To appear in ACM Transactions on Graphics (TOG), volume 33, issue 5, 2014 - This version is the author's manuscript.

Dynamic and Robust Local Clearance Triangulations

Marcelo Kallmann
University of California, Merced

The Local Clearance Triangulation (LCT) of polygonal obstacles is a cell decomposition designed for the efficient computation of locally shortest paths with clearance. This paper presents a revised definition of LCTs, new theoretical results and optimizations, and new algorithms introducing dynamic updates and robustness. Given an input obstacle set with n vertices, a theoretical analysis is proposed showing that LCTs generate a triangular decomposition of $O(n)$ cells, guaranteeing that discrete search algorithms can compute paths in optimal times. In addition, several examples are presented indicating that the number of triangles is low in practice, close to $2n$, and a new technique is described for reducing the number of triangles when the maximum query clearance is known in advance. Algorithms for repairing the local clearance property dynamically are also introduced, leading to efficient LCT updates for addressing dynamic changes in the obstacle set. Dynamic updates automatically handle intersecting and overlapping segments with guaranteed robustness, using techniques that combine one exact geometric predicate with adjustment of illegal floating point coordinates. The presented results demonstrate that LCTs are efficient and highly flexible for representing dynamic polygonal environments with clearance information.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Geometric algorithms, languages, and systems*; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Graphics data structures and data types*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*

General Terms: Algorithms

Additional Key Words and Phrases: Path Planning, Navigation Meshes, Character Navigation

1. INTRODUCTION

Efficient path planning and navigation in virtual environments remains a central problem in many areas of computer animation. One important class of applications is related to computer games and simulation of autonomous agents [Shao and Terzopoulos 2005],

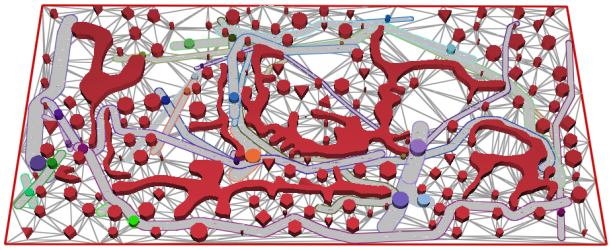

Fig. 1. The LCT of this environment enables the efficient computation of paths with arbitrary clearance.

where efficiency and flexibility of use are important requirements. The design of a powerful solution starts with the underlying environment representation, which plays a significant role in the types of paths that can be computed, in the performance of maintenance operations, and in the additional navigation queries that can be supported.

Local Clearance Triangulations (LCTs) achieve unique capabilities as a navigation mesh structure. They are computed by refinement operations on a Constrained Delaunay Triangulation of the input obstacle set. The refinements are designed to ensure that two local clearance values stored per edge are sufficient to precisely determine if a disc of arbitrary size can pass through any narrow passages of the mesh. This property is essential for the correct and efficient extraction of paths with clearance directly from the triangulation, without the need to represent the medial axis.

LCTs exactly conform to any given set of polygonal obstacles and common degeneracies such as polygon overlaps and intersections can be robustly handled. LCTs are well suited for supporting generic navigation and environment-related computations, such as for computing free corridors, visibility, accessibility and proximity queries. LCTs were proposed in previous work [Kallmann 2010] and this paper presents 1) a necessary revision of basic definitions, 2) new theoretical proofs demonstrating the key properties of the structure and related algorithms, and 3) new algorithms for addressing dynamic updates, robustness, and reduced refinements in cases where the maximum query clearance is known in advance.

2. RELATED WORK

Character navigation in complex environments may involve multiple aspects, from perception and behavioral modeling to collision avoidance and group interactions [Shao and Terzopoulos 2005; Kuffner and Latombe 1999; Metoyer and Hodgins 2003; Noser and Thalmann 1995]. For instance, interesting structures such as elastic roadmaps [Gayle et al. 2009] and multi agent navigation graphs [Sud et al. 2008] have been proposed for maintaining agent relationships during navigation. While these and other similar types of work address important topics related to character navigation, the focus is often on the behaviors to be achieved and not on the efficient environment representation and path computation. The related work analysis that follows focuses on these aspects and specifically reviews prior work on path planning with clearance.

Author's address: University of California, Merced, 5200 N. Lake Road - Merced CA 95343; email: mkallmann@ucmerced.edu.

DOI 10.1145/
http://doi.acm.org/10.1145/

2.1 Traditional Approaches to Path Planning

Grids are classical representations for path planning and they have been extensively used for computing paths for virtual characters [Shao and Terzopoulos 2005]. Grids are robust and simple to implement, and can be easily integrated with discrete search methods such as A* [Hart et al. 2007], D*-Lite [Koenig and Likhachev 2002], ARA* [Likhachev et al. 2003], etc. Unfortunately, grids do not represent polygonal obstacles precisely, and the computation time and solution quality greatly depend on the chosen grid resolution. Fine resolutions can produce high quality paths but quickly become prohibitive for large environments.

Polygonal representations are in general more efficient because they can generate a reduced and resolution-free set of cells decomposing the environment, therefore greatly improving the performance of discrete search methods. Path planning on polygonal representations is a classical topic studied in computational geometry and the problem of computing globally shortest paths from polygonal obstacles, or Euclidean shortest paths, has received significant attention due its importance in many applications.

Euclidean Shortest Paths Probably the most well-known approach for computing Euclidean shortest paths among polygonal obstacles is to build and search the *visibility graph* [Nilsson 1969; Lozano-Pérez and Wesley 1979; De Berg et al. 2008] of the obstacles. This can be achieved in $O(n^2)$ time [Overmars and Welzl 1988; Storer and Reif 1994], where n is the total number of vertices in the obstacles. The Euclidean shortest path problem can however be solved in sub-quadratic time [Mitchell 1993] and an algorithm running in $O(n \log n)$ time is available [Hershberger and Suri 1997]. The approach is based on the *continuous Dijkstra* paradigm, which simulates the propagation of a wavefront maintaining equal length to the source point, until the goal point is reached. After the environment is processed in $O(n \log n)$ for a given source point, paths to any destination can be computed in $O(\log n)$.

In practice, algorithms suitable for implementation remain related to visibility graphs, and extensions for computing globally shortest paths with arbitrary clearance have been proposed [Chew 1985; Liu and Arimoto 1995; Wein et al. 2007]. However, the computation and query times of existing methods remain at least $O(n^2)$. The LCT representation does not address the computation of globally shortest paths and instead focuses on computing locally shortest paths efficiently.

Medial Axis If the desired path does not need to be the global optimal, one popular approach for computing paths with clearance is to search the medial axis graph of the environment [Bhattacharya and Gavrilova 2008; Geraerts 2010]. The medial axis can be computed from the Voronoi diagram of the environment, and methods based on hardware acceleration have been developed to improve computation times [Hoff et al. 2000].

One benefit of explicitly representing the medial axis is that locally shortest paths can be easily interpolated towards the medial axis in order to reach maximum clearance when needed. In contrast, LCTs offer a triangular mesh decomposition that carries just enough clearance information to be able to compute paths of arbitrary clearance, without the need to represent the intricate shapes the medial axis can have. As a result, the LCT decomposition graph uses less nodes to represent a given environment. An example comparison (shown in Figure 16) is discussed in Section 8.

2.2 Triangulations

Triangulations offer a natural approach for cell decomposition and they have been employed for path planning in varied ways. Kapoor et al. [1997] have explored the reduction of a triangulated environment in corridors and junctions in order to compute the relevant subgraph of the visibility graph for a given path query. The method computes globally optimal paths in $O(n+h^2 \log n)$, where h is the number of holes in the environment. Without the goal of computing globally shortest solutions, several methods have employed the Constrained Delaunay Triangulation (CDT) as a cell decomposition for discrete search. Whenever a CDT is kept with $O(n)$ cells, discrete search algorithms can compute channels (or corridors) containing a solution path in optimal times. The *funnel algorithm* [Chazelle 1982; Lee and Preparata 1984; Hershberger and Snoeyink 1994] has emerged as an efficient way to extract the shortest path inside a triangulated channel [Kallmann et al. 2003; Demyen 2007; Geraerts 2010].

Techniques for handling clearance have also been explored. One approach to capture the width of a corridor is to refine constrained edges that have orthogonal projections of vertices on the opposite side of a corridor, adding new free CDT edges with length equal to the width of the corridor [Lamarche and Donikian 2004]. However, such a refinement can only address simple corridors and the total number of vertices added to the CDT can be significant. A more generic approach is to compute a measure of clearance per traversed triangle, for example by computing the distance between every triangle corner and the closest constraint behind the edge opposite to the corner. This is the measure used in the LCT (see Figure 2), and an equivalent measure was used before in CDTs during path search [Demyen and Buro 2006; Demyen 2007]. However, as shown in Figure 6, this measure does not correctly handle all cases in a CDT.

The LCT decomposition provides a solution for correctly determining clearance in a triangulation with straight edges. The approach is based on a novel type of refinement operation, and clearance values can be pre-computed and stored in free edges so that on-line clearance tests are reduced to a simple value comparison per traversed edge.

Extensions based on the interconnection of floor plans in multilayer and non-planar environments have also been developed in order to address 3D scenes [Lamarche 2009; Jorgensen and Lamarche 2011; Oliva and Pelechano 2013]. Such techniques can be directly applied to the proposed LCT decomposition.

Triangulation Refinement The proposed approach of triangulation refinement is inspired by solutions developed in the area of mesh generation for finite element analysis, where triangulations are refined to adaptively represent polygonal regions with well-shaped triangles [Shewchuk 1996]. Here, refinements are used to subdivide triangles until a simple clearance test per triangle can be safely performed. Maintenance of refinements for supporting dynamic changes in the constraints is a topic that has not been addressed before. The proposed algorithms solve dynamic insertions and removals of constraints with local operations, first updating the underlying CDT, and then removing or adding LCT refinements as needed. Different strategies are presented for customizing the operations according to the relative number of path queries and dynamic updates in a given scenario.

Robustness One point that is important to be addressed is the robustness of the involved geometric algorithms. Simple implementations based on floating point representation are not sufficient for achieving robustness. A common approach is to rely on arbitrary precision representation and on exact geometric predicates [Shewchuk 1997; Devillers and Pion 2003], however imposing a significant performance penalty on the final system. Similarly to Held [2001], the presented approach provides a solution favoring speed of computation over accuracy. The presented solution is

based on floating point arithmetic and relies on a carefully designed combination of robustness tests and one exact geometric predicate. Robustness is guaranteed for any set of input polygons, which are handled on-line in any configuration. The presented solution is the first to robustly address intersecting constraints in a triangulation, it always converges when multiple consecutive intersections happen, and it does so in a watertight manner.

Conclusion In summary, LCTs introduce a new approach for modeling and computing navigation queries with clearance, and are able to well address key requirements: fast computations, clearance, robustness, and dynamic updates.

3. BACKGROUND AND OVERVIEW

Let $\mathcal{S} = \{s_1, s_2, ..., s_m\}$ be a set of m input segments describing polygonal obstacles. Segments in $\mathcal{S}$ may be isolated or may share endpoints forming closed or open polygons. The number of distinct endpoints is n and the set of all endpoints is denoted as $\mathcal{P}$. When inserted in a triangulation, the input segments are also called constraints.

The proposed method starts from a Constrained Delaunay Triangulation (CDT) of the input segments. Let T be a triangulation of $\mathcal{P}$, and consider two arbitrary vertices of T to be visible to each other if the segment connecting them does not intercept the interior of any constraint. Triangulation T will be a CDT of $\mathcal{S}$ if: 1) it enforces the constraints, i.e., all segments of $\mathcal{S}$ are also edges in T, and 2) it respects the *Delaunay criterion* as much as possible, i.e., the circumcircle of every triangle t of T contains no vertex in its interior which is visible from all three vertices of t.

Although $CDT(\mathcal{S})$ is already able to well represent a given environment, an additional property, the *local clearance property*, is needed in order to achieve correct and efficient clearance determination per triangle during path search. Whenever the local clearance property fails in $CDT(\mathcal{S})$, refinement operations on the input segments are performed for enforcing it. The result is called a *Local Clearance Triangulation* (LCT) of the input segments. Given the possible refinement operations, the edges in $\mathcal{S}$ may be subdivided into smaller segments forming a new set of constrained edges $\mathcal{S}_{ref}$. The refinement process results with $LCT(\mathcal{S}) = CDT(\mathcal{S}_{ref})$.

Two methods are presented for computing LCTs with refinements: global refinement operations (Section 4.1) are most suitable for computing the LCT of input segments from scratch, and local refinements (Section 5) achieve efficient dynamic updates of constraints in order to reflect dynamic changes in the obstacle set. Robustness in the involved geometric computations is addressed in Section 6.

Once $T = LCT(\mathcal{S})$ is computed, T becomes an efficient representation for computing free paths of arbitrary clearance. Let p and q be two points in $\mathbb{R}^2$. A path between p and q is considered free if it does not cross any constrained edge of T. A free path may cross several triangles sharing unconstrained edges and the union of all traversed triangles is called a *channel*.

A path of r clearance is called *locally optimal* if 1) it has clearance r from all constrained edges in T and 2) it cannot be reduced to a shorter path of clearance r on the same channel. Such a path is denoted π_r, and its channel C_r. Path π_r may or not be the globally shortest path. If no shorter path of clearance r can be found in all possible channels connecting the two endpoints, the path is then a *globally optimal* one, it is denoted as π_r^* and its channel is denoted as C_r^*.

Given $T = LCT(\mathcal{S})$, two arbitrary points $p, q \in \mathbb{R}^2$, and $r \in \mathbb{R}^+$, two main steps are needed in order to compute $\pi_r(p, q)$. First, a channel search over the adjacency graph of T is employed for finding $C_r(p, q)$, or determining that a channel of clearance r does not exist. Then, if a channel exists, $\pi_r(p, q)$ is computed in linear time with respect to the number of triangles in the channel. The overall search procedure is discussed in Section 7.

The next section presents a revision of the original LCT definitions [Kallmann 2010] in order to address possible cases where disturbances would not be properly detected. The revised disturbance definition now considers all possible configurations of edges between the disturbance and the traversal exit (Figure 3). This leads to an updated determination of refinements (Figure 5), and a new characterization of when disturbances can occur is also presented (Figure 4). These revisions are necessary for the correctness of the proposed methods and proofs.

4. LOCAL CLEARANCE TRIANGULATION

Let $\mathcal{S} = \{s_1, s_2, ..., s_m\}$ be the set of input segments and $T = CDT(\mathcal{S})$. Let π be a free path in T, and let t be a triangle in its channel such that t is not the first or the last triangle in the channel. In this case π will always traverse t by crossing two edges of t. Let a, b, c be the vertices of t and consider that π crosses t by first crossing edge ab and then bc. This particular traversal of t is denoted by τ_{abc}, where ab is the entrance edge and bc is the exit edge. The shared vertex b is called the traversal corner, and the traversal sector is defined as the circle sector between the entrance and exit edges, and of radius $min\{dist(a, b), dist(b, c)\}$, where $dist$ denotes the Euclidean distance. Edge ac is called the interior edge of the traversal. The local clearance of a traversal is now defined.

DEFINITION 1. (TRAVERSAL CLEARANCE.) *Given a traversal τ_{abc}, its clearance $cl(a, b, c)$ is the distance between the traversal corner b and the closest vertex or constrained edge intersecting its traversal sector.*

Because of the Delaunay criterion, a and c are the only vertices in the sector, and thus $cl(a, b, c) \leq min\{dist(a, b), dist(b, c)\}$. In case $cl(a, b, c)$ is determined by a constrained edge s crossing the traversal sector, as illustrated in Figure 2, then $cl(a, b, c) = dist(b, s)$ and s is the closest constraint to the traversal. If edge ac is constrained, then ac is the closest constraint and $cl(a, b, c) = dist(b, ac)$. If the traversal sector is not crossed by a constrained edge then $cl(a, b, c) = min\{dist(a, b), dist(b, c)\}$.

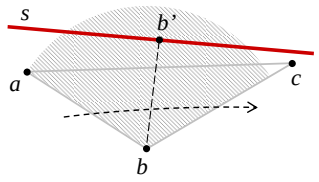

Fig. 2. The triangle traversal with entrance edge ab and exit edge bc is denoted as τ_{abc}. Segment s is the closest constraint crossing the sector of τ_{abc}, thus $cl(a, b, c) = dist(b, s) = dist(b, b')$, where b' is the orthogonal projection of b on s.

The closest constraint to a traversal is now formalized in order to take into account relevant constraints that may not cross the traversal sector of τ_{abc}.

DEFINITION 2. (CLOSEST CONSTRAINT.) *Given a traversal τ_{abc}, its closest constraint is the constrained edge s that is closest to the traversal corner b, such that s is either ac or s lies on the opposite side of ac with respect to b.*

In certain situations, the closest constraint of a traversal may generate narrow passages that are not captured by the clearance value of the traversal. The clearance value only accounts for the space occupied by the traversal sector. If a triangle happens to be too thin and long, other vertices not connected to the traversal may generate narrow passages that are not captured by any clearance value of the involved traversals.

The essence of the problem is that when a triangle is traversed it is not possible to know how the next traversals will take place: if the path will continue in the direction of a possibly long edge (and possibly encounter a narrower space ahead) or if the path will *rotate around* the traversal corner. Each case would require a different clearance value to be considered. For example, Figures 6a and 6c show examples of long CDT triangles where their clearance values are not enough to capture the clearance along the direction of their longest edges. The LCT refinements will fix this problem by detecting these undesired narrow passages and by breaking them down into sub-traversals until a single clearance value per traversal can handle all possible narrow passages. The vertices that cause undesired narrow passages are called disturbances, and they are defined below.

DEFINITION 3. (DISTURBANCE.) *Let τ_{abc} be a traversal in T such that its adjacent traversal τ_{bcd} is possible, i.e., edge cd is not constrained. Let s be the closest constraint to τ_{abc} and let v be a vertex on the opposite side of bc with respect to a. Among the vertices connected to v, let d and e be the ones forming $\triangle dve \in T$ crossed by segment vv', where v' is the orthogonal projection of v on s. In this situation, vertex v is a disturbance to traversal τ_{abc} if:*

1. *v is not shared by two collinear constraints,*
2. *v can be orthogonally projected on ac,*
3. *segment vv' crosses ac and bc,*
4. *$dist(v, s) < cl(a, b, c)$, and*
5. *$dist(v, s) < dist(v, e)$.*

Figure 3 illustrates the definition. A disturbance will always be paired with a constraint disturbing the traversal. A disturbed traversal may contain an arbitrary number of edges between bc and v, however, disturbed traversals will in most cases appear in simpler forms.

Disturbances can occur on any side of a triangle but only need to be defined with respect to the exit edge of a traversal. For example, disturbances on the left side of $\triangle abc$ in Figure 3 can occur with respect to τ_{cba}, but not τ_{abc}.

In certain configurations traversals cannot be disturbed. If vertex b does not have orthogonal projection in ac, traversals τ_{abc} and τ_{cba} cannot be disturbed. In addition, τ_{abc} can only be disturbed if its closest constraint s intersects its traversal sector S_1 or the symmetrical sector S_2, as defined in Figure 4-left. If S_1 and S_2 are not crossed by a constraint, τ_{abc} cannot have a disturbance because no vertex satisfying the conditions of Definition 3 will be closer to s than b and at the same time outside the empty circumcircle that protects the traversal from external vertices.

If a constraint s is found crossing S_1 or S_2, a disturbance is possible and a procedure to search for disturbances is needed. The procedure will traverse all edges crossing the disturbance region R of the traversal, and check if a vertex is found inside R. Figure 4-right illustrates the disturbance region R, which is delimited by segment bc, the line parallel to s and passing by b, and the orthogonal lines to s and ac passing by c. Region R encloses all points closer to s than b and with valid orthogonal projection on s. If a vertex v is found inside R and v satisfies conditions 1 and 5 of Definition 3,

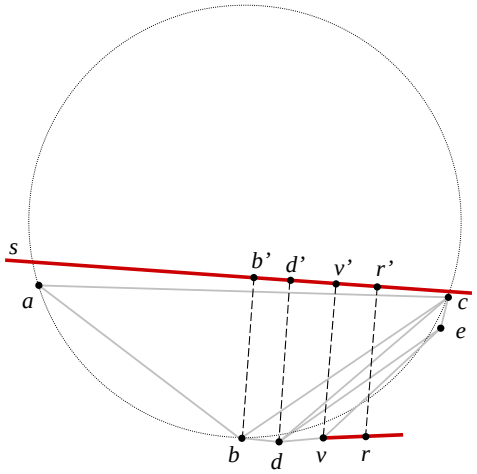

Fig. 3. The shown traversal τ_{abc} is disturbed by vertex v because $dist(v, v') < dist(b, b') = cl(a, b, c)$ and $dist(v, v') < dist(v, e)$. The dashed lines show the orthogonal projections of several vertices on s. Vertices d, e and r are not disturbances since $dist(d, d') > cl(a, b, c)$, $dist(e, s) > dist(e, c)$, and r is shared by two collinear constraints.

then v will be a disturbance. If no vertices are found inside R the traversal is clear.

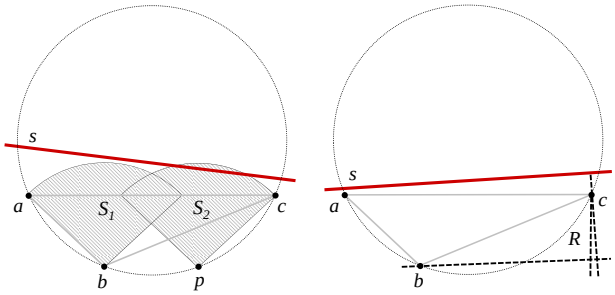

Fig. 4. A disturbance can only occur when a constraint s crosses the traversal sector S_1 or the symmetric sector S_2 (left diagram). If a disturbance is possible, it will be inside the disturbance region R, which encloses all points closer to s than b and with valid orthogonal projection on s (right diagram).

The local clearance triangulation (LCT) can be now defined with the following definitions.

DEFINITION 4. (LOCAL CLEARANCE.) *A traversal τ_{abc} in T has local clearance if it does not have disturbances.*

DEFINITION 5. (LCT.) *A Local Clearance Triangulation is a CDT with all traversals having local clearance.*

4.1 Computing LCTs by Global Refinements

The first approach for computing $LCT(\mathcal{S})$ is based on iterative refinements of disturbed traversals. The algorithm starts with the computation of triangulation $T_0 = CDT(\mathcal{S})$. A linear pass over all traversals of T_0 is then performed, and traversals detected to have a disturbance are refined with one subdivision point p_{ref} added to

the constraint associated with the disturbance. Each refinement operation is equivalent to one vertex insertion in the current CDT and can be implemented using the recursive Delaunay flips of the incremental CDT algorithm. Every time a constraint $s \in \mathcal{S}$ is refined, s is replaced by two new sub-segments. After all disturbed traversals are processed, a new (refined) set of constraints $\mathcal{S}_1$ is obtained. Triangulation $T_1 = CDT(\mathcal{S}_1)$ is the result of the first global refinement pass.

T_1 however may not be free of disturbances and the process has to be repeated k times, until $T_k = CDT(\mathcal{S}_k)$ is free of disturbances, in which case $\mathcal{S}_{ref} = \mathcal{S}_k$ and T_k is the desired $LCT(\mathcal{S})$. Since refinements are performed one at a time, the number of iterations k mainly depends on the existence of multiple disturbances with respect to a same constraint.

Let v' be the orthogonal projection of disturbance v on constraint s. A suitable refinement point p_{ref} for solving disturbance v with respect to τ_{abc} and s can be obtained with the mid-point of the intersections of s with the circle passing by vertices d, v and e, where dve is the CDT triangle crossed by segment vv', as shown in Figure 5-left. Most often v will be directly connected to b and c, and in such cases the circle passing by b, v and c is taken. In case of multiple disturbances, v is selected such that no other disturbance on the left side of vv' is closer to s. In such cases v is said to be the *first disturbance*.

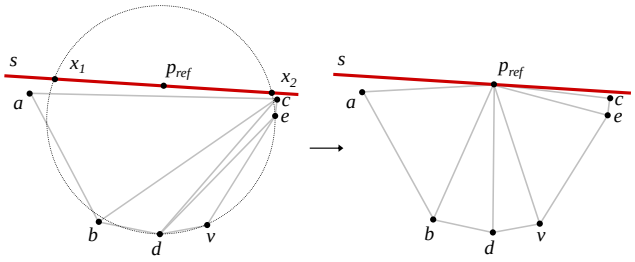

Fig. 5. Vertex v is a disturbance to traversal τ_{abc} and therefore constraint s is subdivided. Points x_1 and x_2 are the intersection points of s and the circle passing by d, v and e. The subdivision point p_{ref} is defined as the midpoint between x_1 and x_2. After refinement, all vertices between b and v will connect to p_{ref}.

The point of subdivision p_{ref} is carefully chosen in order to generate new traversals free from the original disturbance, and to ensure that the global refinement procedure converges. By making p_{ref} to be inside the circle passing by d, v and e, the refinement operation will cause p_{ref} to be connected to v, thus creating new traversals that will not be anymore disturbed by v. This is shown by Theorem 2 (in Appendix A).

The achieved local clearance property guarantees that a simple local clearance test per triangle traversal is enough for determining if a path π_r can traverse a channel without any intersections with constraints.

Given the desired clearance radius r, π_r will not have any intersections with constraints if $2r < cl(a,b,c)$ for all traversals τ_{abc} of its channel. Figure 6 presents examples where local clearance tests are not enough to produce correct results in CDTs, while correct results are always obtained in LCTs.

Local clearance tests per triangle are enough for determining if paths can traverse triangles, however clearance near end points require specific departure and arrival tests in order to ensure that a given path can depart or arrive at specific locations. These tests are explained in previous work [Kallmann 2010].

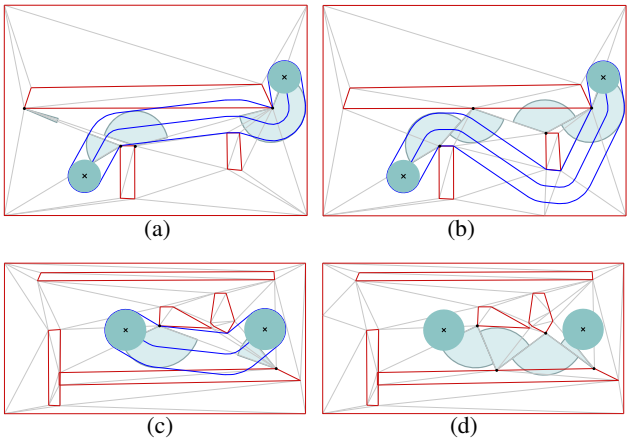

Fig. 6. Triangulations (a) and (c) are CDTs showing illegal paths that however satisfy their local clearance tests per traversed triangle. The traversal sectors are highlighted and they all have enough clearance. These examples show that local clearance tests per traversal are not enough in CDTs. However, once the existing disturbances are solved and the corresponding LCTs (b) and (d) are computed, local clearance tests become sufficient. In the top LCT example (b) a valid path passing by an alternate channel can still be found, however in the bottom LCT example (d) no solution with the given clearance exists.

4.2 Lazy Clearance Precomputation

Ensuring that local tests are enough is critical for achieving efficient search algorithms. By being local, the clearance test does not depend on adjacent traversals and therefore each traversal clearance value can be pre-computed and stored per edge of the triangulation. This reduces the local clearance test to a simple value comparison per traversal.

Given a traversal τ_{abc}, the computation of $cl(a,b,c)$ requires checking if there is a constrained edge s in the opposite side of ac with respect to b, such that $dist(b,s) < min\{dist(a,b), dist(b,c)\}$. Clearance values are precomputed and stored in the edges of the LCT. There are a total of 8 possible traversals passing by each edge, among them four pairs are symmetrical and only 4 traversals may have distinct values. Each traversal passes by two edges (the entrance and exit edges) and thus only 2 of the 4 values have to be stored per edge. Let bc be an edge of the LCT and a and d the remaining vertices of the two triangles sharing bc. The two values chosen to be stored at edge bc are the clearances of the traversals having bc as exit edge: $cl(a,b,c)$ and $cl(d,c,b)$.

Clearance values can be computed and stored during the LCT construction, however, each traversal refinement would require the update of all values associated with the affected edges. In addition, since a given edge may be refined (or affected) several times, unnecessary computation of intermediate values would happen.

An alternative approach is to initialize all precomputed values with a flag (or a negative value) indicating that the clearance values have not yet been computed. Then, clearance values are computed and stored as needed during path search queries. Every time a path search is launched, each clearance value that is not yet available will be computed and stored in its corresponding edge in order

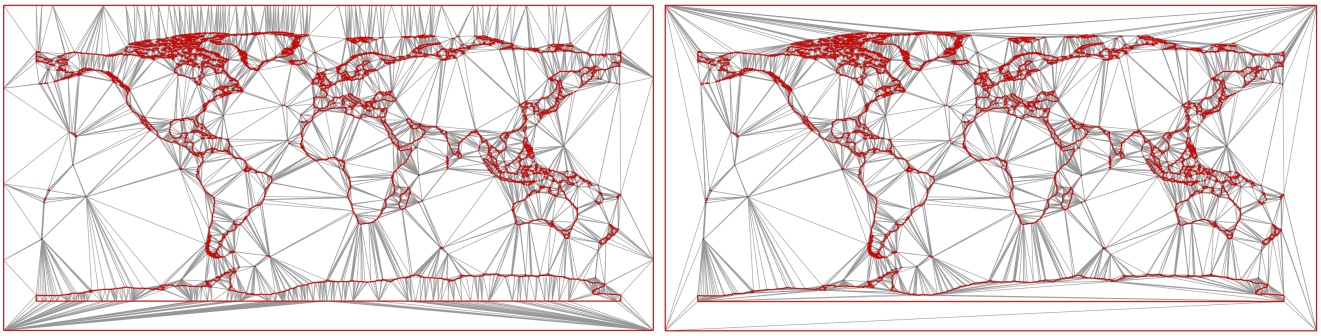

Fig. 7. This world map dataset has n=60374 and m=60379. Its LCT requires 1784 refinements (left). The optimized LCT for queries up 20 km clearance requires 1088 refinements (right), eliminating the need for the many refinements on the long horizontal edges. This environment involves large distances and with maximum clearance query values of 10 km and 1 km the number of refinements is reduced to 680 and 4 respectively.

to become readily available for subsequent queries. With this approach, clearance values are only computed in regions reachable by the path queries, avoiding computations in parts of the environment that are not used.

Lazy precomputation of clearance values is also a good strategy when there are dynamic LCT updates (Section 5). Clearance values associated with modified traversals can be simply marked as invalid, and later recomputed when needed by path queries. Dynamic LCT updates are local and will therefore also lead to a local invalidation of the affected clearance values.

It is tempting to develop a similar lazy strategy for the LCT refinement of traversals. However the problem is that, during a search query, already expanded triangles may have their shape and connectivity modified in a refinement, what could require an entire path search to be re-started to accommodate the changes. Of course, if parts of the environment are known to be never traversed (like in the interior of obstacles), refinements and clearance values do not have to be computed for them.

4.3 Bounded Clearance

One important optimization is to consider the local clearance property only up to a given maximum value M representing the maximum clearance allowed to be used in path queries. In most cases, M will be the clearance required by the largest agent that needs a path. The triangulation can be then optimized accordingly.

Let traversal τ_{abc} be disturbed with respect to disturbance v and constraint s. In order to perform the bounded clearance optimization, refinement operations are adapted to only refine τ_{abc} if $dist(v,s) < min\{cl(a,b,c), M\}$, instead of the original $dist(v,s) < cl(a,b,c)$ condition in Definition 3.

This optimization can greatly reduce the number of required refinements. The smaller is M, the smaller will be the number of refinements, leading to a faster computation of the corresponding LCT^M and to less cells processed during path search. See Figure 7 for an example.

4.4 Analysis

Four theorems are proposed in Appendix A in order to establish the size and correctness of LCTs. The total number of refinements is limited by the upper bound of $3n$, showing that the global refinement algorithm terminates and produces a $LCT(S)$ with $O(n)$ vertices. The bound of $3n$ translates to a cell decomposition of no more than $6n$ triangles since, using the Euler formula, $t = 2n - 2 - k \Rightarrow t < 2n$, where t is the number of triangles in a triangulation and k is the number of edges in the outer border ($k = 4$ in all presented environments). Examples are presented in Section 8 indicating that in practice the number of added vertices is much lower than the bound of $3n$ and that the number of triangles remains close to $2n$. It is also possible that a bound lower than $3n$ exists. While a vertex can disturb three traversals at the same time (see Appendix C) it is unlikely that all vertices can.

5. DYNAMIC UPDATES

Local refinement operations are important in order to achieve quick repairs in response to dynamic changes in the environment. See Figure 10 for an example.

Two dynamic operations are needed: insertion and removal of constraints. The approach described by Kallmann et al. [2003] is followed where a same id is associated to all the constraints forming one polygonal obstacle. The insertion routine will process all constraints of an obstacle at once, and then return the id that is assigned to the obstacle. Later, the removal routine can remove all constraints associated to a given obstacle id.

5.1 Local Insertion

Let $\mathcal{S}$ be the set of constraints being represented in $LCT(\mathcal{S})$ and $\mathcal{O}$ be a set of k segments describing a new polygonal obstacle. The local insertion of $\mathcal{O}$ in $LCT(\mathcal{S})$ is performed in three steps:

1. First, the k segments in $\mathcal{O}$ are inserted using regular incremental CDT operations and all modified vertices and constraints are stored in two lists: list V contains all adjacent vertices to modified edges (including edges modified due CDT swaps), and list C contains all edges that were constrained during the insertion.
2. Then, for each constrained edge s in C, a local search is performed to determine if s leads to disturbances. The search is performed by procedure *SearchDisturbances(s,V)*, which is detailed in Algorithm 2 and illustrated in Figure 9. The search will recursively visit and test all traversals that may have a disturbance caused by s, and all disturbances encountered are added to V (if not already in V).
3. Finally, all traversals influenced by the vertices in V will be tested with respect to the local clearance property and refined when needed, a process performed by procedure *LocalRef(V)*, as described below.

Procedure *LocalRef* is detailed in Algorithm 1. It identifies and tests all traversals that may need to be refined when a change occurs nearby the vertices in V. For each $v \in V$, all triangles around v are visited. Let t be the current triangle around v being processed. Two tests are performed (line 4 in Algorithm 1): *TriDisturbed(t)* examines if any of the six possible traversals of t needs to be refined, and *TravsDisturbed(v, t)* examines if the traversals with disturbance region intersecting t are disturbed by v. Procedures *TriDisturbed* and *TravsDisturbed* are illustrated in Figure 8. When a disturbed traversal is found by these procedures, the traversal is refined and both the new refinement vertex v_{ref} and v are inserted in V (if not already in V). Then, the algorithm continues testing all vertices in V. The overall algorithm is based on evaluating disturbed traversals nearby vertices since vertices remain unchanged during refinement operations, while the edge connectivity can be considerably re-arranged.

Algorithm 1 Local Refinement

LocalRef (V)

```
1.  while ( V not empty ) do
2.    v ← remove one element of V;
3.    for all ( triangles t adjacent to v ) do
4.      if ( TriDisturbed(t) or TravsDisturbed(v, t) ) then
5.        v_ref ← refine disturbed traversal;
6.        insert v in V;        // continue around v
7.        insert v_ref in V;    // propagate around v_ref
8.        break;                // break inner loop
9.      end if
10.   end for
11. end while
```

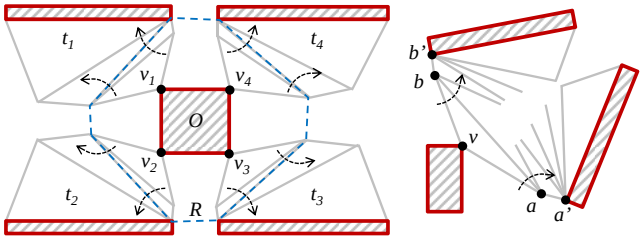

Fig. 8. Left: region R delimits all triangles tested by *TriDisturbed* after insertion of $\mathcal{O}$ in the underlying CDT. Traversals outside of R are tested by *TravsDisturbed*, which will select triangles t_1, t_2, t_3, and t_4 to be refined. Right: *TravsDisturbed(v, △vab)* will test all traversals behind ab for which v may be a disturbance.

Consider the example given in Figure 8. In Figure 8-left obstacle $\mathcal{O}$ has been inserted in the underlying CDT, and region R delimits all triangles tested by *TriDisturbed*. Traversals outside of R may also be disturbed and they are tested by routine *TravsDisturbed* (Figure 8-right). In the example, triangles t_1, t_2, t_3 and t_4 will be detected for refinement since they have traversals disturbed by v_1, v_2, v_3 and v_4 respectively. Given vertex v and an adjacent triangle $\triangle vab$, *TravsDisturbed(v, △vab)* will test traversals around a in a clockwise fashion and then traversals around b in a counterclockwise fashion. Traversals are sequentially tested around vertices a and b only while v can be orthogonally projected on the interior edge of the traversal being tested. When v cannot be anymore orthogonally projected on the interior edge, a and b are switched to their last visited neighbor vertices (a' and b' in Figure 8-right), and the process repeats until a switch cannot lead to a traversal that can have v orthogonally projected on its interior edge, or until a traversal that needs to be refined is found.

Algorithm 2 Search Disturbances Caused by a Constraint

SearchDisturbances (s, V)

```
1. {t1, t2} ← triangles sharing edge s;
2. Propagate ( s, V, t1, vertex of t1 that is not in s );
3. Propagate ( s, V, t2, vertex of t2 that is not in s );
```

Propagate (s, V, t, v)

```
1.  T ← traversals of t that have v as corner (there are two);
2.  for all ( traversals τ_abc in T ) do
3.    {S1, S2} ← sectors of τ_abc according to Figure 4-left;
4.    if ( s intersects sectors S1 or S2 ) then
5.      if ( τ_abc is disturbed ) then
6.        insert disturbance vertex in V (if not already in V);
7.      else
8.        t' ← the other triangle sharing edge bc with t;
9.        Propagate ( s, V, t', vertex of t' that is not in t );
10.     end if
11.   end if
12. end for
```

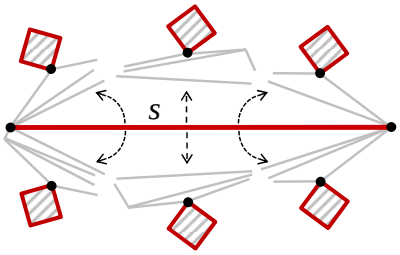

Fig. 9. Procedure *SearchDisturbances(s,V)* searches on both sides of s for possible disturbances caused by s. For each disturbed traversal found, the associated disturbance vertex v is added to list V for later processing.

5.2 Local Removal

In addition to identifying affected traversals that need to be refined, the removal operation has to take into account refinements that are no longer necessary after the removal. The overall procedure consists of four steps:

1. First, $\mathcal{O}$ is removed from the underlying CDT and all vertices adjacent to modified edges are stored in list V_1.
2. Then, all vertices that are not in V_1 but are neighbors to the refinement vertices in list V_1 are detected and stored in another list V_2.
3. All refinement vertices in V_1 are now removed from the underlying CDT since they may not be needed anymore.
4. Finally, all needed refinements are determined and performed by calling *LocalRef*($V_1 \cup V_2$).

Depending on specific situations, removal operations may be performed with only a subset of the steps above. Three modes of operation can be identified:

• Simple: only the CDT removal is performed. In this case the local clearance property is not restored and refinements that are

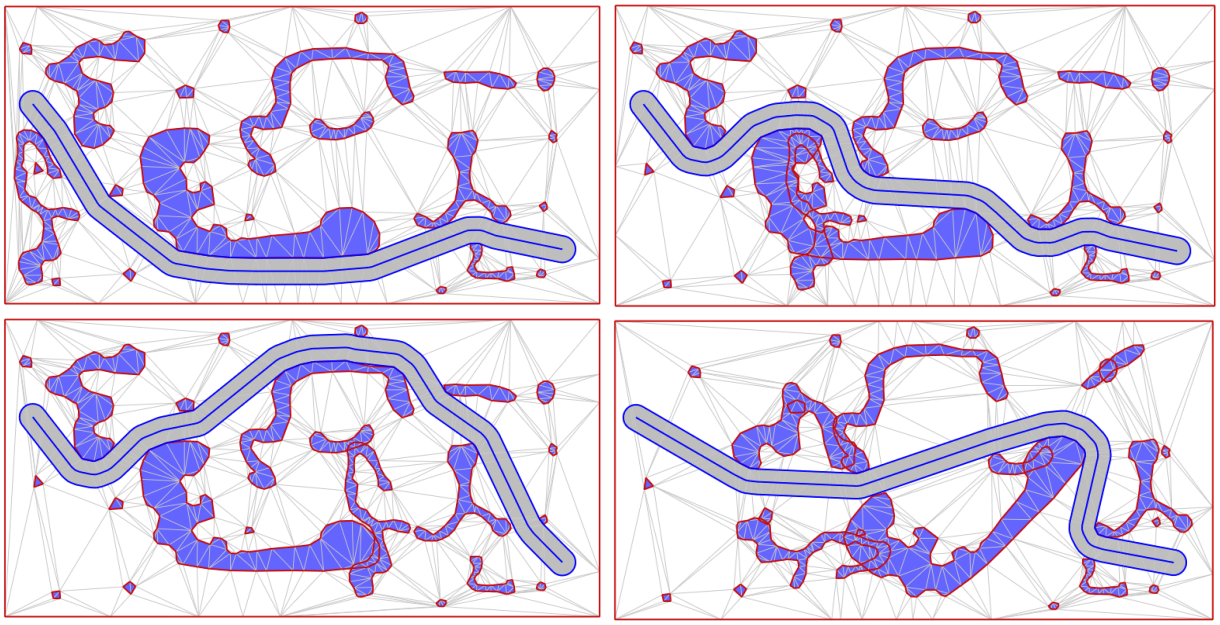

Fig. 10. Dynamic updates are handled with local operations and are robust when obstacles intersect. The diagrams show the path on the top-left environment being adapted as obstacles move. The first three diagrams, in left-right, top-down order, show the left-most obstacle being translated to the right. The fourth diagram has several obstacles displaced to random locations. Despite the several intersections, the resulting decompositions are always valid LCTs.

no longer needed are not removed (steps 2, 3 and 4 above are not performed). This is the fastest option but will require global LCT refinements in a later stage and refinement vertices may accumulate in case several sequential insertions and removals are performed.

• Adjacent: refinement vertices are evaluated for removal but new LCT refinements are not evaluated (step 4 not performed). This option prevents accumulation of refinements, but does not maintain the local clearance property.

• Full: in this mode the complete removal operation is performed.

5.3 Customization and Analysis

In addition to removal modes, the overall behavior of LCT updates can also be customized. Three useful modes can be defined:

• Global: only CDT operations are performed during both dynamic insertions and removals, and the global LCT refinement algorithm is automatically executed when the first path query with clearance is requested. This mode will be most suited for cases where few paths are computed but many dynamic changes occur everywhere.

• Local: in this mode, complete local refinements are performed at every polygon insertion and removal. This mode will be most suited for environments with relatively few dynamic updates but many path queries.

• Auto: this mode starts behaving as the global mode, and switches to local mode after the first global refinement is performed. This mode considers the typical case in most applications: first, all obstacles are inserted with CDT operations only, and a global refinement pass will be performed only when needed for the first path query. After this point the LCT is left in local mode.

The above selection of modes illustrates several possibilities for customization. The best mode will depend on how large is the LCT and how often dynamic updates are made. Mode *auto-full* will be the best option when only a few dynamic LCT updates are made. If more LCT updates than path queries are made, mode *global-simple* or *global-adjacent* may be more efficient, with only the latter preventing over accumulation of refinements. Another alternative to be considered is to perform a global removal of unnecessary refinements after a number of updates.

The described algorithms can also be optimized by reducing some of the redundancies in the performed tests; however, not all optimizations will lead to noticeable improvements. For example, although several of the tests performed by *SearchDisturbances* will be later repeated in $LocalRef$, minimizing this redundancy will not lead to noticeable speed gains because in practice *SearchDisturbances* only adds disturbances to V in very few situations involving long constraints.

It is important to observe that local updates are only beneficial if a relatively small portion of the environment is affected. When an obstacle is inserted or removed the dominant procedure is *LocalRef*, which will take $O(n_v^2 n_r)$ time to process an operation that affects n_v vertices, and where n_r is the maximum number of triangles processed when a new refinement is inserted (line 5 of the algorithm). Procedure *SearchDisturbances* will take $O(n_t n_c)$, where n_t is the total number of traversals visited, and n_c is the average number of edges visited when checking if a traversal is disturbed (line 6 of *Propagate*). Section 8 presents experiments quantifying the cost of local updates in several scenarios.

6. ROBUSTNESS

Robustness of geometric algorithms is a well studied problem in computational geometry. CDTs can be robustly computed with the use of two exact geometric predicates: the *ccw* test, for testing if three points are in counter-clockwise order, and the *in-circle* test, for testing if a point is inside the circle passing by three other given points [Shewchuk 1996; Devillers and Pion 2003].

However, exact geometric predicates only guarantee robustness in the combinatorial logic of algorithms and are not enough for achieving robust refinements and intersections of constraints. The problem is that points lying along a segment may not have exact representation in floating point coordinates, even when the segment endpoints have. Intersection or subdivision points computed with floating point arithmetic will be approximations with no guarantees of always being at acceptable locations.

The exact solution for this robustness problem would be to rely on arbitrary precision number representation, however requiring arbitrary amounts of space to represent numbers and slowing down computations significantly. Since the applications targeted by this work favor speed over accuracy, a solution based only on floating point representation has been developed.

In order to robustly determine the location of inserted points it is essential to rely on an exact *ccw* primitive. This work relies on a portable algorithm that progressively decomposes the *ccw* test into sums of double precision terms until the exact answer is found [Gavrilova et al. 2000]. Filtering techniques are also integrated [Devillers and Pion 2003] for improved efficiency. The exact *in-circle* test is not included since the approximation obtained with its floating point version can be used without posing robustness issues.

6.1 Robust Intersections and Refinements

Let p^{ex} be an exact intersection or refinement point on constraint s and let p be its approximation represented in floating point coordinates. In most cases $p \neq p^{ex}$ and p will not exactly lie on s, but p will still be an acceptable approximation for subdividing s in two sub-segments that are almost collinear. If p is exactly determined (using primitive *ccw*) to be on s or exactly inside one of the triangles adjacent to s, then p can be safely used as a subdivision point. However if not, then another edge exists between s and p and the refinement routine cannot subdivide s at p.

The first step to reduce the number of such robustness problems is to include a mechanism for merging points that are too close to each other. This is also useful for cleaning overly sampled obstacle contours, for removing gaps that should not exist between constraints, etc. Given a user-defined ϵ, two points are ϵ-close if the distance between them is less than or equal to ϵ. The incremental LCT triangulator will not insert a new vertex that is ϵ-close to an existing vertex in the LCT, it will instead re-use the existing vertex. Parameter ϵ controls the resolution to clean the input on-line, automatically merging points that are too close to each other.

If an unfeasible refinement of a constraint s at p is still detected, a legal refinement point p^{ref} is searched by evaluating new points along s. A good strategy is to evaluate new points following a binary partition pattern of s. The goal is to subdivide s in order to eliminate the need for the current infeasible refinement. Usually only a few iterations are needed until a feasible p^{ref} is found and in most of the cases the first iteration (using the midpoint of s) will already be successful and at the same time re-arrange the disturbed traversal. This strategy has showed to be more efficient than focusing the search nearby the location of the problematic original refinement. In the event that no refinement is found after a few iterations, then the LCT refinement is considered unfeasible and is not tried again. Such a case is usually not encountered in practice but may happen if s is overly short or the refinement region is overly dense. A non-performed refinement in such cases would only lead to insignificant variations in the clearance values computed for the affected area of the LCT.

Now consider the case of an unfeasible subdivision of constraint s at point p, where p is the intersection point between s and a new constraint s' being inserted on-line in the LCT. If p cannot subdivide s, then a search for a feasible point p^{ref} is also needed. Here the search focuses on points nearby p, searching from p towards the endpoints of s by increments. A good strategy for the increment is to start with $\epsilon/2$ and gradually increase it as the iterations progress. Usually only a few iterations are needed until p^{ref} is found nearby p. A feasible point has to be determined, and in the limit one of the endpoints of s will be used as p^{ref}. To minimize the "deformation" of s', two new points along s' are also inserted, one before s (p^{bef}) and another after it (p^{aft}). Points p^{bef} and p^{aft} are also robustly inserted with incremental search if needed. Figure 11 shows several examples of inserted points.

In Figure 11, the new constraint s' being inserted is shown as a dashed horizontal segment in the left-most diagrams. Each intersection point p between s' and the existing constraints is only inserted at feasible locations that respect the ϵ separation between vertices. If p lies in an invalid location, the search for a new location is performed. Such perturbations lead to small deformations in s', however these deformations are small scale and only at highly dense regions. The second row in Figure 11 illustrates the example where several constraints emanate from a single vertex v and s' intersects all of them. If s' gets arbitrarily close to v, at some point v will be used as intersection point, eliminating the need to compute intersections that can be arbitrarily close to each other.

6.2 Convergence of Multiple Intersections

Let s' be a constraint being inserted with end points at vertices a and b, and consider a situation similar to the one illustrated in the second row of Figure 11. The multiple intersections are se-

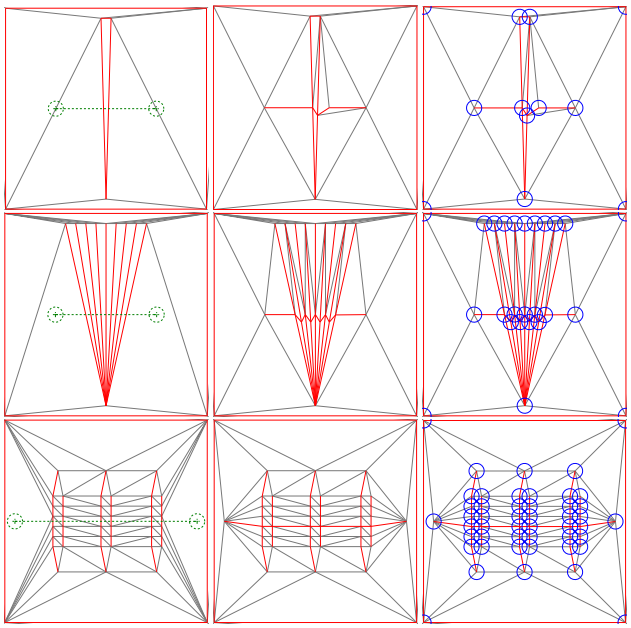

Fig. 11. Example situations with an exaggerated ϵ value. Left column: the dashed line is the new constraint s' to be inserted. Center column: result of the insertion. Right column: all vertices preserve ϵ distance from each other. Parameter ϵ controls the resolution to automatically merge points, cleaning the input data and reducing the number of cases requiring adjustment of coordinates in order to guarantee robustness.

quentially processed, starting from a and until b is reached. Although such an insertion procedure seems straightforward, intersection points may have arbitrary locations around their exact coordinates, possibly leading to cases where they are not directly connected by an edge to each other, and cases where they are not converging towards b. Such problems happen in practice and have to be handled robustly.

Let s_i be one constraint intersecting s' at p_i, and let p_{i-1} be the previous intersection point inserted. If p_i is the first intersection ($i = 1$), let p_{i-1} be a. If p_i is exactly determined to be on s_i or exactly inside one of the triangles adjacent to s_i, then p_i can be safely used. If not, then a search for a suitable robust insertion point p_i^{ref} along s_i is performed. However, the result may lead to a point p_i^{ref} that is not connected to p_{i-1} by an edge, and p_i^{ref} may also happen to not get closer to b than p_{i-1}. The strategy of inserting p_i^{bef} and p_i^{aft} along s' is also employed here and will ensure that the inserted points will converge towards b. Convergence can be guaranteed because if the insertion of p_i^{aft} also requires a robustness search, than the search along s' will only be in the direction of b. Therefore p_i^{aft} will always be closer to b than p_i. Point p_i^{bef} can be similarly guaranteed to always be closer to a than p_i, so that p_i^{bef} and p_i^{aft} are always consistent.

Still, p_i^{bef}, p_i^{ref} and p_i^{aft} may not have direct edge connections after they are inserted, and a connected sequence of edges joining the three vertices has to be discovered and marked as constrained. Every time a search for finding point p_i^{ref} is performed, the shortest path in the edge graph of the LCT between p_i^{bef} and p_i^{aft} is taken as the current portion of s' being inserted, and then the insertion of s' continues from p_i^{aft} towards b.

This overall process ensures that the multiple imprecise intersections will converge to b for all possible configurations. It also ensures a successful watertight insertion of constraints.

6.3 Overlapping Constraints

Every time a constraint is added, it will store the id of the obstacle that is using it. Since constraints can be shared by several obstacles, each constraint maintains a list of ids. When a new constraint is added between two vertices that are already connected by another constraint, no connectivity update is necessary and the id of the respective new obstacle is simply added to the list of ids stored in the constraint. In this way, obstacle edges that overlap are represented as a single constraint storing the ids of all the obstacles sharing it. If a polygonal obstacle is inserted multiple times at a same location, multiple ids will be generated and stored but only one set of vertices and constraints will exist to represent the obstacle in the triangulation. As an example, the environment in Figure 7 was built from a dataset of country boundaries and the boundaries between adjacent countries overlap.

When an obstacle is removed, the first step is to remove its id from all the constrained edges representing the obstacle. Then, only the constraints whose id lists have become empty are removed.

6.4 Importance and Analysis

The need for handling intersections robustly and dynamically is important in many cases. For example, when nearby static agents are represented as simplified polygons around them and inserted as obstacles in a LCT, the generated constraints will often intersect each other. Another situation is to simplify the design process of users and designers, allowing them to design spaces with long intersecting edges. Ensuring that inserted closed obstacles remain watertight is important for flood fill algorithms that may associate traversal costs to regions delimiting different terrains (grass, sidewalks, etc). All such cases are robustly handled on-line with the described procedures.

Figure 10 illustrates the typical case of removing and inserting obstacles to new positions. Figure 14 shows many intersections handled in a game floor plan design and test. Edges of obstacles can overlap or intersect with other obstacles and all cases are robustly handled.

The described search procedures for inserting refinements and intersections are only triggered in special circumstances that do not occur often; but when they are needed, they achieve a robust result. Section 8 discusses experiments indicating that the proposed solutions are very efficient in practice. In addition, the approach accommodates new constraints to an already existing LCT without changing vertices previously inserted, guaranteeing that static portions of an environment are not disturbed and remain static after several intersecting dynamic operations.

7. CHANNEL SEARCH

Once a LCT of the environment is available, a graph search can be performed over the adjacency graph of the triangulation in order to obtain a channel C_r connecting two input points p and q.

The process first locates the triangle t_{init} containing p using the *oriented walk* search method [Devillers et al. 2001]. The method extensively relies on *ccw* tests and to be most efficient the implemented solution starts with floating point *ccw* tests until a first triangle containing p is determined. Then the tests are switched to the exact *ccw* primitive and the oriented walk continues if needed. This hybrid approach significantly improves the time spent on point location and at the same time ensures correct and robust results. Another possible optimization is to include a hashing mechanism based on a grid overlaid on the environment for quickly determining a seed triangle already very close to the triangle containing p. However, such a global hashing would need to be updated whenever dynamic updates occur in the LCT.

During channel search, a search expansion is only accepted if the clearance of the traversal being expanded (which is precomputed in the free LCT edges) is greater or equal to $2r$. Theorem 5 (in Appendix A) shows that LCTs can be safely searched assuming that every cell will be traversed by a given path only once, allowing the search to mark visited triangles and to correctly terminate after visiting each triangle no more than once. Figure 12 shows that this is not always the case for all types of cell decompositions.

The channel search will return a valid channel C_r if one exists but there are no guarantees that C_r^* will be found. In this work an A* search is employed on an adjacency graph that has its edges oriented towards the goal when the goal is visible [Kallmann 2010]. A typical search tree that is generated is illustrated in

Figure 13. The segments in black represent the expanded edges, and the segments in blue represent the expansion front at the moment of reaching the goal. Extended searches that can find a global optimum are possible [Demyen and Buro 2006; Kallmann 2010] but will take exponential time in the worst case.

Once a channel containing the solution path is found, the shortest path in the channel can be efficiently computed with the *funnel algorithm* [Hershberger and Snoeyink 1994] by extending it to handle clearances [Demyen 2007; Kallmann 2010].

Table I. Global versus local refinements in removal operations.

Environment	v	t_{glob}	$t^{v/4}_{full}$	$t^{v/4}_{adj}$	$t^{v/4}_{sim}$	$t^{v/4}_{sim+g}$	$t^{v/4}_{adj+g}$	$t^{v/2}_{full}$	$t^{v/2}_{adj}$	$t^{v/2}_{sim}$	$t^{v/2}_{sim+g}$	$t^{v/2}_{adj+g}$
hybrid	1705	0.013	0.009	0.005	0.005	0.008	0.012	0.021	0.011	0.010	0.011	0.016
wmap	62158	1.314	0.223	0.179	0.168	0.314	0.557	0.490	0.400	0.373	0.811	0.848
obs1k	1229	0.009	0.010	0.003	0.002	0.003	0.009	0.017	0.006	0.004	0.006	0.008
obs5k	5599	0.055	0.043	0.011	0.008	0.025	0.046	0.079	0.027	0.020	0.039	0.046
obs15k	15044	0.192	0.120	0.028	0.022	0.070	0.091	0.219	0.070	0.056	0.108	0.122
obs23k	23504	0.213	0.411	0.050	0.036	0.104	0.148	0.603	0.120	0.092	0.108	0.180
obs30k	29287	0.297	0.253	0.059	0.045	0.108	0.249	0.451	0.146	0.117	0.159	0.314
obs40k	41506	0.423	0.365	0.087	0.065	0.201	0.357	0.654	0.204	0.174	0.235	0.385
obs54k	55090	0.516	0.529	0.117	0.084	0.249	0.558	0.940	0.277	0.219	0.294	0.572
obs135k	135267	1.361	1.355	0.271	0.217	0.511	1.442	2.300	0.673	0.599	0.798	1.460

Notation: Time t_{glob} is the time taken by the global refinement algorithm on the CDT of the input obstacles in order to compute a LCT with v vertices. Times t^k_{mode} are the times taken to remove a set of obstacles with approximately k vertices from the corresponding LCT, using different strategies according to *mode*: *full* for local-full removals, *adj* for local-adjacent removals, *sim* for local-simple removals, *sim+g* for local-simple removals followed by a global refinement pass, *adj+g* for local-adjacent removals followed by a global refinement pass. All times are in seconds.

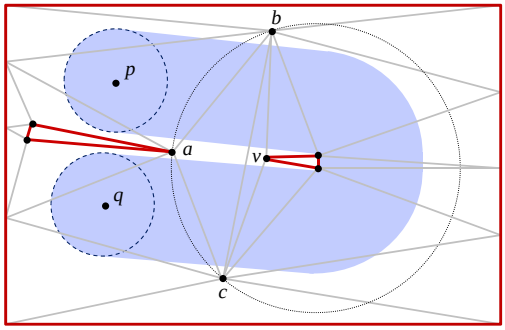

Fig. 12. The shown path $\pi^*_r(p, q)$ is the only solution with clearance r and in the given cell decomposition it traverses $\triangle abc$ and $\triangle bcv$ twice. Clearly, the given triangulation is not a LCT and not a CDT since the circumcircle of $\triangle abc$ has vertices in its interior.

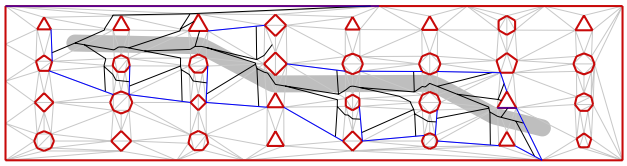

Fig. 13. Expansion tree of the used search metric.

8. RESULTS AND DISCUSSION

The proposed algorithms were implemented and several evaluations are presented in Tables I, II and III. The experiments were based on: environment *hybrid* shown in Figure 1, environment *wmap* shown in Figure 7, and environments *obsNk*, which are similar to the one shown in Figure 13 but with the corresponding set of obstacles totaling approximately NK input vertices. All tests were performed without the bounded clearance optimization and the time taken by the robust point location procedure is included in the reported path computation times. The results were obtained in an Intel Core i7-2600K 3.4GHz. No parallelization or GPU processing was used.

Table I presents computation times of removal operations in different combinations of modes and applied to sets of obstacles of different sizes. By comparing the t^k_{full} columns against the t^k_{adj+g} columns, it is possible to evaluate how far local maintenance of refinements is preferable to global passes. By comparing these two columns with respect to removals of $k = v/4$ size, half of the experiments were favorable to the local-full operations. With respect to removals of $k = v/2$ size, only one experiment was favorable. The exception was the *wmap* environment, which is the environment with the highest number of refinements overall (see column n of Table III). This shows that performances greatly depend on the environment type. More in general the overall times indicate that if more than 25% of the vertices are affected in a local removal, it may be beneficial to employ global refinement passes in order to maintain the LCT. Global passes will be even more attractive if they can be employed after simple removals (instead of after adjacent removals). Simple removals can be enough in several situations, however, they will not check for refinements that are no longer needed, and if applied to successively displace objects by small increments, accumulation of refinements may occur.

Table II. Global versus local refinements in insertion operations.

Environ.	$t^{v/4}_{loc}$	$t^{v/4}_{cdt}$	$t^{v/4}_{cdt+g}$	$t^{v/2}_{loc}$	$t^{v/2}_{cdt}$	$t^{v/2}_{cdt+g}$
hybrid	0.007	0.002	0.007	0.017	0.004	0.014
wmap	0.233	0.076	1.216	0.507	0.154	1.523
obs1k	0.006	0.001	0.006	0.012	0.003	0.008
obs5k	0.029	0.006	0.037	0.058	0.013	0.037
obs15k	0.088	0.019	0.173	0.171	0.043	0.172
obs23k	0.146	0.029	0.221	0.264	0.072	0.259
obs30k	0.174	0.042	0.215	0.359	0.098	0.398
obs40k	0.253	0.064	0.431	0.542	0.155	0.445
obs54k	0.341	0.092	0.536	0.763	0.223	0.726
obs135k	0.943	0.294	1.280	2.293	0.762	2.631

Notation: Times t^k_{mode} are the times taken to insert a set of obstacles with approximately k vertices in the LCT, using different strategies according to *mode*: *loc* for local insertions with local refinements, *cdt* for local CDT insertions without refinements, and *cdt+g* for local CDT insertions followed by a global refinement pass. All times are in seconds.

Table II presents performance times obtained with insertion operations. Local refinements can be evaluated by comparing the t^k_{loc} columns against the t^k_{cdt+g} columns. For insertions of $k = v/4$ size, eight experiments (out of ten) were favorable to local operations. With respect to insertions of $k = v/2$ size, four experiments were favorable. These times show that refinements are more efficiently maintained during insertions than removals, what can be explained by the simpler steps of the local insertion algorithm. Overall the numbers indicate that global refinement passes may be

Table III. LCT size, path computation time, and path optimality.

Environment	n	Refinements	k_{ref}	$\triangle$	$\triangle_{ref}$	$k_{\triangle}$	$\bar{t}$	$\bar{l}_{loc}$	$\bar{l}_{glob}$	$\bar{l}_{diff}$	σ_{diff}
hybrid	1655	50	0.0302	3405	100	2.0574	0.4948	198.8689	197.4160	0.5999	1.4200
wmap	60374	1784	0.0295	124311	3568	2.0590	2.6130	200.1519	198.9831	0.6036	1.5805
obs1k	1211	18	0.0149	2453	36	2.0256	0.2434	153.3504	153.0448	0.1784	0.4067
obs5k	5554	45	0.0081	11193	90	2.0153	0.4516	156.2630	155.9103	0.2188	0.3430
obs15k	14967	77	0.0051	30083	154	2.0100	0.7343	162.2595	161.9046	0.2019	0.2117
obs23k	23438	66	0.0028	47003	132	2.0054	1.0561	150.5734	150.3085	0.1866	0.3447
obs30k	29202	85	0.0029	58569	170	2.0057	1.1885	151.6780	151.3579	0.2107	0.2223
obs40k	41399	107	0.0026	83007	214	2.0050	1.6493	147.0370	146.7173	0.2230	0.2162
obs54k	54878	212	0.0039	110175	424	2.0076	1.7855	136.0318	135.8508	0.1299	0.1408
obs135k	135054	213	0.0016	270529	426	2.0031	5.9430	134.7292	134.4717	0.2160	0.2297

Notation: n is the number of vertices in the input environment, k_{ref} is the number of refinements with respect to n (refinements/n), $\triangle$ is the number of triangles in the obtained LCT, $\triangle_{ref}$ is the number of triangles added to the input CDT due refinements, $k_{\triangle}$ is the number of LCT triangles with respect to n ($\triangle/n$), $\bar{t}$ is the average time taken (in miliseconds) to compute locally shortest paths, $\bar{l}_{loc}$ is the average length of locally shortest paths, $\bar{l}_{glob}$ is the average length of globally shortest paths, $\bar{l}_{diff}$ is the average of how worse (in percentage) local solutions were from their respective optima (average of $l^i_{diff} = 100(l^i_{loc} - l^i_{glob})/l^i_{glob}$, where i represents the index of each individual query), and σ_{diff} is the standard deviation of the l^i_{diff} values. The average times and lengths were computed over 1000 successful path queries among random query points.

beneficial if more than 50% of the vertices are affected in a local insertion. However, environment *wmap* again shows that performances greatly depend on the type of the environment. Local refinements for insertions of $v/2$ size in *wmap* were three times faster than when employing global refinements. This can be explained by the high number of disturbances in this environment, which make the global refinement algorithm to require multiple passes to terminate. In contrast, the local algorithm performs incremental refinements after each individual obstacle is inserted.

Table III presents several statistics related to refinements, path computation time and path length. The search for locally optimal paths is highly efficient, with paths being computed in about 1.8 milliseconds in environments described by 54K segments. The number of refinements needed per environment is also shown to be relatively very small. As a consequence, the number of cells processed by the channel search algorithm was close to $2n$ (see $k_{\triangle}$ column), which is much lower than the theoretical bound of $6n$. It is also possible to observe that the lengths of the locally shortest paths were in average no worse than 0.61% of the globally optimal solutions. These results demonstrate that locally optimal paths are suitable for character navigation, and the small difference from global optima can be used as a way to mimic the humanlike behavior of not always using the same path, for example by varying the heuristics used by the channel search procedure.

The solutions proposed in Section 6 for the robust on-line processing of intersecting constraints have also shown to be efficient in practice. Tests were performed with random obstacle removals and insertions generating many intersections among polygons with several long and parallel edges. The environment is shown in Appendix D. In 1 million intersections processed, only 18 required searching for legal intersection points, and the maximum number of points evaluated in a search was 6. Without the search for robust insertions these cases would have lead to fatal errors in the test application.

Figure 14 shows one test environment for The Sims 4. The small squares represent the position of static characters so that paths for the active characters can account for them. Whenever characters walk, their respective enclosing squares are dynamically removed. The environment shows several intersections and the proposed robustness techniques were essential to always guarantee successful triangulations.

Many extensions can be developed for customization of the proposed solutions to specific needs. One effective technique to control

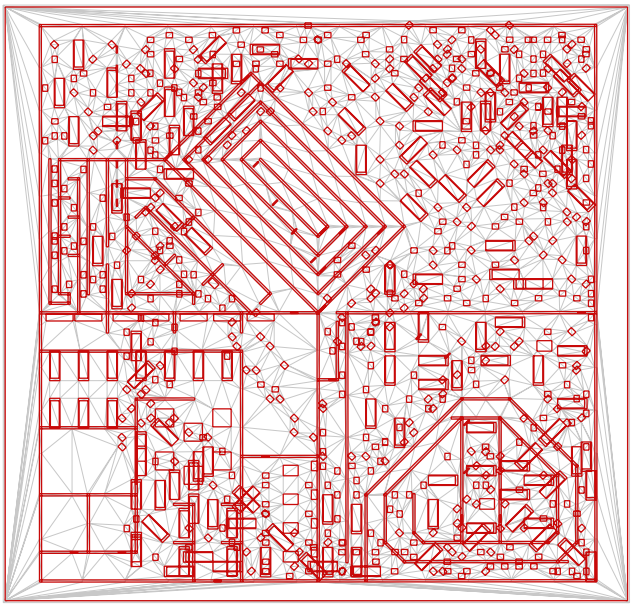

Fig. 14. Test environment used during development of The Sims 4. The small squares represent static characters. Original data produced by Electronic Arts. Used with permission.

arrival orientation is to dynamically insert point-obstacles nearby the target location in order to create a unique feasible path arrival direction. The triangulation can also be used for visibility and proximity queries in different ways. Figure 15 illustrates a dynamic CDT being used for tracking agent-agent and agent-obstacle proximity while multiple agents are following their LCT paths. These and additional examples are presented in the movie accompanying this paper. The movie demonstrates refinements being maintained while obstacles move, paths adapting to dynamic changes, and several agent navigation examples based on LCTs.

Figure 16 presents a comparison against the medial axis representation. In this example, the edge chains of the full medial axis graph in the main corridors of the environment have 12, 14, 12, 7, 40 and 11 nodes. The corresponding corridors in the LCT of the same environment are represented with 9, 11, 8, 4, 30 and 10 adja-

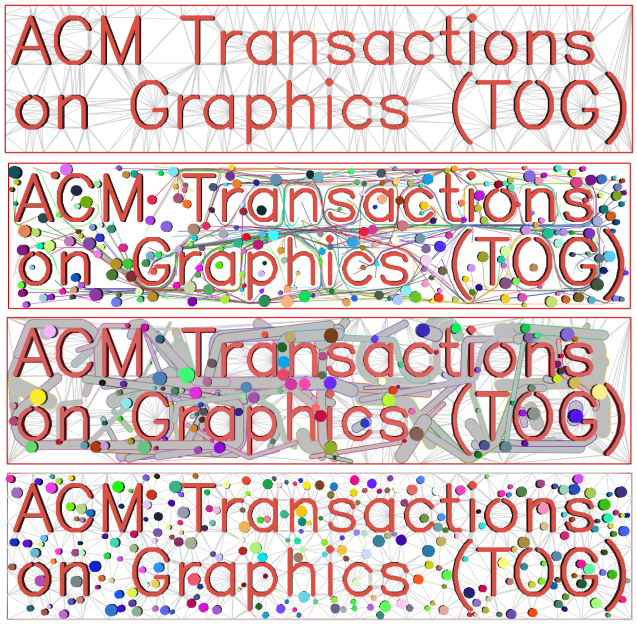

Fig. 15. Multi-agent path following simulation example. A LCT (top) is employed for planning paths to be followed by each agent (middle snapshots), while a dynamic CDT is used to track proximity for fast collision avoidance determination (bottom). The two snapshots in the middle show different path visualizations and their paths were kept relatively short in order to improve clarity.

cent triangles, what represents 75% of the number of nodes used by the full medial axis. While the full medial axis needs a larger number of nodes to represent all pairs of closest features, both structures are $O(n)$ in size and have a practically equal number of nodes with degree 3. Both structures will therefore achieve similar path query times when only considering degree 3 nodes.

If speed, storage space, and dynamic updates are the main criteria, LCTs provide unmatched efficiency and flexibility. It is possible to devise decompositions with larger cells or to add coarser hierarchical layers in the adjacency graph in order to achieve faster path searches, however in such cases it becomes difficult to address efficient dynamic updates and arbitrary clearance at the same time. The overall approach is also highly flexible given that the underlying CDT operations are useful for solving a number of geometric problems, as for example for tracking proximity (Figure 15-bottom).

All provided examples were computed geometrically without the use of any additional structures. The presented methods for addressing robustness are important because they let users safely edit and design their own environments without restrictions, a desirable feature in computer games. The independence of a GPU allows the methods to be highly portable and broadly usable, as for example in game servers without GPUs.

Although LCTs are well suited to multi-agent navigation, reactive behaviors for avoiding bottlenecks and collisions with other agents during path following are still needed. A number of approaches have been proposed in the crowd animation area [Shao and Terzopoulos 2005; Singh et al. 2009; Berg et al. 2008]. The examples given in Figure 15 only include the simple behavior of stopping and computing a new random path every time an agent collides with another agent during path following.

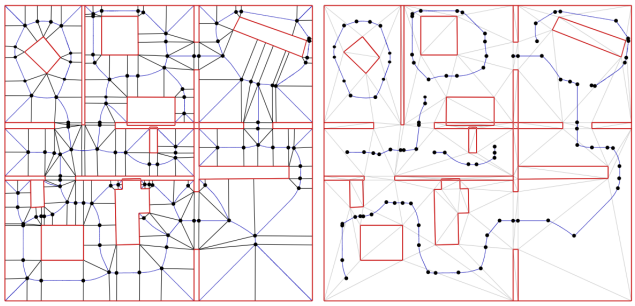

Fig. 16. The left image shows the full medial axis graph of an example environment. The black segments connect to the closest features and determine the nodes of the graph. The right image shows the medial axis of main corridors on top of the LCT of the same environment. The LCT adjacency graph represents the corridors with 72 nodes, while the medial axis uses 96 nodes.

9. CONCLUSION

This paper demonstrates the theoretical properties of local clearance triangulations and presents new algorithms for handling dynamic updates and robustness. The presented methods introduce a new meshing-based methodology for representing environments for navigation, and lead to a powerful representation with clearance information. The presented solution represents a highly flexible and efficient approach for extracting paths with clearance from polygonal environments.

ACKNOWLEDGMENTS

The author is grateful to Lee Wilson for fruitful discussions and for the test data used in Figure 14.

REFERENCES

BERG, J., LIN, M., AND MANOCHA, D. 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*. IEEE.

BHATTACHARYA, P. AND GAVRILOVA, M. 2008. Roadmap-based path planning - using the voronoi diagram for a clearance-based shortest path. *Robotics Automation Magazine, IEEE 15,* 2 (june), 58 –66.

CHAZELLE, B. 1982. A theorem on polygon cutting with applications. In *SFCS '82: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 339–349.

CHEW, L. P. 1985. Planning the shortest path for a disc in $o(n^2 logn)$ time. In *Proceedings of the ACM Symposium on Computational Geometry*.

DE BERG, M., CHEONG, O., AND VAN KREVELD, M. 2008. *Computational geometry: algorithms and applications*. Springer.

DEMYEN, D. 2007. Efficient triangulation-based pathfinding. MSc Thesis.

DEMYEN, D. AND BURO, M. 2006. Efficient triangulation-based pathfinding. In *AAAI'06: Proceedings of the 21st national conference on Artificial intelligence*. AAAI Press, 942–947.

DEVILLERS, O. AND PION, S. 2003. Efficient exact geometric predicates for delaunay triangulations. In *Proceedings of the 5th Workshop Algorithm Engineering and Experiments*. 37–44.

DEVILLERS, O., PION, S., AND TEILLAUD, M. 2001. Walking in a triangulation. In *ACM Symposium on Computational Geometry*. 106–114.

GAVRILOVA, M. L., RATSCHEK, H., AND ROKNE, J. G. 2000. Exact computation of delaunay and power triangulations. *Reliable Computing 6,* 1, 39–60.

GAYLE, R., SUD, A., ANDERSEN, E., GUY, S. J., LIN, M. C., AND MANOCHA, D. 2009. Interactive navigation of heterogeneous agents using adaptive roadmaps. *IEEE Transactions on Visualization and Computer Graphics 15*, 34–48.

GERAERTS, R. 2010. Planning short paths with clearance using explicit corridors. In *ICRA'10: Proceedings of the IEEE International Conference on Robotics and Automation*.

HART, P. E., NILSSON, N. J., AND RAPHAEL, B. 2007. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on 4,* 2 (February), 100–107.

HELD, M. 2001. Vroni: An engineering approach to the reliable and efficient computation of voronoi diagrams of points and line segments. *Computational Geometry 18,* 2, 95–123.

HERSHBERGER, J. AND SNOEYINK, J. 1994. Computing minimum length paths of a given homotopy class. *Computational Geometry Theory and Application 4,* 2, 63–97.

HERSHBERGER, J. AND SURI, S. 1997. An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing 28*, 2215–2256.

HOFF, III, K. E., CULVER, T., KEYSER, J., LIN, M., AND MANOCHA, D. 2000. Fast computation of generalized voronoi diagrams using graphics hardware. In *ACM Symposium on Computational Geometry*.

JORGENSEN, C.-J. AND LAMARCHE, F. 2011. From geometry to spatial reasoning: automatic structuring of 3d virtual environments. In *Proceedings of the 4th international conference on Motion in Games (MIG)*. Springer-Verlag, Berlin, Heidelberg, 353–364.

KALLMANN, M. 2010. Shortest paths with arbitrary clearance from navigation meshes. In *Proceedings of the Eurographics / SIGGRAPH Symposium on Computer Animation (SCA)*.

KALLMANN, M., BIERI, H., AND THALMANN, D. 2003. Fully dynamic constrained delaunay triangulations. In *Geometric Modeling for Scientific Visualization*, G. Brunnett, B. Hamann, H. Mueller, and L. Linsen, Eds. Springer-Verlag, Heidelberg, Germany, 241–257. ISBN 3-540-40116-4.

KAPOOR, S., MAHESHWARI, S. N., AND MITCHELL, J. S. B. 1997. An efficient algorithm for euclidean shortest paths among polygonal obstacles in the plane. In *Discrete and Computational Geometry*. Vol. 18. 377–383.

KOENIG, S. AND LIKHACHEV, M. 2002. D* lite. In *Proceedings of AAAI Conference on Artificial Intelligence*. 476–483.

KUFFNER, J. J. AND LATOMBE, J.-C. 1999. Fast synthetic vision, memory, and learning models for virtual humans. In *CA'99: Proceedings of Computer Animation*. IEEE.

LAMARCHE, F. 2009. Topoplan: a topological path planner for real time human navigation under floor and ceiling constraints. *Computer Graphics Forum 28,* 2, 649–658.

LAMARCHE, F. AND DONIKIAN, S. 2004. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum 23,* 3, 509–518.

LEE, D. T. AND PREPARATA, F. P. 1984. Euclidean shortest paths in the presence of rectilinear barriers. *Networks 3,* 14, 393–410.

LIKHACHEV, M., GORDON, G., AND THRUN, S. 2003. ARA*: Anytime A* search with provable bounds on sub-optimality. In *Neural Information Processing Systems (NIPS)*. MIT Press.

LIU, Y. H. AND ARIMOTO, S. 1995. Finding the shortest path of a disk among polygonal obstacles using a radius-independent graph. *IEEE Transactions on Robotics and Automation 11,* 5, 682–691.

LOZANO-PÉREZ, T. AND WESLEY, M. A. 1979. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of ACM 22,* 10, 560–570.

METOYER, R. A. AND HODGINS, J. K. 2003. Reactive pedestrian path following from examples. In *CASA '03: Proceedings of the 16th International Conference on Computer Animation and Social Agents*. IEEE Computer Society.

MITCHELL, J. S. B. 1993. Shortest paths among obstacles in the plane. In *Proceedings of the ninth annual symposium on computational geometry (SoCG)*. ACM, New York, NY, USA, 308–317.

NILSSON, N. 1969. A mobile automaton: an application of artificial intelligence techniques. In *In Proceedings of the 1969 International Joint Conference on Artificial Intelligence (IJCAI)*. 509–520.

NOSER, H. AND THALMANN, D. 1995. Synthetic vision and audition for digital actors. In *Proc. of Eurographics*.

OLIVA, R. AND PELECHANO, N. 2013. Neogen: Near optimal generator of navigation meshes for 3d multi-layered environments. *Computer & Graphics 37,* 5, 403–412.

OVERMARS, M. H. AND WELZL, E. 1988. New methods for computing visibility graphs. In *Proceedings of the fourth annual symposium on Computational geometry (SoCG)*. ACM, 164–171.

SHAO, W. AND TERZOPOULOS, D. 2005. Autonomous pedestrians. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. SCA '05. ACM, New York, NY, USA, 19–28.

SHEWCHUK, J. R. 1996. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, M. C. Lin and D. Manocha, Eds. Lecture Notes in Computer Science, vol. 1148. Springer-Verlag, 203–222. From the First ACM Workshop on Applied Computational Geometry.

SHEWCHUK, J. R. 1997. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry 18,* 3 (Oct.), 305–363.

SINGH, S., KAPADIA, M., FALOUTSOS, P., AND REINMAN, G. 2009. SteerBench: a benchmark suite for evaluating steering behaviors. *Computer Animation and Virtual Worlds 20,* 56 (september), 533–548.

STORER, J. A. AND REIF, J. H. 1994. Shortest paths in the plane with polygonal obstacles. *J. ACM 41,* 5, 982–1012.

SUD, A., ANDERSEN, E., CURTIS, S., LIN, M. C., AND MANOCHA, D. 2008. Real-time path planning in dynamic virtual environments using multiagent navigation graphs. *IEEE Transactions on Visualization and Computer Graphics 14*, 526–538.

WEIN, R., VAN DEN BERG, J., AND HALPERIN, D. 2007. The visibility-voronoi complex and its applications. *Computational Geometry: Theory and Applications 36,* 1, 66–78.

Received September 2008; accepted March 2009

APPENDIX

A. THEOREMS

The main properties discussed in the paper are addressed by the theorems below. The term *first disturbed* is used to specify that v is the first disturbance in case of multiple disturbances (see Section 4.1). The referenced lemmas are given in Appendix B.

THEOREM 1. (INTERNAL ANGLE.) *Let traversal $\tau_{abc} \in T = LCT(\mathcal{S})$ be first disturbed by vertex v with respect to constraint s, and let v' be the orthogonal projection of v on s. Let d and e be the vertices connected to v forming $\triangle dve \in T$ that is crossed by segment vv' (see Figure 3). In this situation, the internal angle of the disturbance $\angle dve > \pi/2$.*

PROOF. First consider the simpler case where $d = b$ and $e = c$. In this situation, by Lemma 1, $\exists o$ such that $\angle boc = \pi/2$ and v lies inside $\triangle boc \Rightarrow \angle bvc > \pi/2$. The same reasoning is now applied to generic vertices d and e connected to v. Let u be the line parallel to s passing by v and let R be the region of all possible disturbances closer to s than v and on the same side of vv' as b. Region R forms a right angle α at vertex v, which is the angle between u and segment vv' (see Figure 17-left). By definition region R is free of disturbances, and since by Lemma 3 any vertex inside R would be a disturbance, it then follows that no vertices can appear inside R. Therefore d has to be below u and e right of $vv' \Rightarrow \angle dve > \alpha = \pi/2$. □

THEOREM 2. (REFINEMENT.) *Let traversal τ_{abc} be first disturbed by vertex v with respect to constraint s, and let v' be the orthogonal projection of v on s. The new CDT obtained with the refinement of s with p_{ref} (see Figure 5) will lead to a new vertex v_{ref} connected by edges to v and to all the vertices of the triangles crossing vv'.*

PROOF. Let d and e be the vertices connected to v forming $\triangle dve$ crossed by segment vv', and let u be the line parallel to s passing by v. Using Lemma 3 and the same reasoning as in Theorem 1, d is below u, and e right of vv' (see Figure 17-left). Since v is a disturbance, then $dist(v, e) > dist(v, v')$ and thus e has to be outside the circle centered at v and with radius vv' (see Figure 17-right). Let $\mathcal{C}$ be the circumcircle of $\triangle dve$. Since $\angle dve > \pi/2$, then the center of $\mathcal{C}$ has to lie on the opposite side of de with respect to v. Figure 17-right illustrates that for any valid positions of d and e, $\mathcal{C}$ will have to intersect s and entirely contain segment vv'. Since $\mathcal{C}$ intersects s, then p_{ref} will be inside $\mathcal{C}$. Therefore, in order to maintain the empty circle criterion, when p_{ref} becomes a new CDT vertex v_{ref}, $\triangle dve$ will no longer exist and v_{ref} will be connected to v. The originally disturbed $\triangle abc$ becomes invalid when v and v_{ref} are connected by an edge and thus b will also become connected to v_{ref}. The same applies to all vertices that originally had an edge crossing vv'. □

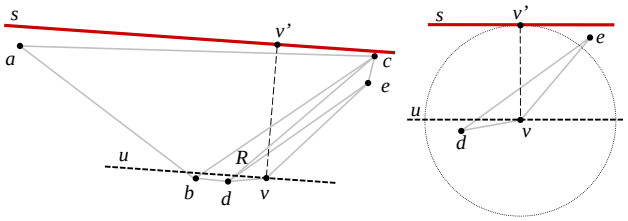

Fig. 17. Illustrations used by theorems 1 and 2.

THEOREM 3. (LINEAR REFINEMENTS.) *The number of refinements for obtaining $LCT(\mathcal{S})$ with the iterative refinement algorithm is at most $3n$, where n is the number of input points.*

PROOF. Without loss of generality it can be assumed that if a traversal τ_{abc} has multiple disturbances with respect to constraint s, the disturbance that is processed first is the first disturbance vertex v. If other disturbances remain after the refinement with respect to v is performed, they will be processed in subsequent passes.

For vertex v to disturb traversal τ_{abc}, by Theorem 1, the internal angle of the disturbance has to be larger than $\pi/2$. Therefore, at most 3 incident triangles to v can have internal angle at v greater than $\pi/2$, and so v can disturb a maximum of three different traversals. As the starting triangulation $T_0 = CDT(S)$ has exactly n vertices, then the maximum possible number of refinement vertices inserted in T_0 is $3n$ (see Figure 21 in the Appendix for an example). By Theorem 2 each disturbance v with respect to τ_{abc} will lead to v and b connected to p_{ref} after s is refined. Therefore vertices v and b will become corners of new traversals with entrance and exit edges that are directly connected to s and that have clearance values $dist(b, s)$ and $dist(v, s)$. Since s was the closest constraint to the original τ_{abc} disturbed by v, after refinement, v may not disturb any other traversal between b and s. Therefore, if originally v was disturbing $k \leq 3$ traversals, after refinement, it will disturb only up to $k - 1$ traversals. The total number of refinements after several passes therefore remains limited to $3n$.

The final aspect to be considered is that refinement vertices are always inserted subdividing a constraint in two new collinear constraints and so they cannot become new disturbances by definition. Refinement vertices may re-arrange triangles that can become disturbed by original vertices, however, the original vertices are still limited to $3n$ disturbances. Therefore the refinements cannot indefinitely propagate and the maximum number of refinements remains bounded by $3n$. □

THEOREM 4. (LCT CORRECTNESS.) *Let C be a channel of $T = LCT(\mathcal{S})$ containing a path π such that for each traversal τ_{abc} in C, $cl(a, b, c) > 2r$. Then, π_r exists in C.*

PROOF. Let $diam(p)$ denote the maximum diameter of the disc centered at a point p such that no vertices or constraints of T lie in its interior. The disc is denoted D_p and only its boundary will intersect with vertices or constraints of T. The intersections are called the contact points of D_p.

Let p be the point of the medial axis inside channel C such that $diam(p)$ is minimum and that p lies in a triangle fully traversed by π (cases related to the path endpoints are treated with specific arrival and departure tests). Point p represents the location of the narrowest passage in C. If there is enough clearance at the narrowest point of C, then there is enough clearance everywhere in C and a continuous deformation from the medial axis in C to π_r exists. Therefore it is enough to show that $diam(p) \geq 2r$. Consider all pairs formed by two contact points of D_p that are on opposite sides of π. For each pair, three cases may exist: 1) both points are vertices of T, 2) both points lie on constraints of T, or 3) one point is a vertex and another lies on a constraint (see Figure 18).

In the first case, let v_1 and v_2 be the two contact points. Since $diam(p)$ is minimum, $dist(v_1, v_2) = diam(p)$ and no other vertices or constraints can be inside D_p. Therefore an empty circle passes by v_1 and v_2 and by the Delaunay criterion the two vertices have to be connected by an edge e of T. Edge e will be an exit and entrance edge of two adjacent traversals with local clearance at most $dist(v_1, v_2) = diam(p)$. Since T has the local clearance property, then $diam(p) \geq 2r$ and π_r will safely pass by the narrow passage.

In the second case, the two contact points lie in two constraints s_1 and s_2. Since $diam(p)$ is minimum, s_1 and s_2 have to be parallel otherwise a lower value for $diam(p)$ would be obtained with respect to an endpoint of s_1 or s_2. Since the two constraints are parallel, the same value of $diam(p)$ will be obtained with respect to an endpoint, reducing this case to case 1 or 3.

In the third case, let v be the contact vertex and s be the constraint containing the second contact point v'. Since $diam(p)$ is minimum, v' has to be the orthogonal projection of v on the interior of s since otherwise the second contact point would be an endpoint of s and this would be a case 1. If segment vv' is not crossed by any edge of T then v will form a triangle with the endpoints of s, and the triangle will be crossed by π with a traversal where v is the corner and the clearance is $d(v, s)$. Since all clearances are greater or equal to $2r$, then π_r will be a free path with respect to v and s. If segment vv' is crossed by edges of T, other traversals will be crossed instead. For each crossed traversal τ_{abc}, $dist(v, s) \geq dist(b, s)$ since T is a LCT free of disturbances (see Definition 3). Therefore since all clearances are greater than $2r$, $cl(a, b, c) > 2r$ and π_r will be a free path with respect to v and s. □

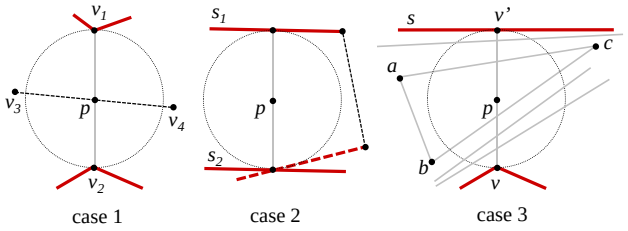

Fig. 18. Cases considered by Theorem 4.

THEOREM 5. (SINGLE TRAVERSABILITY.) *For every path π_r found in a LCT, each triangle in its channel C_r will only be traversed once.*

PROOF. Suppose that $t \in C_r$ is traversed twice and let the two passable traversals of t be τ_{abc} and τ_{bca} (see Figure 12). Since τ_{abc} is passable, then: $cl(a, b, c) \geq 2r \Rightarrow dist(b, a) \geq 2r$ and $dist(b, c) \geq 2r$. Analogously, since τ_{bca} is passable, then: $cl(b, c, a) \geq 2r \Rightarrow dist(c, b) \geq 2r$ and $dist(c, a) \geq 2r$. Therefore $dist(a, b) \geq r$ and $dist(a, c) \geq r$, but since the channel search algorithm could

not find the shorter solution passing by traversal τ_{bac}, then τ_{bac} cannot be passable. The only possible situation is that a constrained edge s exists behind bc such that $dist(a, s) < 2r$ (because by the Delaunay property no vertex can cause the reduced clearance). But if s exists, s would as well block the passable traversals τ_{abc} and τ_{bca}. □

B. LEMMAS

LEMMA 1. (ENCLOSING RIGHT TRIANGLE.) *Given traversal τ_{abc}, there exists a point $o \in \mathbb{R}^2$ with $\angle boc = \pi/2$ such that all possible disturbances to τ_{abc} will lie in the interior of $\triangle boc$.*

PROOF. Let s be the closest constraint to τ_{abc}. If a disturbance v exists, it will be with respect to s. Now let o be the intersection of the line orthogonal to s passing by c, with the line parallel to s passing by b. Figure 19 illustrates the lines determining o for three possible configurations of s. In all cases, $\angle boc = \pi/2$. There might be points inside $\triangle boc$ that are not disturbances, but according to Definition 3, if a disturbance v exists, it will be inside $\triangle boc$ since bo delimits all points closer to s than b, and co delimits all points with valid orthogonal projections on s. □

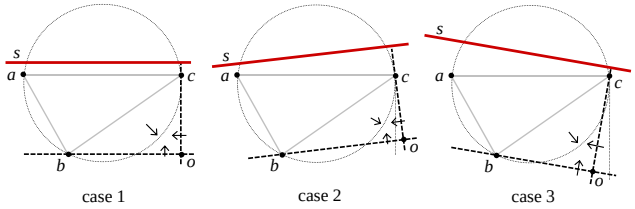

Fig. 19. Cases for Lemma 1.

LEMMA 2. *Let traversal τ_{abc} be disturbed by vertex v with respect to constraint s. Let v' be the orthogonal projection of v on s, and let d and e be the vertices connected to v forming $\triangle dve$ crossed by segment vv'. If vertex d lies on segment vv', then d is also a disturbance.*

PROOF. The key to this proof is to show that property $dist(d, s) < dist(d, e)$ is true. This can be informally shown as follows: Since v is a disturbance $\Rightarrow$ $dist(v, v') < dist(v, e)$ and thus e has to lie outside the circle centered at v with radius $dist(v, v')$. In this situation, Figure 20-left illustrates that $dist(d, v') < dist(d, e)\ \forall d \in vv' \Rightarrow dist(d, s) < dist(d, e)$, since v' is also the orthogonal projection of d on s.

An analytical proof of $dist(d, s) < dist(v, e)$ is now derived as follows. Let p be the orthogonal projection of e on segment vv', dividing vv' in two parts of lengths m and n_1, and forming the right triangle of dimensions n_1, k, and r_1, as shown in Figure 20-right. If d lies between p and v, d divides pv in two parts of lengths $n_3 > 0$ and $n_2 > 0$. If d lies between v' and p, let $n_3 = dist(d, v) > 0$ and $n_2 = -dist(d, p)$. Finally, let r_2=$dist(d, e)$. Starting from the fact that v is a disturbance, it follows that: $dist(v, v') < dist(v, e) \Rightarrow m + n_1 < r_1$

$\Rightarrow m + n_1 - n_3 < r_1 - n_3$

$\Rightarrow m + n_2 < r_1 - n_3$

$\Rightarrow m + n_2 < \sqrt{k^2 + n_1^2} - n_3$

$\Rightarrow (m + n_2)^2 < (\sqrt{k^2 + n_1^2} - n_3)^2$

$\Rightarrow (m + n_2)^2 < k^2 + n_1^2 - 2n_3\sqrt{k^2 + n_1^2} + n_3^2$

$\Rightarrow (m + n_2)^2 < k^2 + n_1^2 - 2n_3 r_1 + n_3^2$

$\Rightarrow (m + n_2)^2 < k^2 + (n_2 + n_3)^2 - 2n_3 r_1 + n_3^2$

$\Rightarrow (m + n_2)^2 < k^2 + n_2^2 + 2n_2 n_3 + n_3^2 - 2n_3 r_1 + n_3^2$

$\Rightarrow (m + n_2)^2 < k^2 + n_2^2 + 2n_2 n_3 + 2n_3^2 - 2n_3 r_1$

$\Rightarrow (m + n_2)^2 < k^2 + n_2^2 + 2n_3(n_2 + n_3 - r_1)$

$\Rightarrow (m + n_2)^2 < k^2 + n_2^2 + 2n_3(n_1 - r_1)$,

and since $n_3 > 0$ and $n_1 < r_1 \Rightarrow 2n_3(n_1 - r_1) < 0$

$\Rightarrow (m + n_2)^2 < k^2 + n_2^2 + 2n_3(n_1 - r_1) < k^2 + n_2^2 = r_2^2$

$\Rightarrow (m + n_2)^2 < r_2^2$

$\Rightarrow m + n_2 < r_2 \Rightarrow dist(d, s) < dist(d, e)$.

Since all other requirements of Definition 3 are trivially verified, it follows that d is a disturbance. □

LEMMA 3. *Let traversal τ_{abc} be disturbed by vertex v with respect to constraint s. Let v' be the orthogonal projection of v on s. If a vertex z exists closer to s than v and on the same side of vv' as b, then z is also a disturbance.*

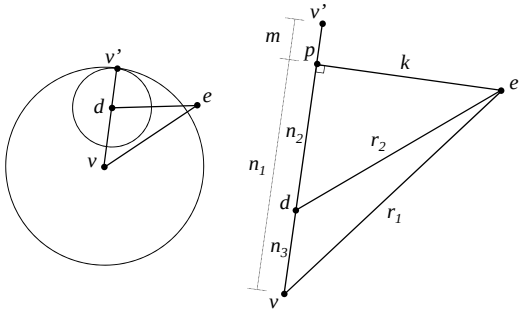

Fig. 20. Diagrams for Lemma 2.

PROOF. Let $\vec{r}$ be the unit vector parallel to s such that $v + \vec{r}$ is on the left side of vv'. If v is a disturbance and is translated to another valid location with $v + \vec{r}$, then it will remain a disturbance. This can be easily observed in Figure 20-right: if v is translated to the left, its distance to s remains the same while its distance to e increases, assuming all other requirements of Definition 3 remain valid. Let now z_0 be a point on segment vv' and $t \in \mathbb{R}^+$, such that the position of vertex z can be written as $z = z_0 + t\vec{r}$. From Lemma 2, z_0 is a disturbance $\Rightarrow$ $dist(z_0, s) < dist(z_0, e) \Rightarrow dist(z, s) = dist(z_0 + t\vec{r}, s) = dist(z_0, s) < dist(z_0, e) < dist(z_0 + t\vec{r}, e) = dist(z, e) \Rightarrow dist(z, s) < dist(z, e)$. Therefore the key property $dist(z, s) < dist(z, e)$ holds and since all other requirements of Definition 3 are trivially verified, it follows that z is a disturbance. □

C. TRIPLE DISTURBANCE EXAMPLE

While it is unlikely that all vertices in a CDT can disturb three traversals, Figure 21 illustrates a case where one vertex v simultaneously disturbs three different traversals.

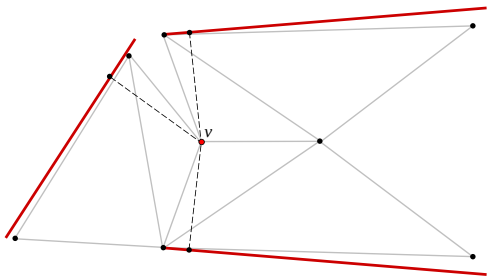

Fig. 21. Example where v disturbs three traversals.

D. ROBUSTNESS TEST ENVIRONMENT

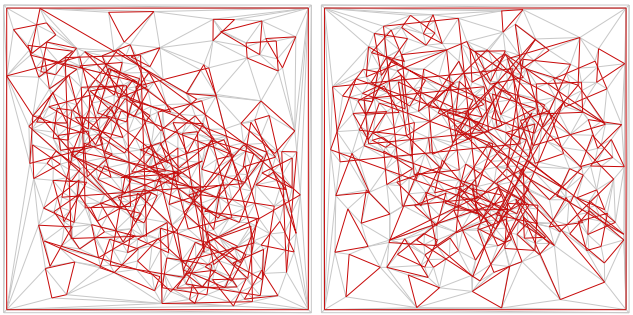

Fig. 22. Two random instances of the environment used for the reported robustness tests.