2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Planning Short Paths with Clearance using Explicit Corridors

Roland Geraerts

***Abstract*— A central problem of applications dealing with virtual environments is planning a collision-free path for a character. Since environments and their characters are growing more realistic, a character's path needs to be visually convincing, meaning that the path is smooth, short, has some clearance to the obstacles in the environment, and avoids other characters. Up to now, it has proved difficult to meet these criteria simultaneously and in real-time.**

We introduce a new data structure, i.e. the *Explicit Corridor Map*, which allows creating the shortest path, the path that has the largest amount of clearance, or any path in between. Besides being efficient, the corresponding algorithms are surprisingly simple. By integrating the data structure and algorithms into the Indicative Route Method, we show that visually convincing short paths can be obtained in real-time.

I. Introduction

One of the main challenges in applications dealing with virtual environments is planning a path for a character. Traditionally, algorithms were devised which computed the character's positions from a start to a goal location without colliding with obstacles and other characters. While these algorithms were successfully applied in fields such as mobile robots, manipulation planning and human robot planning [1]–[3], current virtual environment applications, such as games and crowd simulations, pose many new challenges to the algorithms. That is, *visually convincing* short paths for many characters traversing in the ever growing environments need to be planned simultaneously and in real-time. Hence, only a fraction of a millisecond per second CPU time may be spent per character for computing the visually convincing short path, i.e. a path that is smooth, short, keeps some clearance to obstacles and avoids other characters. The path should be short because it usually takes the least time to traverse. However, the path should not be the shortest one because such a path would touch the obstacles, possibly leading to collisions with the environment when the character is animated. Hence, a shortest path having some amount of minimum clearance to the obstacles is required. Current planners do not satisfy these criteria simultaneously, are slow or cost much memory when operating on large environments.

Introduced in 1968, the A*-algorithm is one of the first planners [4]. This planner is still popular because of its simplicity and ability to find the shortest path in a grid of free cells covering the environment. Nevertheless, resulting paths tend to have little clearance to the obstacles and can be aesthetically unpleasant, so care must be taken to smooth them.

This work was partially supported by the ITEA2 Metaverse1 (www.metaverse1.org) Project and the Dutch BSIK/BRICKS Project. R. Geraerts is with Institute of Information and Computing Sciences, Utrecht University, 3508 TA Utrecht, the Netherlands. E-mail: roland@cs.uu.nl.

Fig. 1. A smooth, short path with clearance inside an Explicit Corridor.

Like the A*-algorithm, the Potential-Field method [5] is able to include obstacle avoidance. In addition, smooth paths can be produced by following the direction of the steepest descent of the potential toward the goal. While it is possible to compute a potential without local minima [6], the computation is rather expensive, and, hence, may compromise the real-time performance.

Roadmap-based methods, such as Visibility graphs [2], Rapidly-exploring Random Trees [7] and Probabilistic Roadmap Methods [8], [9], do not have local minima and usually ensure that a path can be found if one exists. These methods build a roadmap graph which represents the free space in the environment. Because this graph can be constructed off-line, real-time performance can be achieved when a path is extracted from this graph. In addition, their strength is that they can be applied to problems with many degrees of freedom. Nevertheless, they lack flexibility because they output a fixed path extracted from a one-dimensional graph. In addition, the paths are jerky. While optimization algorithms exist, they are still too slow for real-time performance [10].

Another category of algorithms use the Voronoi diagram for obtaining paths that have the maximum amount of clearance to obstacles [11], [12]. In contrast, references [13] and [14] find the shortest path for disks moving in the plane.

Recently, the concept of path planning inside corridors has been introduced [15]–[17]. Such a corridor is defined as a sequence of empty disks. Because the union of these disks is two-dimensional, corridors facilitate creating collision-free smooth paths for characters while avoiding other characters. This flexibility is hard to obtain for methods operating on one-dimensional graphs. Besides this desired flexibility, corridors allow creating many paths in real-time because they are extracted from a data structure that can be computed off-

line. Nevertheless, the references mentioned above currently do not support creating shortest paths with some amount of minimal clearance. Creating such paths is supported though in [18]. Paths are extracted from the Visibility-Voronoi Complex, which is a data structure that allows interpolating between the Visibility graph (which yields shortest paths) and Voronoi diagram (which yields maximum clearance paths). The strength of their work is that their algorithm is exact and yields visually convincing short paths. In addition, global shortest paths can be computed, but at the cost of quadratic storage requirements. Also, their algorithm is intricate (because it uses complex algebraic elements) and slow in practice (because it uses exact number types).

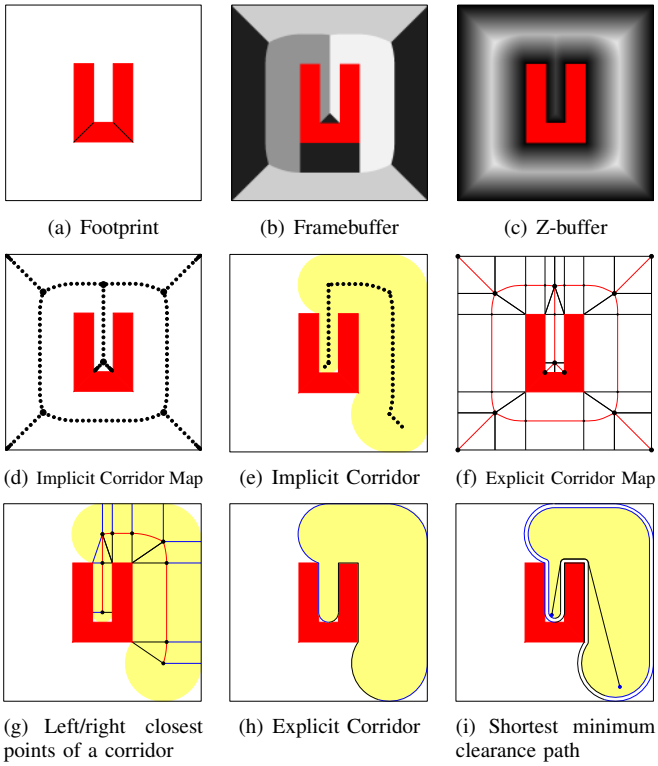
We propose a data structure that has linear complexity (in the number of vertices of the obstacles), at the cost of the possibility of not finding the global shortest path. This structure, i.e. the *Explicit Corridor Map*, allows our new and simple algorithm to compute the shortest path with a variable amount of minimum clearance inside a corridor. By integrating the algorithm into the Indicative Route Method [19], flexible smooth short minimum-clearance paths are obtained within a few milliseconds. Hence, our algorithm may be applied in interactive environments with many characters.

The paper has been organized as follows: In Section II we introduce the Explicit Corridor Map and show how this data structure can be constructed efficiently by using graphics hardware. From this structure, we extract an Explicit Corridor given a certain start and goal position of the character. Then, we give in Section III and IV algorithms for computing the shortest path or a path with a desired amount of minimum clearance to the obstacles within this corridor. While the resulting paths are now short and safe, they are rigid because the character's motions are limited to the computed path. We show in Section V that a corridor allows for deviating from the computed path so that other characters can be avoided and smoothness is added. Next, we will conduct experiments in Section VI and conclude in Section VII that visually convincing short paths, such as the one depicted in Fig. 1, can be computed in real-time.

II. The Explicit Corridor Map

Our goal is to construct an efficient data structure with small storage costs which allows for fast extraction of the shortest path (within a corridor) with a desired amount of minimum clearance to the obstacles. A small memory footprint of the data structure is desirable, i.e. a structure that is linear in the number of vertices of all obstacles, because modern virtual environments can be huge. Since the structure may be queried by many characters in a short amount of time, a path needs to be computed efficiently. The data structure we propose, i.e. the *Explicit Corridor Map* (ECM), will meet these criteria.

Definition 1 (Explicit Corridor Map)**.** *The Explicit Corridor Map is a generalized Voronoi diagram $G = (V, E)$, where V and E are its Voronoi vertices and Voronoi edges. The edges are annotated with* event points *together with their closest points to obstacles.*



(a) Footprint (b) Framebuffer (c) Z-buffer

(d) Implicit Corridor Map (e) Implicit Corridor (f) Explicit Corridor Map

(g) Left/right closest points of a corridor (h) Explicit Corridor (i) Shortest minimum clearance path

Fig. 2. Construction of the Corridor Map, Implicit and Explicit Corridor.

While the diagram consist of 1-dimensional curves, its annotation explicitly defines a 2D-arrangement of the *free space* in the environment. We will show that this arrangement can be used for planning the desired paths.

Before we show how to annotate an edge (Section II-B) and how the arrangement can be constructed (Section II-C), we will discuss some preliminaries and review how the Generalized Voronoi diagram can be computed efficiently using graphics hardware.

A. The Generalized Voronoi Diagram

Path planning occurs in the *free space*, i.e. the 2D walkable space, of the virtual environment. The free space is defined as the space that is not occupied by the *footprint* of the environment. This footprint consists of a collection of convex *obstacles*, including points, lines and convex polygons, all lying in the ground plane.

The walkable space can be represented concisely by the *Generalized Voronoi Diagram* (GVD), which is a decomposition of the free space into regions such that all points p in a region $\mathcal{R}(p)$ are closer to a particular obstacle than to any other obstacle in the environment [20]. Such a region is called a *Voronoi region*. The boundaries of the Voronoi regions form the underlying graph of the GVD. Each vertex in this graph is located at a non-convex corner (induced by at least two obstacles) or at a location at which three or more edges of the graph meet. An edge connects two vertices and manifests itself as the boundary between two Voronoi regions.

A GVD can be computed efficiently by exploiting graphics hardware [11]. A 3D distance mesh, consisting of polygons, is computed for each obstacle. Each of the meshes is rendered on the graphics card in a different color. A parallel projection of these meshes gives the GVD. The diagram can be retrieved from the graphics card's frame buffer and the clearance values (i.e. distance values) can be found in its Z-buffer, which is visualized in Fig. 2(a-c).

We require that all obstacles are convex, which assures that edges run into concavities, and, hence, the GVD is a complete representation for path planning purposes [21]. This requirement does not limit us because non-convex obstacles (read: polygons) can be converted into convex ones by applying a decomposition scheme, such as triangulation [22] or optimal convex decomposition [23], [24].

Usually, exactly two different convex obstacles form an edge in the graph because the points on the edge are on the boundaries separated by two different colors in the frame buffer. By linking each color to an obstacle, we know for each point its two closest obstacles. These facts will be useful in our analysis. In general, an edge can be formed by a set of obstacles forming a convex chain of obstacle parts. In our analysis we will process the convex chain as a single 'obstacle' casting a single color. Hence, without loss of generality, we will now assume that an edge traces the boundary of two Voronoi regions induced by exactly two convex obstacles.

In [20], we sampled the edges of the GVD and referred to the result as the (implicit) Corridor Map. An example of such a map is given in Fig. 2(d). To each sample point, the radius of the largest empty disk was assigned. Each point/radius combination formed a maximum clearance disk. Then, a sequence of adjacent disks was referred to as an *(implicit) Corridor*, see Fig. 2(e). While an implicit Corridor (Map) was used for planning reasonably short paths, they do not easily support computing shortest minimum-clearance paths because they do not provide an explicit description of the corridor's boundaries.[1] Another disadvantage of having samples is that they do not fully cover the free space. In addition, their memory footprint can be more than linear.

By assigning to some selected samples their left and right closest points on the obstacles, we can obtain an explicit description of the corridor's boundaries, as is shown in Fig. 2(f-h). As we will see, they allow for efficiently computing a shortest minimum-clearance path. We refer the reader to Fig. 2(i) for an example of such a path.

B. Annotating a Voronoi edge

In this section we will describe how an edge of the Explicit Corridor Map can be computed. An edge consists of a Voronoi edge connecting its incident Voronoi vertices, and on this edge, a set of *event points*, together with their closest points on the two obstacles that formed this edge.

[1]While we could compute the boundaries of a set of disks, this would still be an approximation of the real boundaries and would take more than linear time [22], which may compromise real-time computations.

A Voronoi edge is the locus of points that are equidistant to two obstacles. This edge is formed by a concatenation of curves, i.e. *line segments* and *parabolic arcs*. Let B_1 and B_2 denote the end points of such a curve and (l_1, r_1) and (l_2, r_2) denote the corresponding closest points on their left and right closest obstacle, respectively. The curve is a line segment (i.e. bisector) if $l_1 = l_2 \wedge r_1 = r_2$, or if the line segment connecting l_1 with l_2 traces (a part of) the left obstacle and the line segment connecting r_1 with r_2 traces (a part of) the right obstacle. Note that a line segment traces an obstacle if it does not intersect the obstacle. The curve is a parabolic arc[2] (by definition) if $l_1 = l_2 \wedge r_1 \neq r_2$, assuming the line segment from l_1 to l_2 does not intersect its left obstacle. Similarly, we have a parabolic arc if $l_1 \neq l_2 \wedge r_1 = r_2$.

For efficiency reasons, it is important to determine the minimum number of curves that describe the edge. Therefore, we need to find the positions where a curve must end and another one must start. Such a position is referred to as an *event point* on the edge. Let the edge be parameterized by $B[t]$, where $0 \leq t \leq 1$, and let $l[t]$ and $r[t]$ be their corresponding left and right closest points, respectively. Furthermore, let $n_l[t]$ and $n_r[t]$ denote their left and right normals formed by vectors $B[t] - l[t]$ and $B[t] - r[t]$, respectively. Note that $n_l[t]$ and $n_r[t]$ are the normals to their left and right obstacles, and that they can only change at a vertex of an obstacle. Then the event points are given by the discrete set of points S for which the normals first start changing or first stop changing, i.e. for the left side $S = \{B[t] : (n_l[t - \delta t] = n_l[t] \wedge n_l[t + \delta t] \neq n_l[t]) \vee (n_l[t - \delta t] \neq n_l[t] \wedge n_l[t + \delta t] = n_l[t])\}$, where δ is an infinitesimal number. Similarly, the event points are added to S for the right side.

If we are given the edge, we can compute the set S as follows. First, we compute for each obstacle its normals. We associate normal n_i with edge e_i and vertex v_i with normals n_i and n_{i+1}. The event points then correspond to the points on the edge which intersect with the half lines starting at v_i with directions n_i and n_{i+1}, respectively.

For efficiency reasons, we compute the Voronoi edge and its set of closest points using graphics hardware. Let $P = p_1, \cdots, p_n$ correspond to the coordinates of the pixels in the frame buffer tracing the Voronoi edge [20]. Since the edge is determined by two obstacles, the left and right obstacles can be retrieved by looking at the *local direction* determined by any two adjacent points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$. For instance, the corresponding direction is upwards if $y_2 < y_1$. Hence, its left obstacle can be retrieved by considering the obstacle with the id stored at pixel $(x_1 - 1, y_1)$ in the frame buffer. For the right obstacle we need to look at pixel (x_1, y_1). The id can be retrieved similarly if the direction is leftwards, rightwards, or downwards.

Now that we know the coordinates of the Voronoi edge and

[2]This arc can be represented by three control points defining a quadratic Bézier spline with control points B_1, the intersection of the *tangents* at B_1 and B_2, and point B_2. The tangent at B_1 is given by the line through B_1 and the midpoint between points l_1 and r_1. For B_2, we have the line through B_2 and the midpoint between points l_2 and r_2.

its left and right closest obstacle, we can compute the event points S together with their closest points using simple linear algebra. Note that we do not have to store the intermediate points on the edge in between two event points since the corresponding curve is determined by their closest points.

The storage complexity of a GVD is linear in the number of obstacle vertices $|v|$ [22]. Since each obstacle (with m vertices) yields at most two edges and we have at most $O(m)$ events per obstacle, the storage complexity of the ECM is also linear, i.e. $O(|v|)$. Consequently, the ECM is an efficient data structure, being able to represent large environments.

While the graphics card can efficiently compute a GVD, it may introduce errors in the ECM due to being limited to a finite resolution while two obstacles can be placed arbitrarily close to each other. Such an error manifest itself as an edge not being found (when the distance between them is less than the width of a few pixels). In practical problems however, this is not an issue because we can use high enough resolutions. Such a high resolution can be obtained by rendering the GVD in several blocks while concatenating the resulting ECM's. Furthermore, if an edge is found, the obstacle boundaries are traced exactly while their vertices may be closer to each other than the width of a pixel.

C. The Explicit Corridor

For finding the global route of a character, an *Explicit Corridor* is extracted from the Explicit Corridor Map. We refer the reader to [20] for more details on this procedure. The elements of the corridor can be a part of a single edge, or can be a concatenation of edges forming a chain in the graph. The Explicit Corridor is defined as follows:

Definition 2 (Explicit Corridor)**.** *An Explicit Corridor is the sequence (B_i, R_i, l_i, r_i), where (B_i, R_i) is the sequence of maximum clearance disks with center points B_i and radii R_i $(1 \leq i \leq n)$. Each center B_i is assigned its left (l_i) and right (r_i) closest point to the obstacles' boundaries.*

While the corridor is represented as a discrete sequence of annotated disks, their closest points embody a continuous sequence (i.e. an infinite number of maximum clearance disks if $n > 1$), forming a continuous corridor.

The boundaries of this corridor will play an important role in efficiently computing the shortest path with a variable amount of clearance to the obstacles. Below, we will show that these boundaries are represented explicitly by the closest points and center points, hence the name Explicit corridor.

Let (B_i, R_i, l_i, r_i) be an Explicit Corridor $(1 \leq i \leq n)$, and $\mathcal{BC}_l$ and $\mathcal{BC}_r$ denote the left and right sides of the corridor's boundaries, respectively. Both sides consist of a beginning, a middle, and an ending part. We will only give the construction of the right side because the left side is constructed similarly. We refer the reader to Fig. 3 for a visual impression of the construction.

The beginning part of $\mathcal{BC}_r$ is the counter-clockwise arc, tracing disk (B_1, R_1), from s to r_1. The start point s lies exactly in the middle of the arc connecting l_1 to r_1.

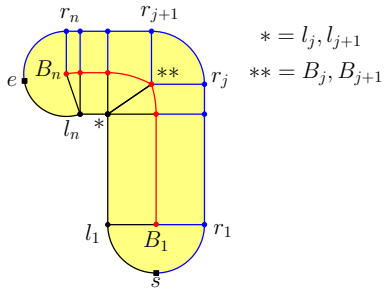
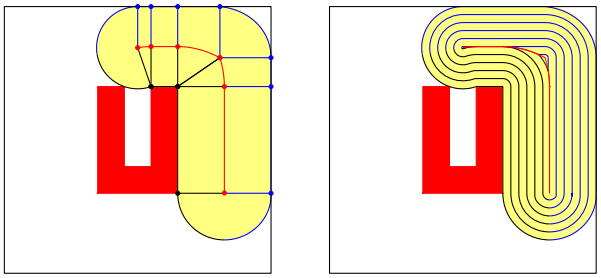


Fig. 3. Constructing the boundaries of an Explicit Corridor.



(a) Explicit Corridor (b) Different shrunk corridors

Fig. 4. Acquiring an amount of minimum clearance in an Explicit Corridor.

Similarly, the ending part of $\mathcal{BC}_r$ is the counter-clockwise arc, tracing disk (B_n, R_n), from r_n to e. The end point e lies exactly in the middle of the arc connecting r_n to l_n.

The middle part of $\mathcal{BC}_r$ consists of a concatenation of curves. The type of the curve is dependent on the tuple under consideration. Let (B', R', l', r') and (B'', R'', l'', r'') be a pair of adjacent tuples of an Explicit Corridor.

We first look at the situation in which disks from a pair of incident edges were concatenated. While these disks have the same coordinates, they can have different (right) closest points, i.e. $B' = B'' \wedge r' \neq r''$. The corresponding part of $\mathcal{BC}_r$ is the counter-clockwise arc, tracing disk (B', R'), from r' to r''. In Fig. 3, this happens at disks j and $j+1$. The right boundary part is a circular arc from r_j to r_{j+1}. Note that the left boundary part is a degenerated circular arc, consisting of a single point, because $l_i = l_{i+1}$.

If a pair of adjacent disks is taken from the same edge, the boundary is the line segment that traces an obstacle from point r' to r''. Note that the boundary cannot intersect an obstacle because adjacent closest points always lie on the same edge of the obstacle.

The construction shows that we can compute the left and right boundary parts for any pair of adjacent disks in constant time. Consequently, the corridor's boundaries can be computed in $O(n)$ time.

III. Minimum Clearance in Explicit Corridors

In many path planning applications, a path should keep some amount of minimum clearance, cl_{min}, to the obstacles. Traveling along such a path avoids collisions with these obstacles. We will show in this section how to shrink the corridor such that the distance from each point on the

corridor's boundaries to any obstacle is at least cl_{min}. A set of shrunk corridors is displayed in Fig. 4.

For an Explicit Corridor (B_i, R_i, l_i, r_i) it holds that the distance from a closest point $cp_i \in \{l_i, r_i\}$ to the obstacles is zero and that the distance between the center B_i and closest point cp_i is R_i. For any point p on the line segment through B_i and cp_i we have that cp_i is the closest point to p. Hence, we shrink the corridor by moving cp_i on this segment to B_i. The coordinates of cp_i are set to $cp_i + cl_{min} * (B_i - cp_i)/||B_i - cp_i||$ and its radius equals $R_i - cl_{min}$. However, if $cl_{min} > R_i$, we have the following two cases. If $cl_{min} > R_i$ and $cl_{min} > R_{i+1}$, then $cp_i = B_i$ and $cp_{i+1} = B_{i+1}$; their clearances are set to zero. If one of the corresponding radii is larger than cl_{min} but the other one is smaller, we need to insert *one disk* if the corresponding curve is a line, and *one or two* disks if the curve is a parabolic arc. The point(s) are placed at the position(s) where their clearance equals cl_{min}.

The boundaries of the shrunk corridor are constructed like the boundaries of the original corridor. There is a small difference though. In the original corridor, a sequence of multiple center points can have closest points which have the same coordinates c, see e.g. points l_j and l_{j+1} in Fig. 3. These closest points are mapped onto a circular arc when the corridor is shrunk. Hence, instead of line segments, they must be connected with a circular arc which traces a disk with center c and radius cl_{min}.

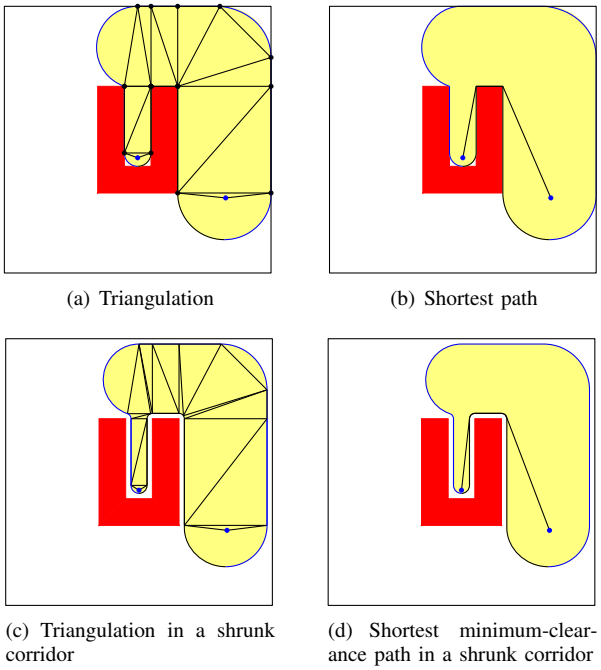
Because it takes constant time per update of each closest point, a shrunk Corridor, whose boundaries have a clearance of at least cl_{min} to the obstacles (wherever possible), can be computed in linear time in the number of disks.

IV. Shortest Paths

In many path planning applications, a path should be short because it usually takes the least time to traverse. We will show in this section how to create the shortest path inside an Explicit Corridor in linear time in the number of disks.[3]

The corridor's boundaries can be considered as a polygon (with circular arcs). Given a (simple) polygon, the shortest path from a start to a goal position can be computed in linear time in the number of its vertices [25]. First, a triangulation of the polygon is computed. Then, the dual graph of this triangulation is determined. In this graph, a vertex is placed in the face of each triangle. An edge in the graph connects two vertices if they share an edge in the triangulation. After computing the Depth First Search (DFS) on the dual graph at the start position, the sequence of diagonals that are crossed by the DFS is known. These diagonals are used by their 'Funnel' algorithm which computes the shortest path. In [26], this algorithm is extended such that triangles are allowed to have circular arcs.

The most difficult step in the algorithm is the triangulation. Theoretically, a polygon can be triangulated in linear time [27]. While being impressive, this algorithm is complex and may have never been implemented. Fortunately, it is easy to build the triangulation with at most $2n+2$ triangles by using the Explicit Corridor (B_i, R_i, l_i, r_i), where $1 \leq i \leq n$. An example of such a triangulation is displayed in Fig. 5(a).



(a) Triangulation (b) Shortest path

(c) Triangulation in a shrunk corridor (d) Shortest minimum-clearance path in a shrunk corridor

Fig. 5. Computing the shortest path in an Explicit Corridor.

The triangulation is constructed by connecting adjacent closest points. That is, each pair of adjacent left closest points (l_j and l_{j+1}) and right closest points (r_j and r_{j+1}) contributes the $2j^{th}$ and $2j+1^{th}$ triangle, i.e, (l_j, r_j, l_{j+1}) and (r_j, l_{j+1}, r_{j+1}), respectively. If $l_j = l_{j+1}$ or $r_j = r_{j+1}$, the corresponding triangle is a line and can be discarded. If the start position s is not included in one of the triangles, we add triangle (s, l_1, r_1) to the front of the triangle list. Likewise, if the goal position g is not included, we add triangle (l_n, r_n, g) to the back of the list.

This triangulation is valid since each triangle shares two edges (except the first and last one). Each triangle is empty because each edge traces the boundary of an obstacle, lies inside one disk, or splits an empty convex quadrilateral.

The triangulation takes $O(n)$ time because we spend constant time per closest point. We can now directly apply the extended Funnel algorithm from [26] because the dual graph of our triangulation is a list (and applying a DFS on a list yields the same list). Since this algorithm takes $O(n)$ time, our algorithm is linear in the number of disks.

A resulting shortest path is displayed in Fig. 5(b). Note that this path can be traversed only by a point (and not by a character) because it is the shortest one in the corridor, touching the obstacles. We just need some clearance.

Shortest minimum-clearance paths

For a character with radius r, we want to compute the shortest path having a desired amount of minimal clearance cl_{safe}. The path can be obtained by first shrinking the corridor with clearance $cl_{min} = r + cl_{safe}$ and then computing

[3]When we refer to the shortest path we mean the shortest path lying in the same homotopic class as the curves in the corridor.

the shortest path for a point in the shrunk corridor. We refer the reader to Fig. 5(c) and Fig. 5(d) for an example. A triangulation was computed inside a shrunk corridor. Next, the shortest path was computed based on this triangulation.

While the path is short and safe, it is rigid because the character's motions are limited to the computed path. In the next section we show how more flexibility can be obtained which is necessary when dealing with smoothness and dynamic character avoidance.

V. Path Planning

Recently, the Indicative Route Method (IRM) has been introduced [19]. The IRM is a framework for real-time path planning in interactive virtual environments. The method directs the global motions of a character traversing a corridor. Such a corridor is extracted from the Corridor Map. Local motions are controlled by potential fields inside a corridor, providing the desired flexibility, including smoothness and obstacle avoidance. The potential field is defined such that the character stays inside the corridor, locally adjusts its path if necessary and is led to the goal. For this purpose, an *attraction point* is defined which attracts the character and moves on a *control path* toward the goal. A good candidate for the control path is the shortest minimum-clearance path. The attraction point attracts the character at position x with force $\mathbf{F}_a(x)$. Obstacle avoidance is included by adding an extra force $\mathbf{F}_o(x)$. The total force, applied to the character (with mass m), is then $\mathbf{F}(x) = \mathbf{F}_a(x) + \mathbf{F}_o(x)$. The differential equation $m\frac{d^2x}{dt^2} = \mathbf{F}_a(x) + \mathbf{F}_o(x)$ gives the positions of the character's path. Because this non-linear equation cannot be solved analytically, a numerical approximation must be used to compute the path, such as Verlet integration [28].

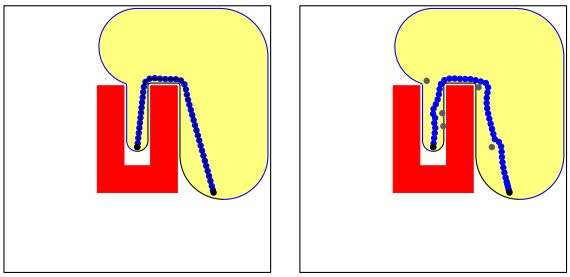
We refer the reader to Fig. 6 which displays visually convincing short paths. Both paths keep some minimum clearance to the corridor's boundaries and try to follow the shortest path. The first one is a smooth path (i.e. C1-continuous). The second one avoids the gray obstacles while it stays smooth and short.

VI. Experiments

In this section, we will describe the experiments we have conducted on Explicit Corridors. In particular, we investigated the efficiency of computing the Explicit Corridor Map, Explicit Corridors, triangulations, and smooth shortest minimum-clearance paths. All the experiments were tested on a large virtual city and were run on a PC with a NVIDIA GeForce 8800 GTX graphics card and an Intel Core2 Quad CPU (2.4 GHz) with 4 GB memory. Only one core was used.

A. Experimental setup

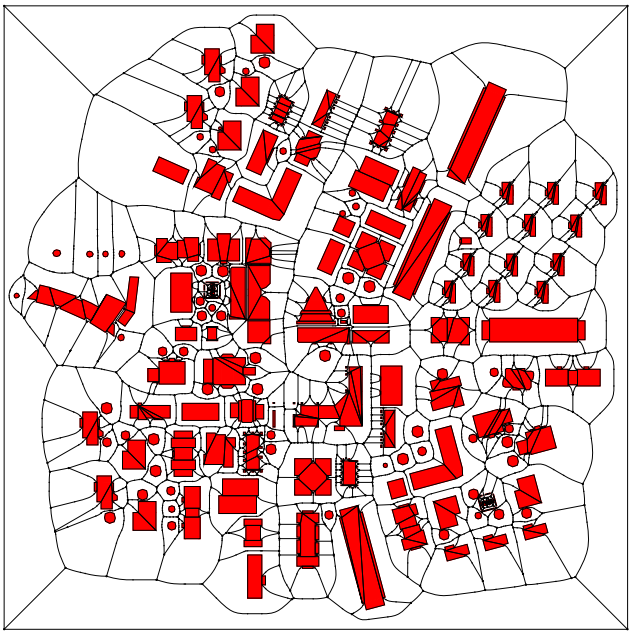
We have conducted experiments with the city environment depicted in Fig. 7(a). The city measured 500x500 meter and was populated with 181 buildings and trees. We created a corresponding footprint. Non-convex polygons were decomposed into convex ones by using the CGAL package [24], which took $1s$. This decomposition is visualized in



(a) Smooth short path (b) Obstacle avoidance

Fig. 6. Flexible path planning.

(a) City environment



(b) Footprint and Explicit Corridor Map

Fig. 7. The test environment, its footprints and Explicit Corridor Map. For clarity, the closest points have been left out. A close-up of these closest points is shown in Fig. 8(a).

Fig. 7(b) as black line segments which are overlaid on the footprint. The resulting 548 convex polygons, some of which were degenerated into a point or line segment, formed the input obstacles of the Explicit Corridor Map. This map was created using a resolution of 4,000x4,000 pixels by the application developed in [20]. We have extended this application to include computing the Explicit Corridor Map. Next, we have integrated the techniques from this paper into the IRM framework [19]. The applications were implemented in Visual C++ under Windows XP.

First, we have conducted experiments to measure the running times for constructing the Explicit Corridor Map.

Next, we measured the average running times of constructing 1,000 random Explicit Corridors and their shortest (minimum-clearance) paths. When we refer to the construction time of a path, it embodies the whole query phase, including querying the map, computing the Explicit Corridor, shrinking the corridor (by 1 meter), building a triangulation and computing the shortest (smooth) path inside the shrunk corridor. To see the resulting times in the right perspective, we defined the CPU-load, which was the spent CPU time / averaged traversed time $*$ 100%. The traversed time was defined as the traveled distance (m) divided by the character's average speed (m/s), which was set to a fast walking pace of $2m/s$.

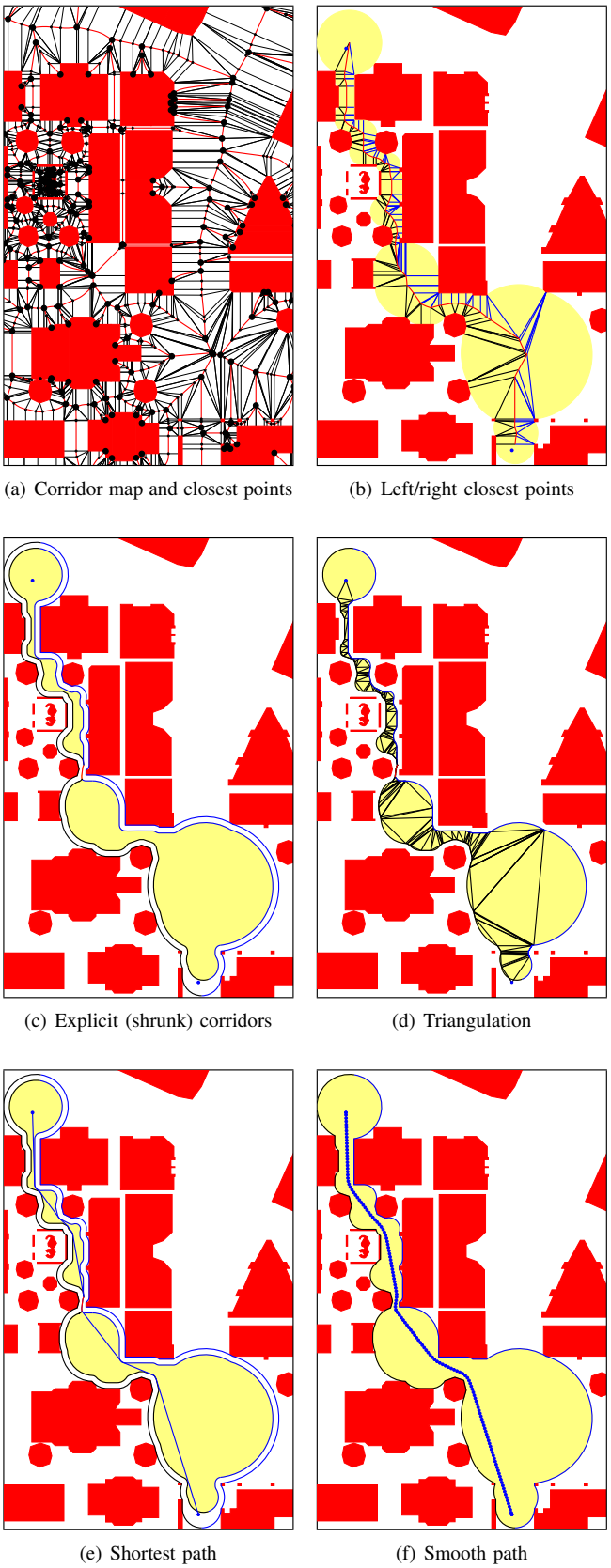
Finally, we picked one representative corridor and corresponding path and studied its construction and quality.

B. Experimental results

The Explicit Corridor Map (ECM) was computed in $0.60s$, resulting in 1,439 vertices and 1,618 edges containing 6,322 disks and pairs of closest points. While computing the map is meant to be done in an off-line construction phase, the method is fast enough to be applied in an on-line setting such as a game in which a level is created by a player. Edges ran in non-convex parts because all input obstacles were convex, ensuring that characters could visit those places.

Next, we extracted 1,000 random corridors from the map. The average running time for extracting an Explicit Corridor was $1.19ms$. Shrinking the corridor added $0.54ms$. Triangulating this corridor and computing the shortest path added another $1.83ms$. Hence, on average, the total time for computing the shortest (minimum-clearance) path in a corridor was $3.56ms$. By using the resulting shortest paths as control paths, smooth paths were obtained, increasing the total time to $4.14ms$. By having an average traveled distance of $321m$, the CPU-load was 0.002% for steering a single character. Consequently, the approach is suitable for steering many characters simultaneously, as occurs in a crowd [29].

We refer the reader to Fig. 8, which shows the construction of a smooth short minimum-clearance path. First, the ECM was computed in the off-line construction phase. A close inspection confirms that the density of the event points is directly related to the complexity of the obstacles. The varying density yielded a small corridor map while paths inside narrow corridors could still be traversed. Next, in the on-line query phase, a representative corridor was extracted.



(a) Corridor map and closest points (b) Left/right closest points

(c) Explicit (shrunk) corridors (d) Triangulation

(e) Shortest path (f) Smooth path

Fig. 8. Construction of a smooth, short minimum-clearance path.

By exploiting its left and right closest points, an Explicit Corridor was obtained. We created a shrunk corridor having a high clearance ($cl_{min} = 2.25m$) to the obstacles for visualization purposes. Note that this clearance was larger than the available clearance in the middle part of the corridor. In this part, the shortest path ran on the Voronoi edges induced by the obstacles. This shortest path was created by triangulating the shrunk corridor before it was fed to the extended Funnel algorithm. Finally, the shortest minimum-clearance path was used as a control path to generate a smooth path. This path, traversed by a character (modeled by a disk with radius $0.4m$), was short, smooth, and did not run too close to the obstacles. The query phase for this example took $2.77ms$, making the approach feasible for efficiently constructing visually convincing short paths.

VII. Conclusion

We have proposed a data structure and algorithms for computing the shortest path with a desired variable amount of minimum clearance inside corridors. The walkable space of the environment was represented by a new structure, referred to as the Explicit Corridor Map. A corridor from this structure was defined as a sequence of maximal empty disks, annotated with closest points lying on the obstacles in the environment. This annotation enabled constructing the corridor's boundaries explicitly, allowing for computing the shortest path with clearance in linear time in the number of disks of the corridor. Our main contribution is a combination of two related features: simplicity of the approach together with its practical efficiency.

While the resulting paths were short and safe, they were fixed in response to a query, leading to possible collisions with other characters and predictability of the characters' motions [19]. The desired flexibility was obtained by using the path as control path in the Indicative Route Method. By following an attraction point, which moved along this control path, a smooth, visually convincing short path for the character was obtained. Experiments showed that this approach led to a CPU-load of 0.002 percent, showing that the algorithm can be used for real-time high-quality path planning in the plane.

We think that our data structure and algorithms can form the basis for solving many challenging applications, including crowd planning, camera planning and stealth planning.

References

[1] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, 1st ed. MIT Press, 2005.

[2] J.-C. Latombe, *Robot Motion Planning*. Kluwer, 1991.

[3] S. LaValle, *Planning Algorithms*. Cambridge University Press, 2006. [Online]. Available: http://planning.cs.uiuc.edu

[4] P. Hart, N. 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, 1968.

[5] J. Barraquand, "Automatic motion planning for complex articulated bodies," Paris Research Laboratory, Tech. Rep., 1991.

[6] C. Connolly and R. Grupen, "Harmonic control," in *IEEE International Symposium on Intelligent Control*, 1998, pp. 503–506.

[7] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 995–1001.

[8] L. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, 1996.

[9] J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan, "A random sampling scheme for path planning," *International Journal of Robotics Research*, vol. 16, pp. 759–744, 1997.

[10] R. Geraerts and M. Overmars, "Creating high-quality paths for motion planning," *International Journal of Robotics Research*, vol. 26, pp. 845–863, 2007.

[11] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha, "Fast computation of generalized voronoi diagrams using graphics hardware," in *International Conference on Computer Graphics and Interactive Techniques*, 1999, pp. 277–286.

[12] H. Rohnert, "Moving a disc between polygons," *Algorithmica*, vol. 6, pp. 182–191, 1991.

[13] L. Chew, "Planning the shortest path for a disc in O($n^2 \log n$) time," *Annual Symposium on Computational Geometry*, pp. 214–220, 1985.

[14] J. Storer and J. Reif, "Shortest paths in the plane with polygonal obstacles," *Journal of the ACM*, vol. 41, pp. 982–1012, 1994.

[15] A. Kamphuis and M. Overmars, "Finding paths for coherent groups using clearance," in *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, 2004, pp. 19–28.

[16] J. Pettré, P. de Heras Ciechomski, J. Maïm, B. Yersin, J.-P. Laumond, and D. Thalmann, "Real-time navigating crowds: Scalable simulation and rendering," *Computer Animation and Virtual Worlds*, vol. 17, pp. 445–455, 2006.

[17] R. Geraerts and M. Overmars, "The corridor map method: A general framework for real-time high-quality path planning," *Computer Animation and Virtual Worlds*, vol. 18, pp. 107–119, 2007.

[18] R. Wein, J.P. van den. Berg, and D. Halperin, "The visibility-voronoi complex and its applications," *Computational Geometry: Theory and Applications*, vol. 36, pp. 66–87, 2007.

[19] I. Karamouzas, R. Geraerts, and M. Overmars, "Indicative routes for path planning and crowd simulation." in *The Fourth International Conference on the Foundations of Digital Games*, 2009, pp. 113–120.

[20] R. Geraerts and M. Overmars, "Enhancing corridor maps for real-time path planning in virtual environments," in *Computer Animation and Social Agents*, 2008, pp. 64–71.

[21] H. Choset and J. Burdick, "Sensor-based exploration: The hierarchical generalized Voronoi graph," *International Journal of Robotics Research*, vol. 19, pp. 96–125, 2000.

[22] M. de. Berg, M. van. Kreveld, M. Overmars, and M. Schwarzkopf, *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2000.

[23] M. Keil and J. Snoeyink, "On the time bound for convex decomposition of simple polygons," *International Journal of Computational Geometry and Applications*, vol. 12, pp. 181–192, 2002.

[24] S. Hert, "2d polygon partitioning," in *CGAL User and Reference Manual*, 3rd ed., C. E. Board, Ed., 2007. [Online]. Available: http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/packages.html

[25] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan, "Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons," *Algorithmica*, vol. 2, pp. 209–233, 1987.

[26] D. Demyen and M. Buro, "Efficient triangulation-based pathfinding," in *21st National conference on Artificial intelligence*, 2006, pp. 942–947.

[27] B. Chazelle, "Triangulating a simple polygon in linear time," *Discrete & Computational Geometry*, vol. 6, pp. 485–524, 1991.

[28] J. Butcher, *Numerical Methods for Ordinary Differential Equations*. Wiley, 2003.

[29] R. Geraerts, A. Kamphuis, I. Karamouzas, and M. Overmars, "Using the corridor map method for path planning for a large number of characters," in *Motion in Games (MIG'08)*, ser. Lecture Notes in Computer Science, vol. 5277. Springer, 2008, pp. 11–22.