Celestial Walk: A Terminating, Memoryless Walk for Convex Subdivisions

Wouter Kuijper, Victor Ermolaev, Olivier Devillers

Celestial Walk: A Terminating, Memoryless Walk for Convex Subdivisions

Wouter Kuijper[1] Victor Ermolaev[1] Olivier Devillers[2]

[1]Nedap N.V. [2] Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

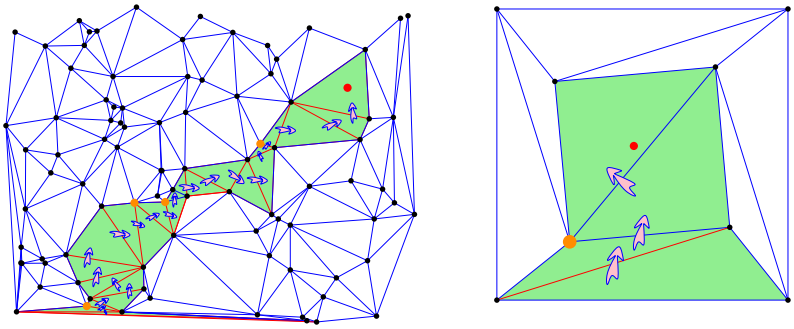


Figure 1. Celestial walk (obtuse angles marked by orange dots).

Abstract

A common solution for routing messages or performing point location in planar subdivisions consists in *walking* from one face to another using neighboring relationships. If the next face does not depend on the previously visited faces, the walk is called *memoryless*. We present a new memoryless strategy for convex subdivisions. The known alternatives are *straight walk*, which is a bit slower and not memoryless, and *visibility* walk, which is guaranteed to work properly only for Delaunay triangulations. We prove termination of our walk using a novel distance measure that, for our proposed walking strategy, is strictly monotonically decreasing.

1. Introduction

One fundamental task when working with meshes is to translate point locations to faces in the mesh. This task is, in fact, so fundamental that it is hard to find a mesh application for which it is not, at some level, relevant. This is especially true for interactive or real-time applications where user input or sensor data must continuously be related back to mesh structure and, subsequently, mesh structure must be adapted incrementally.

 ISSN 2331-7418

One of the simplest approaches to this problem is to use the local neighborhood information inherent in the mesh structure to walk from face to face until we reach the face that contains the query point. Optionally, we can make this more efficient by keeping track of an auxiliary data structure that allows us to jump to a location that is close to the query point and walk from there. In either case, the walking algorithm is instrumental to complete the point location.

In this paper, we show that existing walking algorithms still leave something to be desired. In particular, the easy-to-implement visibility walk may never converge and the more robust straight walk is computationally and memory intensive. We develop a new walking algorithm that can either be seen as a refinement of the visibility walk that is almost as easy to implement but is guaranteed to terminate, or as a simplification of the straight walk that requires less memory and less computation.

In the process, we uncover a practical distance measure to characterize distances between a point as a zero-dimensional object and an edge as a one-dimensional object. This *celestial-distance* measure works by combining angular displacement with linear displacement in an integrated fashion. As it turns out, this distance measure is key to understanding the celestial walk and proving its termination.

1.1. Related Work

Point location in a convex subdivision is a classical problem of computational geometry for which several data structures have been designed; these have good complexities in the worst case [Kirkpatrick 1983; Chazelle and Guibas 1986; Preparata 1990]. These intricate solutions are often passed over in favor of simpler algorithms based on traversal of the planar subdivisions using neighborhood relations between faces, also known as *walking algorithms* [Bose et al. 2002; Bose and Morin 2004; Devillers et al. 2002]. These walking algorithms can also be used as a building block in randomized data structures for point location [Mücke et al. 1999; Devillers 2002].

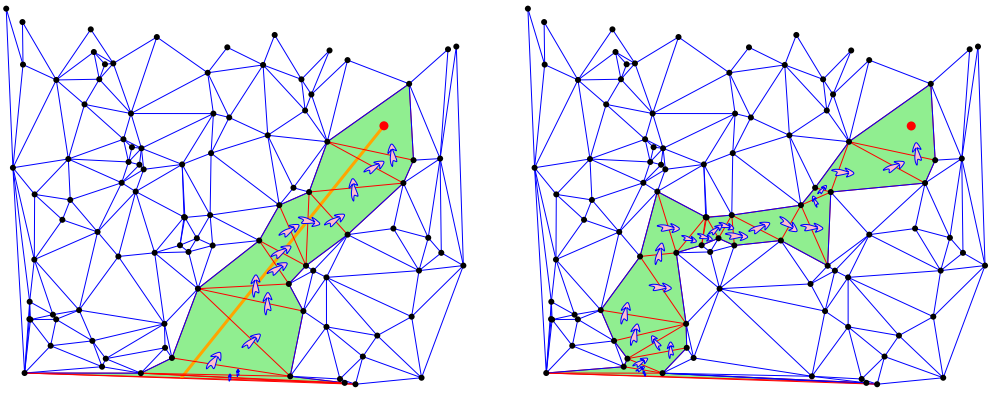


Figure 2. Straight (left) and visibility (right) walks.

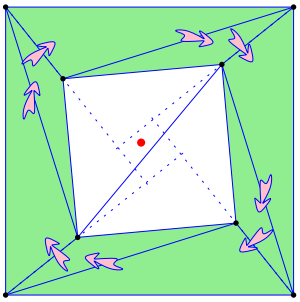


Figure 3. Visibility walk may loop.

Among convex subdivisions, Delaunay triangulations received a lot of attention because of their nice theoretical properties. For Delaunay triangulations, essentially two walking strategies are used: the *straight walk* and the *visibility walk* [Devillers et al. 2002]. The straight walk visits all faces crossed by a line segment between a known face and the query point, while the visibility walk goes from one face to another if the query point is on the side of the new face with respect to the supporting line of the edge common to the two faces (see Figure 2).

The straight walk trivially terminates in the face containing the query point and generalizes to any planar subdivision but with the inconvenience of not being memoryless: the starting face of the walk is used throughout the whole walk.

The visibility walk is memoryless, but proving its termination requires the use of certain properties of the Delaunay triangulation, and the visibility walk may actually loop in other subdivisions [Devillers et al. 2002] (see Figure 3).

Regarding performance, both walks may visit all triangles in the worst case and visit $O(\sqrt{n})$ triangles when the points are evenly distributed [Devroye et al. 2004; Devillers and Hemsley 2016]. From a practical point of view, the visibility walk is simpler to implement and a bit faster in practice because it uses less predicates to walk from a triangle to its neighbor [Devillers 2012].

Contribution

We propose the metric *celestial distance*[1] as a new way to measure the proximity between an edge of the subdivision and a query point. This distance measure enables the design of new walking strategies and proves their termination. By design these strategies are memoryless: only edges of the current face determine the edges by which the walk progresses to the next face.

Our main contribution is the *celestial walk*, which is a refinement of the well-known visibility walk. Unlike the visibility walk, our walk terminates not just on

[1]The name celestial distance refers to the practice of *celestial navigation*, where angular distances between the celestial bodies and the horizon are used for navigation at sea.

Delaunay triangulations but on arbitrary convex subdivisions. This property is particularly useful for constrained and/or incremental meshing where the conditions necessary for termination of the visibility walk can be locally and/or temporarily violated.

Another important feature of the celestial walk is that, like the visibility walk, it uses only orientation predicates to navigate the mesh. In practice, checking an orientation predicate reduces to computing the sign of a second-degree polynomial, the degree of such polynomial being a relevant measure of the predicate complexity [Liotta et al. 1998; Boissonnat and Preparata 2000]. As a consequence, it becomes relatively straightforward to implement the celestial walk in an efficient and robust manner.

2. Prerequisites

Let $\mathcal{G} = (V, E, F)$ be a *planar straight-line graph* (PSLG) consisting of a set of vertices V, a set of half–edges E, and a set of faces F.

We abstract away the borders of $\mathcal{G}$ by assuming that it tiles the entire real plane. At the same time, we rule out dense tessellations by assuming $\mathcal{G}$ is *locally finite*, i.e., the number of vertices (edges, faces) intersecting a given bounded area is always finite.

We assume $\mathcal{G}$ is given in half-edge representation. In particular, we assume the following atomic functions [de Berg et al. 1997] and provide their graphic representation in Figure 4:

$\mathsf{origin} : E \to V$ which maps every half-edge to its start vertex,

$\mathsf{target} : E \to V$ which maps every half-edge to its end vertex,

$\mathsf{edge} : F \to E$ which maps every face to some edge on its perimeter,

$\mathsf{face} : E \to F$ which maps every half-edge to its corresponding (left-hand-side) face,

$\mathsf{next} : E \to E$ which maps every half-edge to its successor half-edge in the counterclockwise winding order of the face perimeter,

$\mathsf{twin} : E \to E$ which maps every half-edge to its twin half-edge running in the opposite direction, i.e., $\mathsf{twin}(\mathsf{twin}(e)) = e$, $\mathsf{origin}(e) = \mathsf{target}(\mathsf{twin}(e))$ and vice versa.

The *point location problem* in PSLGs can now be formulated as follows: given some goal location $p \in \mathbb{R}^2$ and an initial half-edge $e_{\mathsf{init}} \in E$, find some goal edge $e_{\mathsf{goal}} \in E$ such that $p \in \mathsf{face}(e_{\mathsf{goal}})$ using only $\mathsf{next}(\cdot)$ and $\mathsf{twin}(\cdot)$ to get from one half-edge to another, i.e., there must exist a finite path $e_{\mathsf{init}} = e_0 \dots e_n = e_{\mathsf{goal}}$ such that for all $0 < i \leq n$, $e_i = \mathsf{next}(e_{i-1})$ or $e_i = \mathsf{twin}(e_{i-1})$.

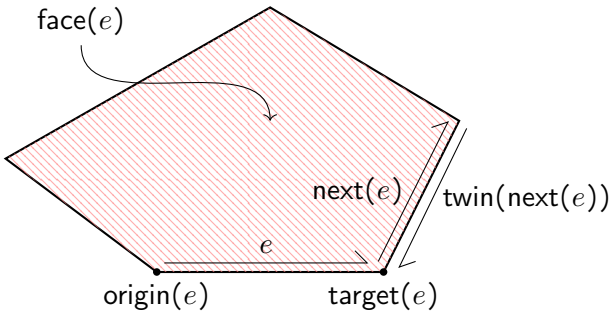


Figure 4. Atomic functions applied to a face in the mesh. For any half-edge e of the face it is true that the half-edge $e' = \mathsf{edge}(\mathsf{face}(e))$ belongs to the face, too.

3. Celestial Distance

One problem that we encounter when we try to use Euclidean distance as a measure of progress for a walking algorithm is the fact that Euclidean distance is not always strictly decreasing for every step in the walk. Thus, it is not possible to prove termination using Euclidean distance alone. For this reason, we define the following augmented distance measure on (half-)edges.

For a given point $p \in \mathbb{R}^2$ and a half-edge $e \in E$, we define the *celestial distance of e to p* as a pair $\mathsf{cd}(e, p) = [d, \alpha]$, where d is the length of the line segment from p to the closest point on e and α is the *wide angle*[2] between e and this line segment or 0 when $d = 0$. Some illustrations of celestial distances for various relative point locations are given in Figure 5.

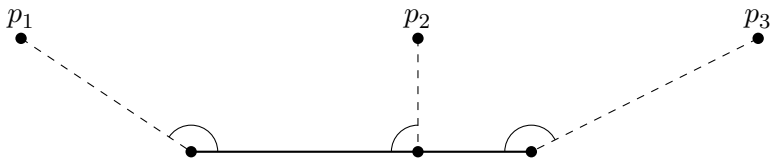


Figure 5. Examples of celestial distances as a combination of linear and angular displacement for given edge e and various locations of the query point (p_1, p_2, or p_3).

We now define a lexicographic order on celestial distances as follows:

$$[d, \alpha] < [d', \alpha'] \text{ iff } d < d' \vee (d = d' \wedge \alpha < \alpha').$$

We illustrate the application of this new distance measure in Figure 6. The figure shows a convex face and the resulting partition of the plane obtained by grouping together points based on their closest edge. This leads to two superimposed Voronoi

[2] The largest of the two unsigned angles between two lines, i.e., $\frac{\pi}{2} \leq \alpha \leq \pi$

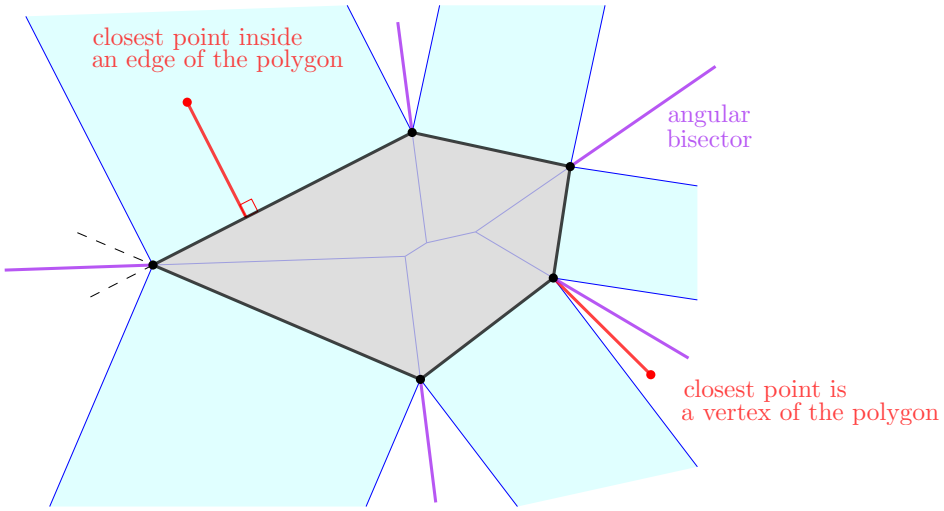


Figure 6. Voronoi diagrams of a polygon. In blue the segment Voronoi diagram of the open edges and the vertices; in purple the celestial Voronoi diagram of the edges.

partitions: one based on classical Euclidean distance and one based on celestial distance. The partition based on celestial distance is a proper refinement of the partition based on Euclidean distance in the sense that points which were ambiguous under Euclidean distance are now partitionable under celestial distance. Note that the points in the *corner-cones* radiating outward from the vertices are all ambiguous under Euclidean distance (since their closest point on the polygon is the corner vertex which is shared by two edges), yet under celestial distance they can be further partitioned. In particular, for the partition based on celestial distance, the ambiguous corner-cones are split according to the angular bisectors.

4. Abstract Walk

With all the prerequisites and our celestial distance measure in place, we are in a position to present the walking algorithm properly. We first present an abstract version of the algorithm and give a correctness proof.

In the next section, we will give an efficient, concrete instantiation of this abstract version where the computation of the celestial distances will be completely implicit. However, the correctness of the final version will rest on the correctness proof of the abstract version as given in this section.

The algorithm is rather simple: given a starting edge e and a query point p to the left of e, we select an edge of the $\mathsf{face}(e)$ that improves the distance to p. The abstract walk is formalized in Algorithm 1.

In lines 1-3, we bootstrap the invariant that the query point is always to the left of the current edge. In line 4, we bootstrap the current set of successor candidates. In

Algorithm 1 Abstract walk.

1: **if** p strictly right of e **then**
2: $\quad e \leftarrow \mathsf{twin}(e)$
3: **end if**
4: $E' \leftarrow \{e\}$
5: **while** $E' \neq \varnothing$ **do**
6: $\quad e \leftarrow \mathsf{select}(E')$
7: $\quad E' \leftarrow \{\mathsf{twin}(e') \mid e' \in \mathsf{face}(e) \wedge \mathsf{cd}(e', p) < \mathsf{cd}(e, p) \wedge p \text{ strictly right of } e'\}$
8: **end while**
9: **return** $\mathsf{face}(e)$

line 5, we enter the main loop. The loop invariant ensures that the loop will terminate as soon as there are no more suitable successor edges that have lower celestial distance to the query point. In the proof below, we will see how this condition is sufficient to ensure that, at termination, it holds that $p \in \mathsf{face}(e)$. In line 6, we nondeterministically select one of the candidate successor edges. In line 7, we compute the next set of candidate successor edges, which are all edges on the current face perimeter that have the query point on the right and have smaller celestial distance to the query point than the current edge.

Theorem 1. *For any locally finite planar subdivision, starting edge, and query point, Algorithm 1 terminates with* $p \in \mathsf{face}(e)$.

Proof. To see that our algorithm is guaranteed to terminate, we first consider the set of all edges that have celestial distance to the query point smaller than or equal to the starting edge. It will be clear that the algorithm will never stray outside of this submesh, since, at every iteration, it will only select a successor edge with strictly smaller celestial distance than the previous.

Now, due to local finiteness of the mesh, the number of edges inside the aforementioned submesh is finite too. Since the algorithm follows a chain of edges with strictly descending celestial distance (inside this finite set), it must eventually terminate.

It remains to prove that, upon termination, it will hold that $p \in \mathsf{face}(e)$. To this end, we proceed by contradiction. In particular, we will prove that, as long as $p \notin \mathsf{face}(e)$, there will be at least one suitable successor edge e' with strictly smaller celestial distance. If we can indeed establish the latter, it will show that the celestial walk would not terminate until, indeed, $p \in \mathsf{face}(e)$.

More formally, we need to prove that: for a half-edge e and goal location p such that p is to the left-of e but not in $\mathsf{face}(e)$, there always exists another e' in the perimeter of $\mathsf{face}(e)$ such that p is strictly on the right of e' and $\mathsf{cd}(e', p) < \mathsf{cd}(e, p)$. The latter would suffice because, in that case, e' would be included as a potential successor edge in set E' as defined on Line 7 in Algorithm 1. As soon as there is one poten-

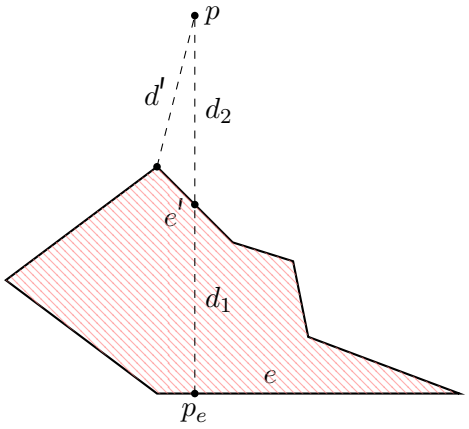


Figure 7. Illustration of the first case in the correctness proof: the ray to the current edge goes through the face interior; it holds that distance $d' < d_1 + d_2$.

tial successor edge, the walk cannot terminate which then establishes the required contradiction.

So let p_e be the point on e closest to p, and let r be the ray from p to p_e. We now distinguish two cases, illustrated in Figures 7 and 8, respectively. Note that, in the illustrations, we deliberately show non-convex faces. We do this to emphasize the fact that the present proof does *not* depend on the faces being convex. Later in the paper, we do introduce the assumption of convexity in order to obtain a more efficient implementation. However, for the moment, this actually means that for Algorithm 1, we are proving the stronger property that it is correct for *arbitrary* planar subdivisions. The two cases we distinguish are the following:

1. The ray r to the current edge e intersects the interior of $\mathsf{face}(e)$—this case is illustrated in Figure 7.

2. The ray r to the current edge e does *not* intersect the interior of $\mathsf{face}(e)$—this case is illustrated in Figure 8.

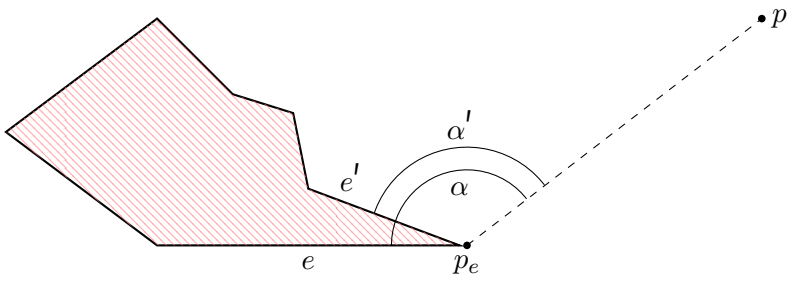


Figure 8. Illustration of the second case in the correctness proof: the ray to the current edge does not go through the face interior—it holds that wide-angle $\alpha' < \alpha$.

First, for case 1, it must hold that r intersects the boundary of $\mathsf{face}(e)$ at least one more time in another edge e', otherwise p would lie in $\mathsf{face}(e)$. It follows that e' is closer to p than the current edge (for the Euclidean distance). From this, in turn, it follows $\mathsf{cd}(e', p) < \mathsf{cd}(e, p)$, and we conclude the algorithm would progress to $\mathsf{twin}(e')$.

Next, for case 2, assume that r does *not* intersect the interior of $\mathsf{face}(e)$. Because p lies to the left of e, it cannot be the case that p_e lies in the interior of e since the ray would then also pass through the interior of $\mathsf{face}(e)$. Now, because p_e cannot lie on the interior of e, it must hold, by elimination, that $p_e = \mathsf{origin}(e)$ or $p_e = \mathsf{target}(e)$.

Assume, without loss of generality, that $p_e = \mathsf{target}(e)$. We claim that, although the Euclidean distance may stay the same, it holds that $e' = \mathsf{next}(e)$ has a strictly smaller celestial distance to p. Intuitively, if the Euclidean distance does not improve, the angle of the next edge with the ray *will* improve because the next edge will "pivot" towards the goal p.

To see the latter, assume, again without loss of generality, that e' provides no improvement over e with respect to Euclidean distance to p. It then immediately follows that $p_{e'}$ (the point on e' that is closest to p) must be the only point that e and e' share which, by assumption, is $p_e = \mathsf{target}(e) = \mathsf{origin}(e') = p_{e'}$. Since r does not intersect the boundary of $\mathsf{face}(e)$ in any point other than p_e, it must hold that e' is between e and r.

Now, from Figure 8, we see that: $\mathsf{wideangle}(e, r) = \mathsf{angle}(e, e') + \mathsf{wideangle}(e', r)$ which implies $\mathsf{wideangle}(e', r) < \mathsf{wideangle}(e, r)$. Thus, $\mathsf{cd}(e', p) < \mathsf{cd}(e, p)$, and we conclude that the algorithm would progress to $\mathsf{twin}(e')$. □

5. Celestial Walk

In the previous section, the successor of an edge in the walk can be the twin of any edge of the current face with a smaller distance to the query point. In this section, we explain a way to actually select such an edge using only orientation checks.

The main problems that we are solving in this section are the facts that Algorithm 1 is non-deterministic and relies on the explicit computation of celestial distances. Both of these properties make it less immediately applicable. In this section, we therefore develop a derived algorithm that works for *convex* PSLGs. As we shall see, the additional assumption of convexity allows us a significantly more efficient walk.

So, let us first consider the problem of determining a successor edge that has lower celestial distance than our current edge without having to explicitly compute these distances.

As an example of a convex face, consider once more Figure 6. If we assume the query point is outside the face, this leaves two possibilities. First, the query point may

be located inside one of the orthogonal slabs (indicated in blue in Figure 6), which means the closest point coincides with the orthogonal projection of the query point on the edge. Second, the query point may be located in one of the intermediate corner-cones separating the orthogonal slabs which means the closest point coincides with the corner vertex.

The latter argument gives us a basic refinement of Algorithm 1 for the convex case: check if the query point is in one of the orthogonal slabs; if so, pick the corresponding edge, else the query point must be inside some corner-cone between two orthogonal slabs. in this case, we can pick either edge (unless one of them is our current edge in which case we are forced to pick the other alternative in order to make progress).

To do even better, we can resolve the remaining non-determinism by using the angle as a tiebreaker to determine which of the two candidate edges is best (i.e., which edge is most favorably oriented towards the query point). Theoretically, the best tiebreaker is the angular bisector (indicated in purple in Figure 6). The only problem is that the explicit computation and representation of the exact angular bisector is at least as hard as the explicit computation and representation of celestial distances.

Fortunately, we can avoid the explicit computation of angular bisectors as well. In particular, we make the following case distinction. If the corner vertex is *not obtuse* (more precisely, the face has an acute or right internal angle at that corner vertex), it holds that the extensions of both edges lie inside the corner-cone (i.e., the leftmost corner in Figure 6). This means that we may use either edge itself as a crude approximation to the angular bisector (this is precisely what the visibility walk does in *all* cases). On the other hand, if the corner vertex is *obtuse*, it holds that the normal to the base of the triangle spanned by the two candidate edges forms a suitable approximation to the angular bisector (because it lies properly inside the corner-cone). In Figure 9, we illustrate this construction. Note that, in the limit, as the internal angle

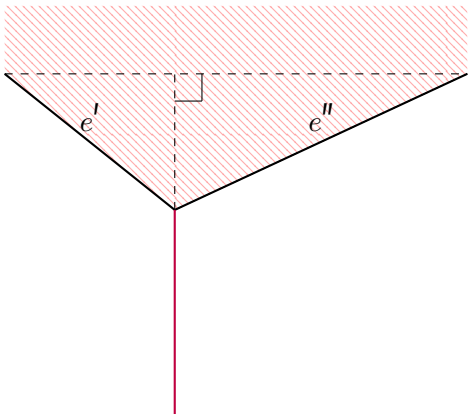


Figure 9. Construction for approximating angular bisector.

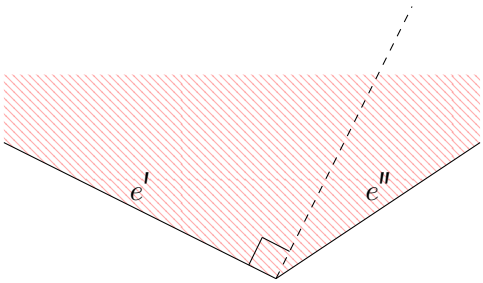


Figure 10. Reducing obtuseness check to orientation check.

approaches 180 degrees, or, alternatively, the ratio between the lengths of the two edges approaches 1, the base normal converges to the exact angular bisector.

The latter considerations lead us to define $\mathsf{approx\_bisector}(e', e'')$ for some pair of consecutive edges e', e'', such that $e'' = \mathsf{next}(e')$ and $\mathsf{angle}(e', e'')$ is obtuse. In particular, we let $\mathsf{approx\_bisector}(e', e'')$ denote the line from $\mathsf{target}(e')$—which is equal by definition to $\mathsf{origin}(e'')$—in the direction orthogonal to, and to the right of, the internal baseline segment that connects $\mathsf{origin}(e')$ and $\mathsf{target}(e'')$. For an illustration of this construction, see Figure 9. We also define $\mathsf{obtuse}(e', e'')$ as a predicate that is true iff $\mathsf{target}(e'')$ is strictly right of the line from $\mathsf{target}(e')$—which we assumed is equal to $\mathsf{origin}(e'')$—in the direction orthogonal to, and to the left of, e'. For an illustration of this construction, see Figure 10. With these additional definitions, Algorithm 2 formalizes the celestial walk.

In lines 1-3, we bootstrap the invariant that the query point is always to the left of the current edge. In line 4, we initialize a perimeter edge variable used for iterating over the outline of the current face. In line 5, we enter the outer loop. As can be seen by inspecting the code-paths inside the loop body, the loop invariant ensures that the loop will terminate iff the query point is inside the current face. In line 6, we check if the query point lies strictly to the right of the perimeter edge. If this is the case we know, due to the face convexity, that the query point is outside the face. From that moment on, the only goal is to find an edge that allows us to walk to a neighboring face. Therefore, in line 7, we introduce a second perimeter edge variable that will be used to iterate over *pairs* (e', e'') of consecutive edges rather than singletons. In line 8, we state the negation of the condition that we are looking for: either the internal angle of the face is acute[3] or, otherwise, the query point falls strictly to the right of the approximate bisector. In line 9, we shift the pair of edges[4] along the face perimeter until said condition is reached. In line 11, we have exited the inner

[3]Note that the acuteness/obtuseness check in line 8 can potentially be memoized by the mesh at the expense of two bits per half-edge, or pre-computed, in bulk or incrementally, at the expense of one bit per half-edge.

[4]For meshes that are guaranteed to be triangular, minor optimizations are possible by unrolling the loops of this general algorithm.

Algorithm 2 Celestial walk.

```
1:  if p strictly right of e then
2:      e ← twin(e)
3:  end if
4:  e' ← next(e)
5:  while e ≠ e' do
6:      if p strictly right of e' then
7:          e'' ← next(e')
8:          while obtuse(e', e'') and p left of approx_bisector(e', e'') do
9:              e', e'' ← e'', next(e'')
10:         end while
11:         e ← twin(e')
12:         e' ← next(e)
13:     else
14:         e' ← next(e')
15:     end if
16: end while
17: return face(e)
```

loop so we know that e' contains the edge that is "sufficiently closest" to the query point for us to continue the walk to the neighboring face. Hence, we re-assign e and e' and drop back into the outer loop as though we would have restarted the algorithm on this new edge.[5]

6. Experiments

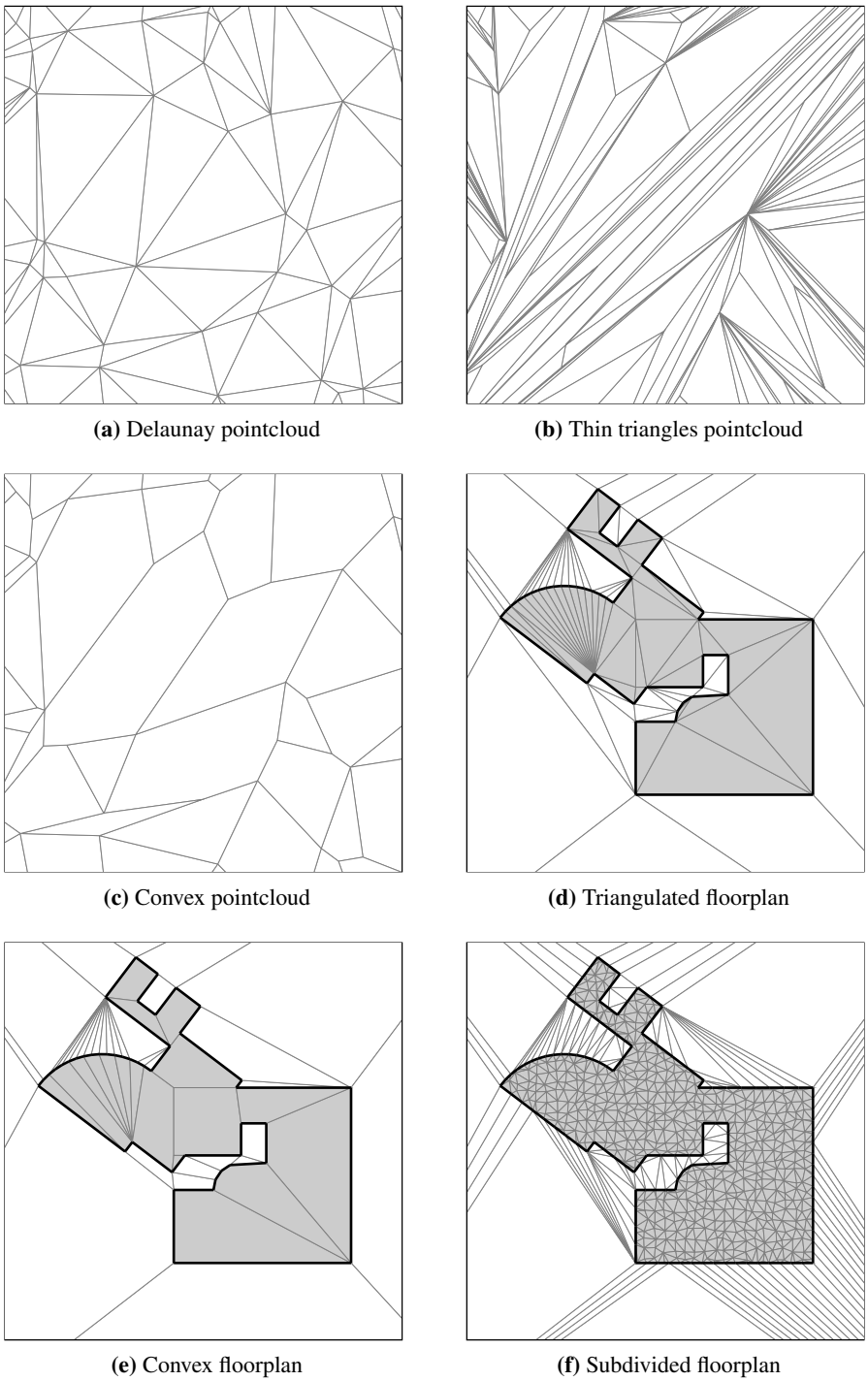
In this section we detail several experiment that were carried out to validate the relative performance of the celestial walk versus its two main competitors: the visibility walk and the straight walk.

Two sets of experiments were performed: a set of small-scale experiments with a slow, prototype implementation and a set of larger-scale experiments based on CGAL [The CGAL Project 2017]. The former compares various mesh types, while the latter shows bottom-line, wall-clock timings.

6.1. Small Scale Experiments

A prototype implementation [Kuijper 2017] was used to run a set of small-scale experiments comparing the performance of the various walks. The performance was

[5] Alternatively, a recursive formulation of the celestial walk is possible, by substituting a recursive call at line 12.



(a) Delaunay pointcloud

(b) Thin triangles pointcloud

(c) Convex pointcloud

(d) Triangulated floorplan

(e) Convex floorplan

(f) Subdivided floorplan

Figure 11. Examples of the mesh types that were used in the experiments.

measured by counting the number of orientation tests for each walk.[6] In order to make the tests more representative, a collection of mesh types was used. The mesh types were designed to exhibit many features that would be encountered in practice. In particular, the algorithms were tested on both non-constrained and constrained meshes, both triangulated and merely convex meshes, and at various levels of density. For a random specimen of each of the six mesh types that were used, see Figure 11.

The first set of three mesh types consists of unconstrained meshes based on uniformly distributed pointclouds: a high-quality mesh meeting the Delaunay criterion, a very low-quality mesh where triangles are very thin and do not meet the Delaunay criterion, and a non-triangular mesh created by starting from a Delaunay mesh and removing edges repeatedly as long as the resulting faces are convex.

The second set of three mesh types consists of constrained meshes, i.e., some edges are fixed to form the outline of a polygon. In our case we created a synthetic dataset of randomly generated polygons called "floorplans."[7] Each randomly generated floorplan occurs in three variants: a minimal triangulation that has been flipped to approach the Delaunay criterion as closely as possible,[8] a convex subdivision obtained by subsequently removing edges as long as the resulting faces are convex, and a subdivided triangulation (i.e., a fine-grained mesh that might be used in a finite-element analysis).

Next we tested each walk type for each mesh type. For the celestial walk, we additionally tested a balanced version that alternates between clockwise and counterclockwise face winding order.[9] The walks traversed the center of each mesh in each of eight compass directions. In Figure 12, we show an example consisting of each of the walk types for the south-west to north-east compass direction on one of the subdivided floorplan meshes.

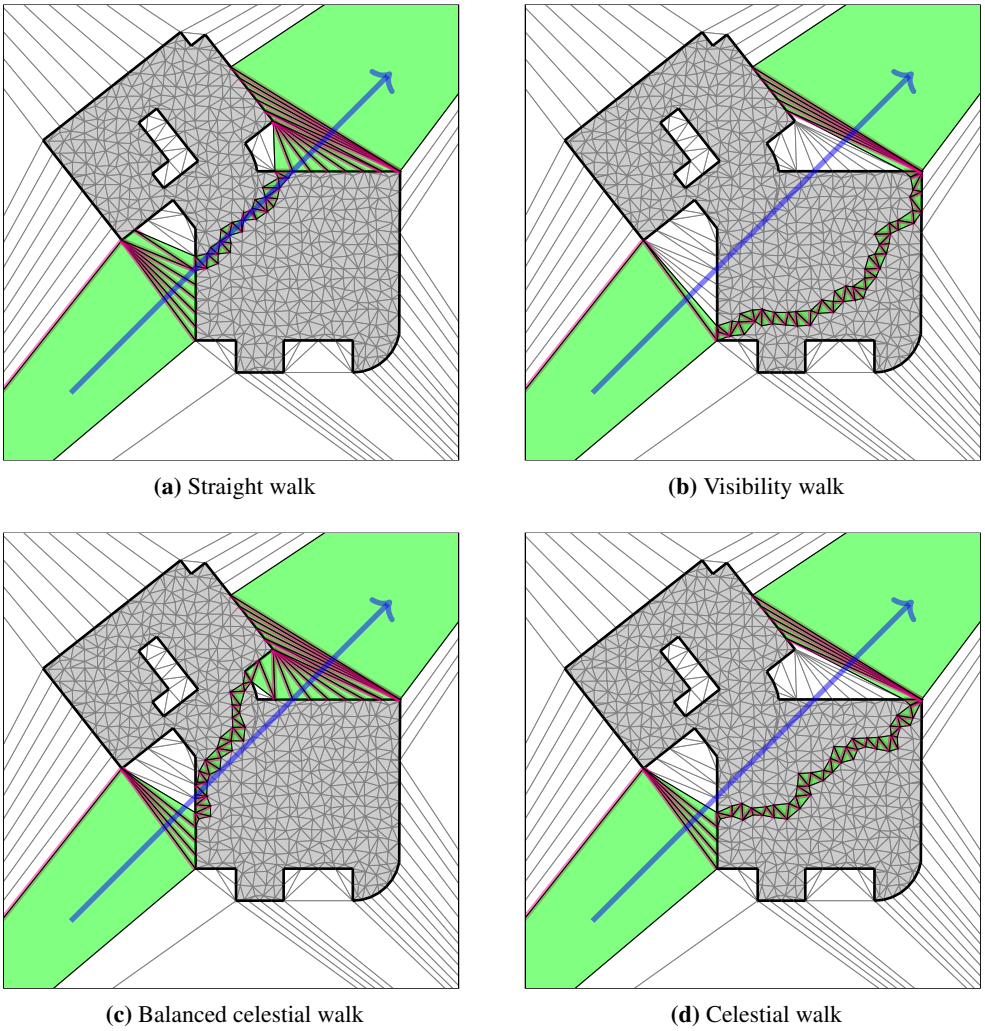
The experiments described above yielded a total of 3072 data points: eight compass directions, for each of 16 randomly generated meshes, for each of six mesh types, for each of four walk types. For details on the generation procedure or for reconstructing the synthetic dataset (based on the included random seeds) cf. [Kuijper 2017].

[6] The implementation uses dynamic programming to make sure the straight walk incurs the absolute minimal number of orientation tests possible. In addition, the implementation maintains the obtusity of each angle incrementally (at mesh construction time) so these additional orientation checks are considered part of the mesh construction rather than the walks and are not reflected in the results (the obtusity bits can, after all, be used for multiple walks).

[7] Because their design is loosely based on building floorplans with irregular outlines and features like holes (i.e., interior garden) and fillets (i.e., rounded facades).

[8] Note that, in a constrained mesh, it is not always possible to fully achieve the Delaunay criterion because the constrained edges cannot be flipped.

[9] Note that this is rather expensive in the sense that, in addition to the `next()` relation, we need to maintain also the `prev()` relation for the entire mesh



(a) Straight walk **(b)** Visibility walk

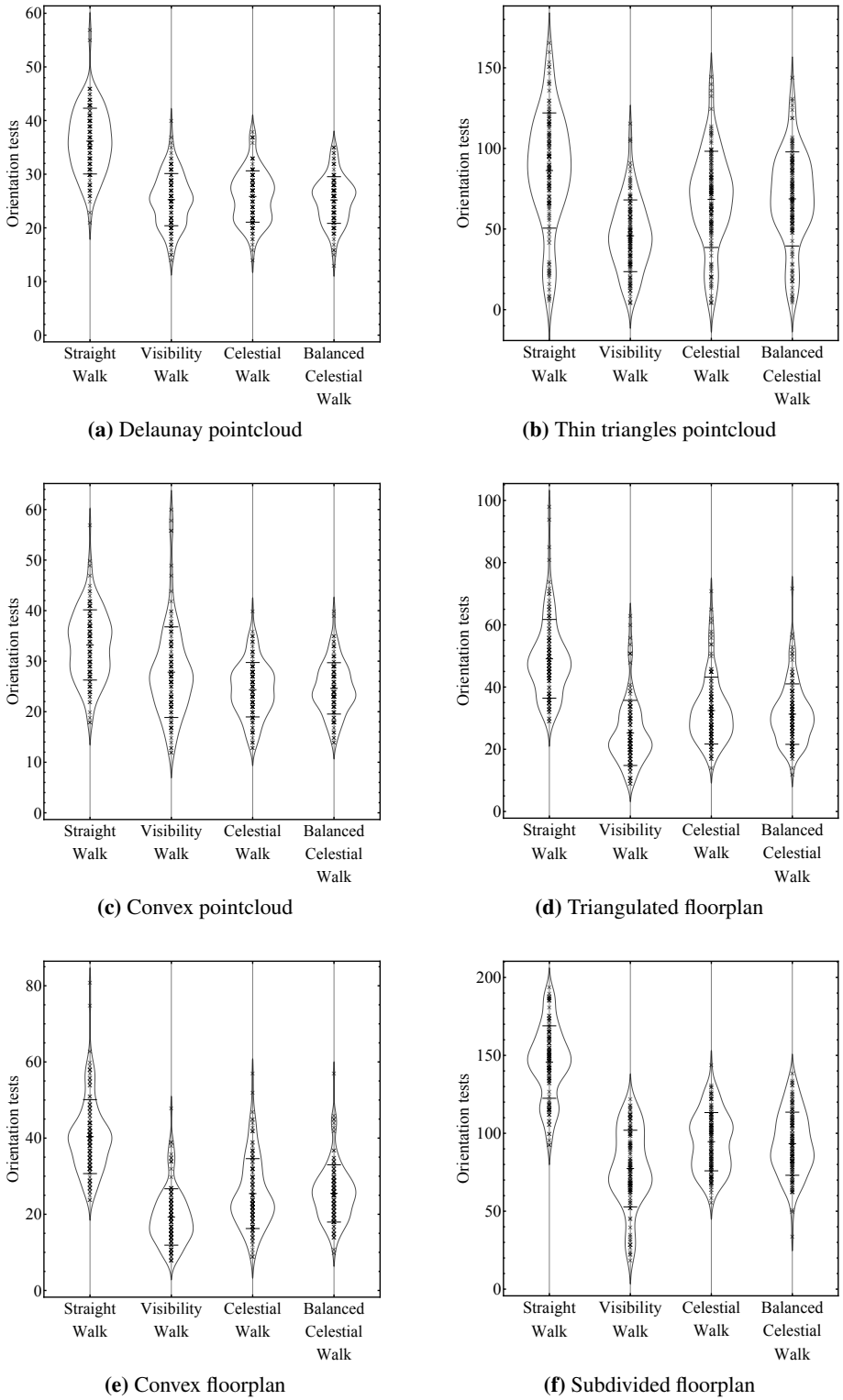
(c) Balanced celestial walk **(d)** Celestial walk

Figure 12. Example paths for four different walking strategies on subdivided meshes.

Results

In Figure 13, we show the results of the various walk types on each of the various mesh types. The graphs show both a scatterplot of the raw data, as well as a violin plot, where the width of the violin shows the local density of the scatterplot. The horizontal bars indicate mean and standard deviation. The general ranking seems to be that the visibility walk is most efficient, followed closely by the celestial walk and the balanced celestial walk. Balancing seems to have no real statistically significant effect except perhaps on some of the outliers. The straight walk uses more orientation tests on each of the mesh types.

We do see some interesting relative performance differences between the various mesh types. In particular, the performance of the celestial walk seems to degrade more



(a) Delaunay pointcloud

(b) Thin triangles pointcloud

(c) Convex pointcloud

(d) Triangulated floorplan

(e) Convex floorplan

(f) Subdivided floorplan

Figure 13. Number of orientation tests performed for each of the mesh types.

sharply than the visibility walk on the poor quality, thin triangle meshes. Presumably this has to do with an increase in obtuse angles that require additional orientation tests. Another interesting result is that, on convex pointcloud meshes, the celestial walk seems to perform slightly better than the visibility walk. Presumably this has to do with the fact that the visibility walk can veer quite widely of course on convex meshes because it is too greedy in moving to the neighboring face while iterating over the convex face perimeter.

In Figure 14, we show the overall results of the various walks on the full, combined set of meshes. These results confirm the general ranking and show that there is no statistically significant performance penalty in switching form the visibility to the celestial walk while there is still a statistically significant performance gain with respect to the straight walk.

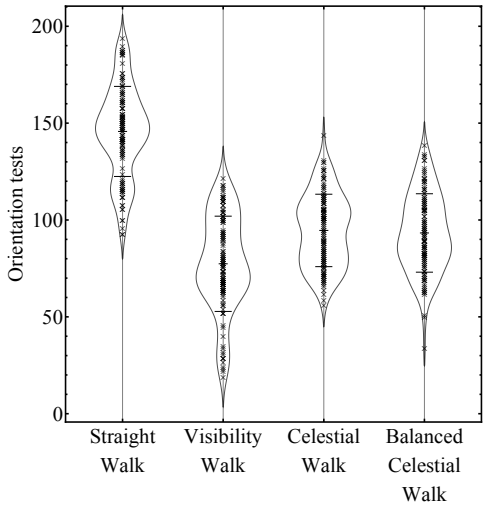


Figure 14. Number of Orientation tests performed on all of the mesh types combined.

6.2. CGAL Experiments

We also implemented the celestial walk with CGAL [The CGAL Project 2017],[10] for comparison with the straight and visibility walk provided by the library.[11] The different strategies have been tested to perform a walk towards a random query point in both Delaunay and non-Delaunay triangulation. Triangulations have one million random points, mean time and its variance are computed for a sample of 1000 strategy runs (see Table 1).

[10]CGAL code is available at http://jcgt.org/published/0007/03/02/cgal_code.zip. Run times in Tables 1 and 2 are on a MacBook Air laptop, processor 1.7GHz Intel Core i7.

[11]Note that the straight walk uses exact orientations tests (because of its sensitivity to robustness-failures) whereas the celestial walk and visibility walk implementations rely on structural filtering instead (orientation tests use rounded arithmetic except for the final triangle).

Walk \ Triangulation Time	μs, Delaunay	μs, non-Delaunay
Straight	230 ± 11	3198 ± 560
Visibility	143 ± 9	2227 ± 783
Celestial	201 ± 18	$2120 \pm 554.$
Celestial with pre-computed obtuseness	143 ± 8	2202 ± 651

Table 1. Run times of the different walks.

Computation \ Triangulation Time	s, Delaunay	s, non-Delaunay
Triangulation	0.899 ± 0.045	1.333 ± 0.148
Obtuseness computation	0.041 ± 0.007	0.045 ± 0.007

Table 2. Triangulation vs. obtuseness-computation times.

Results

We observe no significant differences in run times between the visibility walk and the celestial walk when the obtuse triangles are precomputed. If the obtuse triangles are not precomputed, then the celestial walk becomes slower but remains faster than the straight walk. The time for precomputing the obtuse triangles is small with respect to the time for computing the triangulation (see Table 2).[12]

7. Conclusion

We have shown how our celestial distance measure enables the design of new walking strategies. In particular, we improved the applicability of the popular visibility walk by refining it into what we call the celestial walk. Our walk is almost as simple to implement and is guaranteed to terminate on arbitrary convex subdivisions.

If we assume that all obtuseness checks are pre-computed or memoized (at the expense of one or two bits per half-edge, respectively) we can make the following observations with respect to the expected complexity of our walk:

- The worst case for our walk is when all angles in the mesh are obtuse, i.e., in a regular hexagonal mesh. In this case, we require two orientation tests per visited half-edge (the same number as the straight walk).

[12] Actually the time indicated in the table is the time to iterate on the triangles and compute the angles; the computation can be even cheaper if it is done on the fly when the triangle is created.

- For triangulations, a simple counting argument proves that there is at most a fraction of $\frac{1}{3}$ of obtuse angles. This gives us a worst-case rough estimate of $\frac{4}{3}$ orientation tests per visited half-edge (better than the straight walk and only slightly worse than the visibility walk).

- In practice, based on the experiments as detailed in this paper, we expect better performance since the fraction of obtuse angles will be much less than $\frac{1}{3}$ in triangulations that exhibit some regularity.

Future Work

A natural next question to consider is whether the celestial walk can be generalized to three dimensions. We have some reason to believe this is the case.

In 2D, as we progress according to the celestial walk, we either move forward in the direction of the query point or *orient* ourselves towards it. Similarly, in three dimensions, we expect to either progress in the Euclidean sense or *roll* in the direction of the query point. This intuition leads to developing a suitable non-increasing distance measure quantifying this relationship. In particular tor the walking problem in tetrahedralization, we find that a combination of Euclidean distance as the first progression criterion and two successive angles are sufficient to guarantee the progress towards the tetrahedron containing the query point [Kuijper et al. 2017].

Acknowledgements

The authors wish to thank the anonymous reviewers and JCGT editors for their valuable suggestions and their help in improving the paper and getting it ready for publication.

References

BOISSONNAT, J.-D., AND PREPARATA, F. P. 2000. Robust plane sweep for intersecting segments. *SIAM Journal on Computing 29*, 5, 1401–1421. doi:10.1137/S0097539797329373. 32

BOSE, P., AND MORIN, P. 2004. Online routing in triangulations. *SIAM Journal on Computing 33*, 4, 937–951. doi:10.1137/S0097539700369387. 30

BOSE, P., BRODNIK, A., CARLSSON, S., DEMAINE, E. D., FLEISCHER, R., LÓPEZ-ORTIZ, A., MORIN, P., AND IAN MUNRO, J. 2002. Online routing in convex subdivisions. *International Journal of Computational Geometry & Applications 12*, 04, 283–295. doi:10.1142/S021819590200089X. 30

CHAZELLE, B., AND GUIBAS, L. J. 1986. Fractional cascading: I. a data structuring technique. *Algorithmica 1*, 1, 133–162. doi:10.1007/BF01840440. 30

DE BERG, M., VAN KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O. 1997. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin. URL: http://www.cs.uu.nl/geobook/. 32

DEVILLERS, O., AND HEMSLEY, R. 2016. The worst visibility walk in a random Delaunay triangulation is $o(\sqrt{n})$. *Journal of Computational Geometry 7*, 1, 332–359. URL: https://hal.inria.fr/hal-01348831, doi:10.20382/jocg.v7i1a16. 31

DEVILLERS, O., PION, S., AND TEILLAUD, M. 2002. Walking in a triangulation. *International Journal of Foundations of Computer Science 13*, 181–199. URL: https://hal.inria.fr/inria-00102194. 30, 31

DEVILLERS, O. 2002. The Delaunay hierarchy. *International Journal of Foundations of Computer Science 13*, 163–180. URL: https://hal.inria.fr/inria-00166711. 30

DEVILLERS, O. 2012. Delaunay triangulations, theory vs practice. In *28th European Workshop on Computational Geometry*, EuroCG. URL: https://hal.inria.fr/hal-00850561. 31

DEVROYE, L., LEMAIRE, C., AND MOREAU, J.-M. 2004. Expected time analysis for Delaunay point location. *Computational geometry 29*, 2, 61–89. doi:10.1007/PL00009234. 31

KIRKPATRICK, D. 1983. Optimal search in planar subdivisions. *SIAM Journal on Computing 12*, 1, 28–35. doi:10.1137/0212002. 30

KUIJPER, W., ERMOLAEV, V., AND BERNTSEN, H. 2017. Zig-zagging in a triangulation. *arXiv preprint arXiv:1705.03950*. 47

KUIJPER, W., 2017. Celestial walk demo. https://github.com/wkuijper/celestial-walk-demo. 40, 42

LIOTTA, G., PREPARATA, F. P., AND TAMASSIA, R. 1998. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM Journal on Computing 28*, 3, 864–889. doi:10.1137/S0097539796305365. 32

MÜCKE, E. P., SAIAS, I., AND ZHU, B. 1999. Fast randomized point location without preprocessing in two-and three-dimensional Delaunay triangulations. *Computational Geometry 12*, 1-2, 63–83. doi:10.1016/S0925-7721(98)00035-2. 30

PREPARATA, F. P. 1990. Planar point location revisited. *International Journal of Foundations of Computer Science 1*, 01, 71–86. doi:10.1142/S0129054190000072. 30

THE CGAL PROJECT. 2017. *CGAL User and Reference Manual*, 4.11 ed. CGAL Editorial Board. URL: http://doc.cgal.org/. 40, 45

Author Contact Information

Olivier Devillers
INRIA
615 rue du Jardin Botanique,
B.P. 101,
54602 Villers-ls-Nancy cedex,
FRANCE
members.loria.fr/Olivier.Devillers/

Wouter Kuijper and Victor Ermolaev
Nedap Security Management
Parallelweg 2,
7141 DC Groenlo,
The Netherlands
wouter.kuijper@nedap.com
victor.ermolaev@bitfury.com

Wouter Kuijper, Victor Ermolaev, and Olivier Devillers, Celestial Walk: A Terminating Oblivious Walk for Convex Subdivisions, *Journal of Computer Graphics Techniques (JCGT)*, vol. 7, no. 3, 29–49, 2018
http://jcgt.org/published/0007/03/02/

Received:	2017-04-17		
Recommended:	2018-03-07	Corresponding Editor:	Reynold Bailey
Published:	2018-09-04	Editor-in-Chief:	Marc Olano