York, NY, USA, 2005. ACM.