Constructing the City Voronoi Diagram Faster[*]

Robert Görke[†] Chan-Su Shin[‡] Alexander Wolff[†]

Submitted to IJCGA (special issue on VD'05) April 2006. Revised August 2006.

Abstract

Given a set P of n point sites in the plane, the city Voronoi diagram subdivides the plane into the Voronoi regions of the sites, with respect to the city metric. This metric is induced by quickest paths according to the Manhattan metric and an accelerating transportation network that consists of c non-intersecting axis-parallel line segments. We describe an algorithm that constructs the city Voronoi diagram (including quickest path information) using $O((c+n)\text{polylog}(c+n))$ time and storage by means of a wavefront expansion. For $c \in \Omega(\sqrt{n}\log^3 n)$ our algorithm is faster than an algorithm by Aichholzer et al., which takes $O(n\log n + c^2\log c)$ time.

Key words: wavefront expansion, city Voronoi diagram, straight skeleton, closest pair, minimization query, transportation network

1 Introduction

Imagine Manhattan in 2050—void of car traffic. Only a network of conveyors accelerates the movement of countless busy visitors in this huge pedestrian zone. As is known streets are arranged isothetically in Manhattan, so given a general direction, pedestrians can intuitively find a footpath to one of the many post offices. But time is precious, and thus a technique is required, telling an arbitrary pedestrian the quickest path to the post office that can be reached most quickly. Detours utilizing the transportation network should be accepted if they help to save time. A courier service with several staging posts faces a similar problem. For any incoming job it has to be determined how and starting from which post the pickup point can be reached most quickly.

We concretize the situation as follows. We are given a transportation network $C = \{s_1, \ldots, s_c\}$ which consists of c isothetic line segments that are only allowed to touch and a set $P = \{p_1, \ldots, p_n\}$ of n point sites in the plane. Movement off the network takes place with unit speed with respect to the Manhattan metric, while a segment s_i can be used to move with some speed $g_i > 1$ into either direction. We require that the number of different speeds is constant. A segment can be accessed and left at any point. According to the Manhattan or L_1-metric, the distance $d_{\text{Manhattan}}$ of two

[*]A preliminary version of this article appeared in [13]. Work supported by the European Commission within FET Open Projects DELIS and by grant WO 758/4-2 of the German Research Foundation (DFG).

[†]Fakultät für Informatik, Universität Karlsruhe, P.O. box 6980, D-76128 Karlsruhe, Germany. Internet: `http://i11www.iti.uni-karlsruhe/group`

[‡]School of Digital Information Engineering, Hankuk University of Foreign Studies, Yongin, Korea. Email: `cssin@hufs.ac.kr`

points $a = (x_a, y_a)$ and $b = (x_b, y_b)$ in the plane is defined as $d_{\text{Manhattan}}(a, b) = |a_x - b_x| + |a_y - b_y|$. Observe that the length $\ell(e)$ of an isothetic line segment $e = \overline{ab}$ is the distance $d_{\text{Manhattan}}(a, b)$ of its endpoints. In this paper we use the following notion of distance, which is based on the Manhattan metric. Let $\Pi = (a = u_1, u_2, \ldots, u_{k-1}, u_k = b)$ be an isothetic path from a to b with vertices $u_1, \ldots, u_k$, then the length ℓ of Π is defined as

$$\ell(\Pi) = \underbrace{\sum_{i=1}^{k-1} \left(\ell(\overline{u_i u_{i+1}}) - \sum_{s_j \in C} \ell(\overline{u_i u_{i+1}}) \cap s_j \right)}_{\text{off the network } C} + \underbrace{\sum_{i=1}^{k-1} \sum_{s_j \in C} \frac{\ell(\overline{u_i u_{i+1}}) \cap s_j}{g_j}}_{\text{on the network } C}.$$

Thus, the length of Π is the sum of the lengths of all parts of Π that are off the transportation network, plus the length of those parts that are on the network, individually weighted by the speed g_i of the corresponding network segment. In other words, the length of the path Π from a to b is the time it takes to walk along Π from a to b, see Figure 1 for an example.

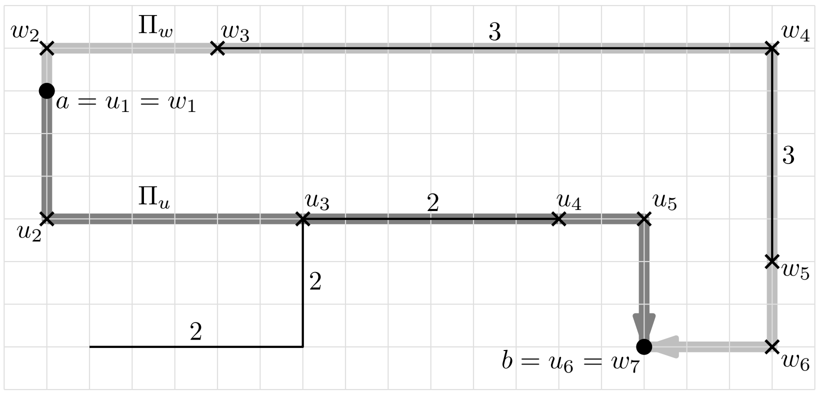


Figure 1: A five-segment transportation network (black lines) and two a–b paths (fat gray lines). Path $\Pi_u = (u_1, \ldots, u_6)$ has length $14+6/2 = 17$, path $\Pi_w = (w_1, \ldots, w_7)$ has length $10+18/3 = 16$.

Now we can define the distance d between two points a and b in the plane to be the temporal length of the quickest path Π between them:

$$d(a, b) = \min_{\Pi \text{ isothetic } a\text{-}b \text{ path}} \ell(\Pi).$$

The definition of quickest paths induces a metric in the plane that we call *city metric* (note that in Figure 1 $d(a, b) = \ell(\Pi_w) = 16$). For this metric we define the *Voronoi region* $\text{reg}(p_i)$ of a site p_i as the set of all points that are closer to p_i than to any other site p_j. We define the *city Voronoi diagram* $V_C(P)$, first described by Aichholzer et. al. [5], as the subdivision of the plane into these Voronoi regions. Given a query point $q \in \mathbb{R}^2$, the site in P closest to q can be determined by point location (see e.g. [9]) in time logarithmic in the complexity of $V_C(P)$. We additionally obtain a refinement $\mathcal{V}_C(P)$ of the city Voronoi diagram, which is a further subdivision of the plane into regions of combinatorically equivalent quickest paths to P. This refinement serves as a *quickest-path map* and can report the quickest path to the closest site in additional time $O(L)$, with L being the path complexity.

In this paper we obtain the following new result. We present a technique to construct the city Voronoi diagram of n sites and c isothetic network segments with a constant number of different speeds in $O((c+n)\log^5(c+n)\log\log(c+n))$ time using $O((c+n)\log^5(c+n))$ storage.

The technique we present for the construction of the city Voronoi diagram can be viewed as an example of a more general approach to solving geometric problems. In a setting that involves objects of high complexity we often have to realize the following three concurrent requirements on our data. First, the objects need to be simplified in a way such that they allow fast handling and processing. This is best accomplished by guaranteeing constant complexity of data objects. Second, this data simplification must not result in a substantial increase in the number of objects. And third, the simplified objects must help to solve the problem on the original data efficiently. Our refinement of the city Voronoi diagram meets these three requirements, as we shall see.

The *trapezoidal decomposition* (see e.g. [9]) is a well-known method for answering point-location queries in the plane that also follows the stated paradigm. A given planar subdivision is augmented by drawing vertical extensions through all vertices. The extensions stop when they meet another edge of the subdivision. Again, this yields a refined planar subdivision with simplified objects of constant complexity. The search structure for this subdivision, a tree, is built by a randomized incremental algorithm. Due to the simple shapes of the trapezoidal regions, the search tree can be built efficiently. The refinement does not increase the complexity of the subdivision asymptotically. And finally, a query point can be located efficiently in the original subdivision via the trapezoidal regions.

This paper is structured as follows. In Section 2 we go through the previous work on city Voronoi diagrams. In Section 3 we analyze the mechanics of the wavefront expansion. Here we also determine the complexity of the diagram and present a lifting into 3-space, where the additional dimension represents the elapsed time. This helps us to apply orthogonal range queries for predicting the next change in the shape of the wavefront. In Section 4 we describe our main contribution, an algorithm that efficiently maintains the shape of the wavefront during the expansion. In Section 5 we put things together. This yields the overall result, the construction of the city Voronoi diagram in $O((c+n)\text{polylog}(c+n))$ time. We conclude the paper with a short discussion and an outlook in Section 6. To get an impression of the city Voronoi diagram of a moderately complicated transportation network and several sites, we refer to Figure 18 at the end of this paper.

2 Previous work

The city metric was introduced by Abellanes et al. [1] (under the name *time metric*) who derived basic properties of quickest-path metrics. Moreover they gave an $O(n\log n)$-time construction algorithm for the city Voronoi diagram $V_C(P)$ for the special case that the transportation network is a single straight line. Hurtado et al. [14] discuss some results for single-line transportation networks under the Euclidean metric. Based on the concept of weighted regions, introduced by Mitchell and Papadimitriou [16], Gewali et al. [11] studied a special case that connects to the city Voronoi diagram. The segments of a given transportation network C can be viewed as one-dimensional instances of weighted regions. The authors construct the quickest path between two points in time $O(c^2)$. Another variant of our setting is the airlift Voronoi diagram, which restricts access to the network to a set of stations. Recently Ostrovsky-Berman [17] presented the first time-optimal algorithm for airlift Voronoi diagrams, running in $O((n+s)\log(n+s)+c)$ time with s being the number of stations. Aichholzer et al. [5] presented an algorithm that constructs the city

Voronoi diagram of n sites and c segments given a uniform network speed in $O(n \log n + c^2 \log c)$ time using $O(c + n)$ space. The resulting data structure, the refined city Voronoi diagram $\mathcal{V}_C(P)$, answers quickest-path queries in $O(L + \log(c + n))$ time. In their algorithm the authors first prepare a set of time-stamped nodes in the grid induced by the segments using the continuous Dijkstra method [15]. Then carefully adapted straight-skeleton figures scheduled at these nodes are computed by employing techniques for the construction of abstract Voronoi diagrams.

Under the Euclidean metric, Abellanas et al. [2] studied shortest paths and Voronoi diagrams again for the special case that the transportation network is a single straight line. Bae and Chwa [6] presented an algorithm that establishes a city Voronoi diagram in the Euclidean plane, admitting arbitrary orientation and speed of network segments. Their technique is similar to the approach of Aichholzer et al. [5] and requires $O(nc^2 \log n + c^3 \log c)$ time and $O(c(c + n))$ space. The authors recently proved that their approach naturally extends to more general metrics including asymmetric convex distances [7].

The two fundamental techniques used in this paper, namely the expansion of a wavefront, tracing out a *straight skeleton* with its vertices [4] and the maintenance of closest pairs in dynamic sets [10], have been employed before, e.g. by Mitchell et al. [15] for solving the discrete geodesic problem and by Agarwal et al. [3] for collision detection in kinetic data structures, respectively.

3 The wavefront expansion

Our algorithm constructs the city Voronoi diagram $V_C(P)$ by simulating the expansion of a wavefront starting at time $t_0 = 0$ at the set P of sites. At time $t \geq 0$ the wavefront is the set of all points whose distance from P is t in the city metric. The key observation is that during the course of the expansion each point of the plane is reached by the quickest possible path starting from P. In order to tell the quickest path from q to P we therefore need to store information about how the wavefront reached q and invert the path taken by the wavefront. This is done as follows. By storing where the wavefronts of different sites merge and by tracing vertices resulting from such mergings, we immediately obtain a subdivision of the plane. A region of this subdivision is the set of points that can be reached most quickly starting at the unique site contained. The borders of this subdivision consist of all points that can be reached equally quickly from at least two sites, thus the subdivision obtained is the Voronoi diagram with respect to the city metric, i.e. the city Voronoi diagram $V_C(P)$, see Figure 2. We can trace the path of other wavefront vertices in order to obtain a refinement of $V_C(P)$. Since the wavefront consists exclusively of vertices and straight line segments (due to the properties of the city metric), this refined city Voronoi diagram $\mathcal{V}_C(P)$ subdivides $V_C(P)$ into regions of uniform wavefront expansion. (Note that the faces, edges, and vertices of $\mathcal{V}_C(P)$ correspond to the edges, vertices and combinatorial changes of the wavefront, respectively.) Thus, if we store for each such region the direction in which the wavefront swept over the region, we can tell for all points of that region how to reach the *oldest* object of this region. This oldest object can either be a vertex or a line segment, being the part of the region that was reached first by the wavefront. By doing this repeatedly, we ultimately reach a point in P, tracing back the expansion of the wavefront. See Figure 3 for an example.

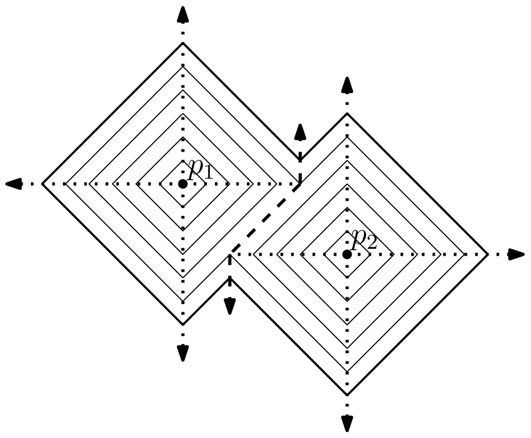


Figure 2: The wavefronts of two sites p_1 and p_2 merge, tracing out the (dashed) boundary of their Voronoi regions.

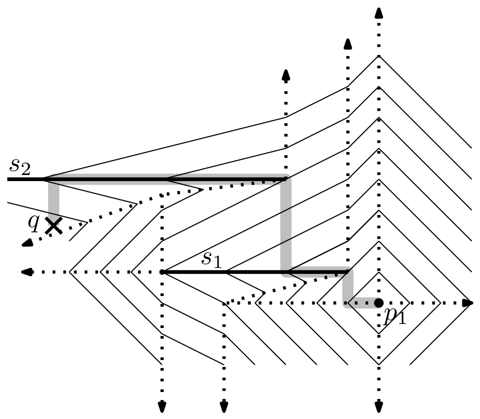


Figure 3: The expansion of the wavefront guides the way from a query point q back to p_1.

3.1 Events

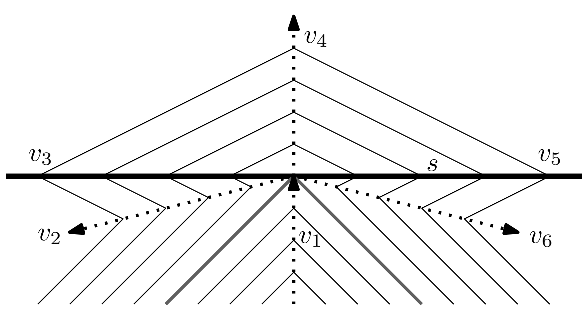
We discretize the continuous expansion of the wavefront at the points in time when a collision between the wavefront and the network or between two parts of the wavefront happens. We call each of these points in time *event*. The combinatorial shape of the wavefront changes at most at such events. An event is a pair of a timestamp and a locus in the plane, which is either a point or a line segment. We distinguish four types of events, depending on the situation. A vertex of the wavefront hitting a segment generates a type-A event, while an edge of the wavefront sliding into a network node triggers a type-B event. A type-C event occurs when a wavefront edge shrinks to zero length and finally, a type-D event is a collision of two parts of the wavefront. See Figures 4 for examples.

It is not hard to see the following:

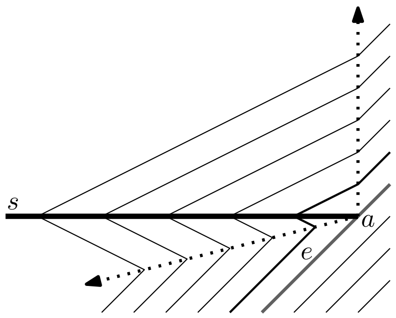
Observation 1 *For any type of event the number of changes in the wavefront is constant.*

3.2 Relevant events

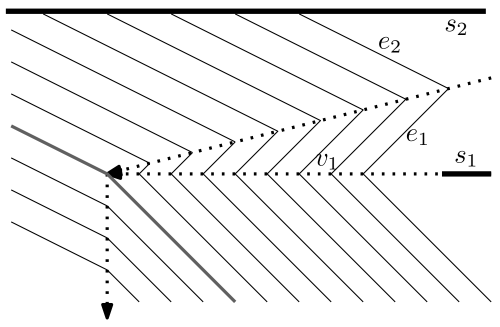
As a consequence of Observation 1, we only need to focus on the detection and on the number of events. An upcoming event can be detected by comparing for all edges and vertices of the wavefront the timestamp of their next collision. This comparison leads us to the notion of *virtual* events. A virtual event is defined by two points of the wavefront, or a point of the wavefront and a point of the transportation network that would collide, given their current movement, but actually do not collide due to the fact that at least one of them is involved in an event that happens earlier. See Figures 5 and 6 for an example. There are also events that do happen, but still do not contribute to the complexity of $V_C(P)$. We call such events *redundant*, see Figure 7 for an example. Events that are neither redundant nor virtual are *relevant* and take part in shaping $V_C(P)$. Next we discuss an important result about the total number of relevant events.



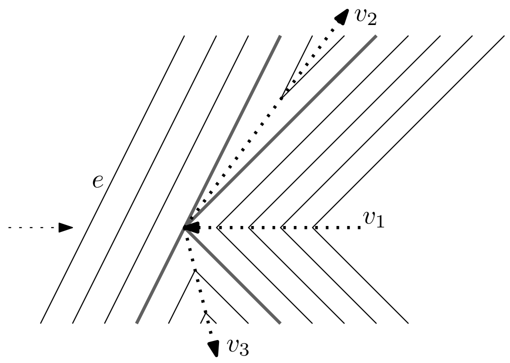
(a) Vertex v_1 hits segment s in a type-A event.



(b) Edge e slides into vertex $v_{\rm C}$ in a type-B event.



(c) Edge e_1 shrinks and causes a type-C event (see e.g. the leftmost portion of Figure 3).



(d) Vertex v_1 and edge e cause a type-D event.

Figure 4: The four different types of events that occur during the wavefront expansion.

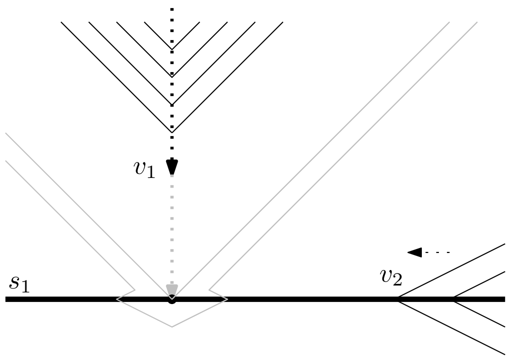


Figure 5: A type-A event involving vertex v_1 and segment s_1 is pending.

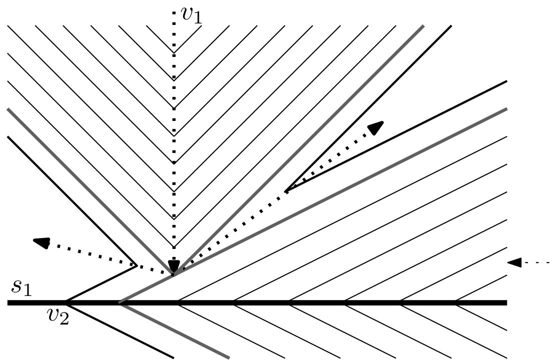


Figure 6: The event has been prevented by another event.

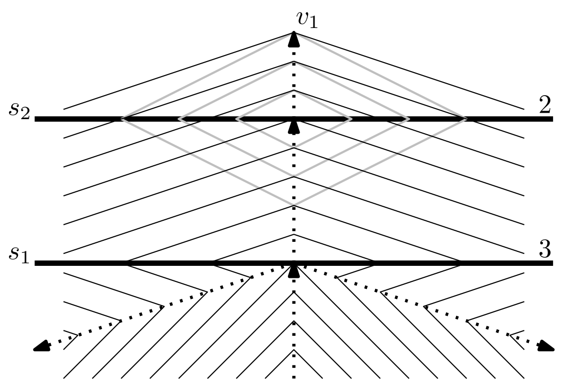


Figure 7: The shape of the wavefront at vertex v_1 after colliding with the segment s_1 makes the subsequent type-A event (gray rhombes) redundant.

3.3 The linear complexity

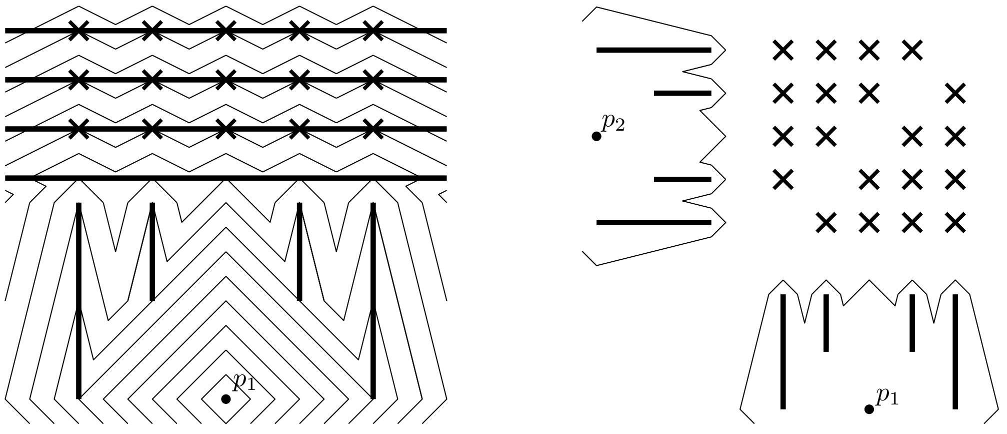
Adapting a result of Aichholzer et al. [5] to a constant number k of network speeds we obtain the following result:

Theorem 1 *The number of relevant events and the complexity of the refined city Voronoi diagram $\mathcal{V}_C(P)$ is $O(k(c+n))$.*

Proof. We build upon Observation 1 and count the number of events that generate additional faces in the region bounded by the wavefront. These are type-A and type-B events (see Figure 4). The number of type-B events is clearly bounded by twice the number of network segments. To estimate the number of type-A events, define a vertex v of the wavefront to be a *peak* if v moves parallel to a coordinate axis, does not move along a segment of the network, and the wavefront is locally contained in a halfplane orthogonal to the direction of movement. Only peaks can cause type-A events. Each point in P generates at most four peaks, each endpoint of a segment in C at most one. No other event generates a new peak. On the other hand a peak can cause at most k non-redundant type-A events, since after colliding with a segment with maximum speed in the network, all further type-A events of this peak will be redundant. See Figure 7 for an example. Thus, the number of faces of $\mathcal{V}_C(P)$ is linear in $k(c+n)$.

As Aichholzer et al. [5] observe, the refined city Voronoi diagram $\mathcal{V}_C(P)$ can be considered the straight skeleton of a set of *figures* and as such is a planar graph whose vertices have degree at least three. Thus, by Euler, the number of vertices and edges of $\mathcal{V}_C(P)$ is linear in the number of faces. As the relevant events of the wavefront are in one-to-one correspondance with the vertices of $\mathcal{V}_C(P)$, the number of events is linear in $k(c+n)$, too. $\square$

As opposed to relevant events, the number of redundant and virtual events can each amount to $\Omega(c^2)$, as can be seen in Figure 8. While these events are easy to identify, we cannot treat them explicitly without a significant increase in the asymptotic running time, since the number of relevant events is only linear in $k(c+n)$. Thus we are left with the task of efficiently detecting the next event while implicitly ignoring non-relevant events. In the next subsection we consider a unifying approach for detecting all four types of events.



(a) Redundant events. (b) Virtual events.

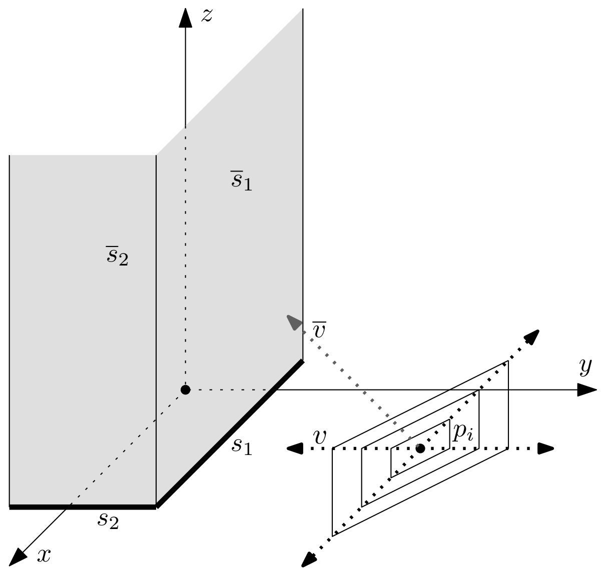
Figure 8: There can be $\Omega(c^2)$ virtual and redundant events (marked by crosses), even if all segments have the same speed (2).

3.4 The wavefront in 3-space

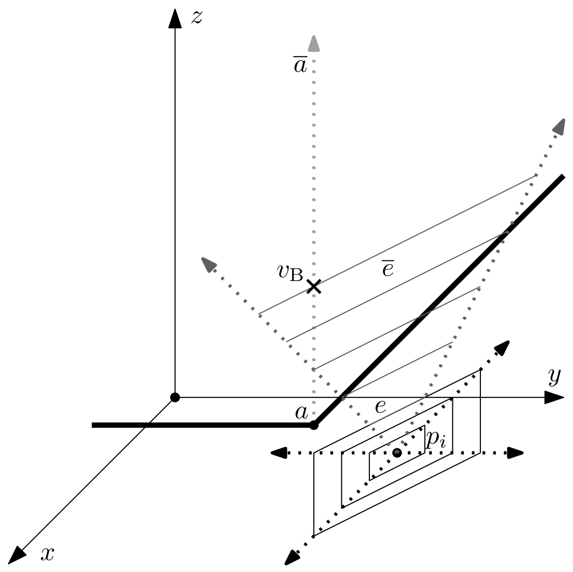
We now add a third dimension to our view of the wavefront expansion, such that a positive z-component represents the time that passed since the start of the wavefront in the x-y-plane. Consequently wavefront vertices and edges trace out rays and polygons, respectively. Note that once the wavefront has reached the last event, the orthogonal projection of these polygons onto the plane yields the regions of the refined city Voronoi diagram. In a similar fashion we extend all network segments to vertically unbounded rectangles and all network nodes to vertical rays. Each ray is defined by its point of origin and by the vector v by which it moves in unit time. The x- and y-components v_x and v_y of this vector correspond to the movement of the corresponding vertex in the original x-y plane. Naturally, the z-component v_z is 1 for all rays. Thus, the speed of a ray $\vec{v} = (v_x, v_y, 1)$ is $|\vec{v}|$, using the Euclidean metric. Note that the vertical rays corresponding to network nodes have speed 1. The z-component of the wavefront expansion thus yields the timestamp of an event by its z-coordinate in space. Figures 9a and 9b show how the z-axis is added in the context of type-B and type-A events, respectively. Analogously, a type-C event, as in Figure 4c, yields a collision of a ray $\overline{v}_1$ and a polygon $\overline{e}_2$ in 3-space, while a type-D event, as in Figure 4d, involves the collision of a ray $\overline{v}_1$ and a polygon $\overline{e}$ in 3-space. Summarizing, for each type of event in space, we observe the following:

Observation 2 *In 3-space any event can be described as a collision between a ray and a polygon.*

Such collisions can be computed using *ray-shooting* techniques, but general methods for ray-shooting have unsatisfactory time bounds. While it is possible to answer general on-line ray-shooting queries among n static arbitrary polyhedra in $\mathbb{R}^3$ in $O(\log n)$ time, a preprocessing time of $O(n^4)$ is required, see [12] for an overview of ray-shooting techniques. Furthermore, to avoid detecting a quadratic number of events, we still need to take care of redundant and virtual events. In the next section we describe how we efficiently detect upcoming events. As from now we will treat the wavefront in 3-space, as described in this section, however, for the sake of readability most figures will depict the projection onto the x-y plane. Since the rays and slabs in 3-space correspond



(a) A type-A event in space, involving ray $\overline{v}$ and polygon $\overline{s}_1$ is imminent.



(b) Ray $\overline{a}$ and polygon $\overline{e}$ collide in a type-B event at point v_B in space.

Figure 9: Examples of events, observed as collisions of a slab and a ray in space.

one-to-one to rays and slabs in 2-space via orthogonal projection, we do not differentiate between them explicitly.

4 Maintaining the next event

Since each event is a collision of a ray and a polygon we can always determine the next event by maintaining the closest pair between these two dynamic sets of objects (polygons and rays). If we can do this quickly and repeatedly, we can efficiently simulate the expansion of the wavefront. In the following we present a hierarchy of event-prediction mechanisms, culminating in the prediction of the next relevant event of the expansion.

4.1 The global prediction

Eppstein and Erickson [10] proposed a method of maintaining the closest pair among two dynamic sets R and B of objects according to a given distance measure $d : R \times B \to \mathbb{R}_0^+$ that can be computed in constant time. Both sets are dynamic in that they are subject to insertions and deletions. As a prerequisite the sets R and B need to support *minimization queries*, i.e. for any object $b \in B$ an object $r \in R$ minimizing $d(r, b)$ can be determined and vice versa. In our application R and B will be partially unbounded polygons and tips of rays, respectively. We use the following result:

Theorem 2 ([10]) *Suppose that after $P(n)$ preprocessing time, we can maintain a data structure of size $P(n)$ that supports insertions, deletions, and minimization queries, each in amortized time $T(n)$. Then after $O(P(n) + nT(n))$ preprocessing time, we can maintain the closest pair between R and B in $O(P(n))$ space, $O(T(n) \log n)$ amortized insertion time, and $O(T(n) \log^2 n)$ amortized deletion time.*

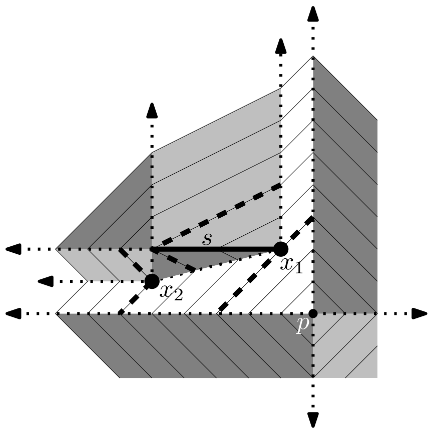
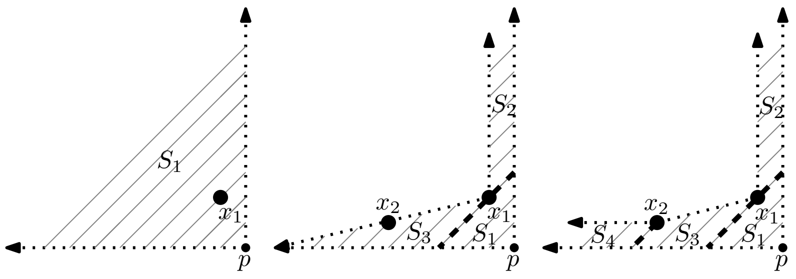


Figure 10: Division (dashed) of regions (shaded) into slabs.



(a) Slab S_1 is created at p and is involved in event x_1.

(b) At x_1 slab S_1 becomes inactive. Slabs S_2 and S_3 are created.

(c) At x_2 slab S_3 becomes inactive. Slab S_4 is created.

Figure 11: The white region of Figure 10 ist split into slabs during the wavefront expansion at events x_1 and x_2.

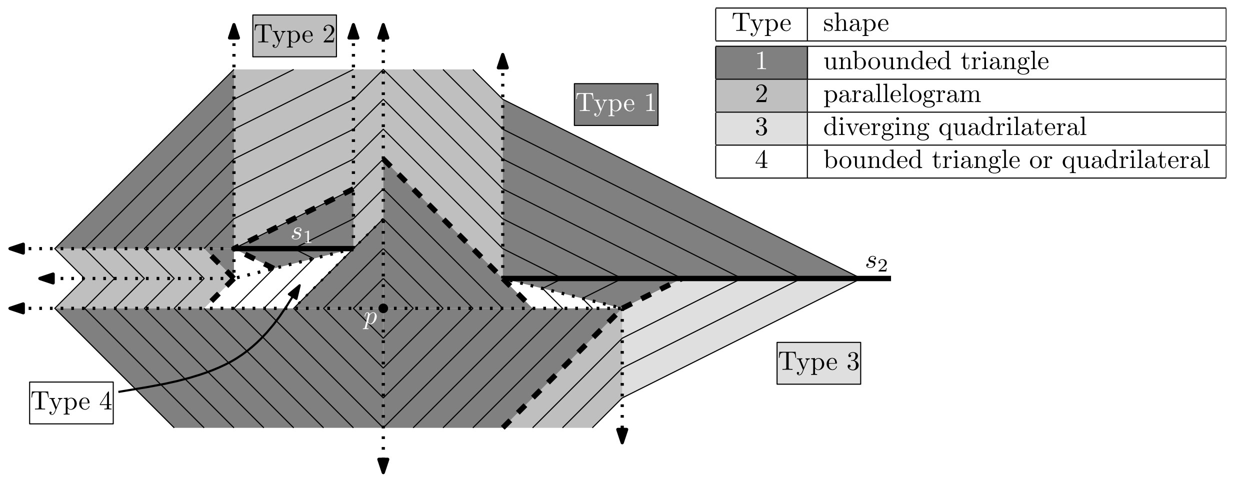
Employing this theorem we are left with the lesser problem of efficiently performing minimization queries. If we consider the requirements of the theorem in our situation, we need to determine for any given ray (i.e. moving vertex) the region (of $\mathcal{V}_C(P)$) it hits next and the inverse, for any given region A the ray that hits A next. These two well-known types of queries are ray-shooting queries and *lowest-intersection* queries, respectively. Let us call the results of such queries *local* predictions and the closest pair in the sense of the above theorem the *global* prediction. We now face the challenge of simplifying our data such that we can implement fast minimization queries while taking implicit care of non-relevant events.

4.2 Simplification of wavefront data

The data we deal with for the purpose of local predictions comprises rather complicated, potentially unbounded polygons in 3-space. Recall that the orthogonal projection of these polygons onto the plane yields the regions of the refined city Voronoi diagram, thus we call these polygons regions. If we split these regions along the current wavefront, as depicted in Figure 10 projected onto the plane, namely each time the region hits the locus of an event, we obtain *slabs* that are possibly unbounded triangles or quadrilaterals in 3-space. The key observation is that all slabs have constant complexity.

For an example, observe the white region in Figure 10. This region of the refined city Voronoi diagram has high complexity, since after its creation at site p, it is involved in two subsequent events at x_1 and at x_2. As depicted in Figure 11 we subdivide the white region during the wavefront expansion at these events along the current wavefront in order to obtain slabs of low complexity.

As long as a slab has not yet been involved in an event (except for the one that created the slab) we call it *active*. Analogously we define active rays. If an active slab is involved in a second event it becomes *inactive* and gets bounded by the current wavefront. This changes the shape of all slabs involved, except for already bounded triangular slabs, which merely become inactive. Then, depending on the type of event, new slabs are spawned. These new slabs either continue the slabs bounded by the event and are thus still contained in the corresponding region of the refined city Voronoi diagram, or create a new region. For example, in Figures 10 and 11 two new slabs (light gray and dark gray) and two continuing slabs (S_2 and S_3) are created at event x_1. Note



Type	shape
1	unbounded triangle
2	parallelogram
3	diverging quadrilateral
4	bounded triangle or quadrilateral

Figure 12: The four types (color-coded) of slabs that occur. Dashed lines indicate where regions of $\mathcal{V}_C(P)$ (boundaries dotted) are divided into slabs along the current wavefront.

that inactive slabs cannot take part in a relevant event. They have already been swept over by the wavefront and thus we exclude them from further event detection. For example, in Figure 10 after both events have occured, only slabs S_2 and S_4 are still active. Neither S_3 nor S_1 can be involved in any future event.

We distinguish four types of slabs depending on the relative direction of bounding rays (diverging or converging) and on the number of bounding edges (three or four). For the definition of the four types of slabs see Figure 12. Note that a slab must belong to one of these types since a slab is both created and terminated by the wavefront, and not involved in an intermediate event. Thus a slab is either a triangle or a quadrilateral of some sort, all of which are covered by the four types.

By Theorem 1 the total number of relevant events that occur during the wavefront expansion is linear in $k(c+n)$. Since by Observation 1 each event causes a constant number of changes in the wavefront, we obtain the following corollary:

Corollary 1 *Subdividing the refined city Voronoi diagram $\mathcal{V}_C(P)$ into slabs yields a partition of complexity $O(k(c+n))$.*

We are now left with answering minimization queries for a linear number of triangles and unbounded quadrilaterals versus rays.

4.3 Ignoring virtual events

The following two lemmas guarantee that only relevant events or redundant events can possibly be globally predicted. Redundant events will be dealt with in Section 4.6.

Lemma 1 *A slab and a ray either miss each other, collide in an event or define a virtual event.*

Proof. Active slabs potentially cover areas beyond the current wavefront. The key observation is that the slabs are designed to cover only those points of the plane that they would cover in the final diagram, if they are not made inactive prematurely by some event involving them. The same holds for rays. Thus, if a slab and a ray intersect, this either actually causes an event during the

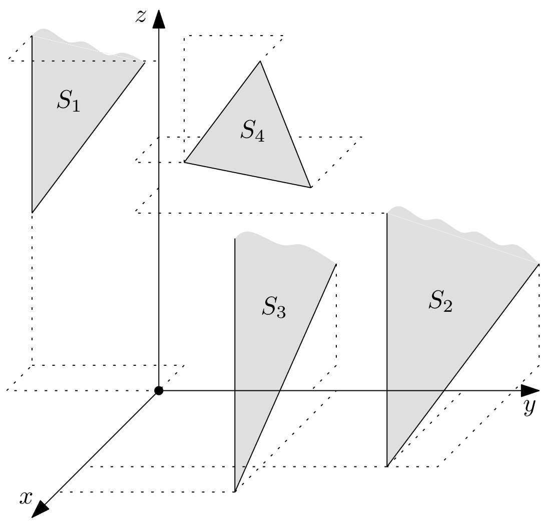


Figure 13: Among these slabs only S_1 and S_2 are similar. Slab S_3 encloses a smaller angle and S_4 is a bounded triangle, thus neither is similar to any other.

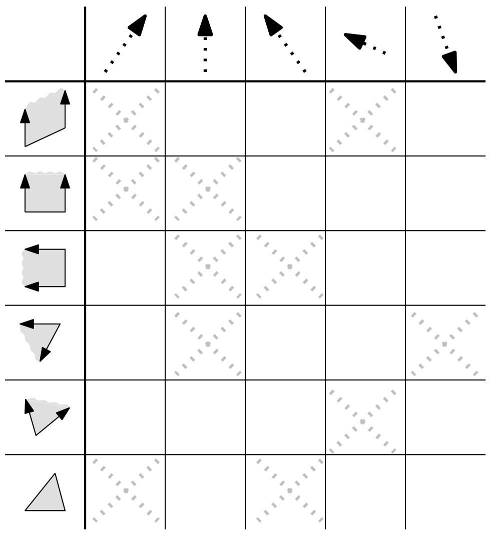


Figure 14: In principle each combination of a slab and a ray class defines a sublocal data structure. However, some combinations are ignored (crosses) by arguments of Section 4.6.

wavefront expansion, or is prevented by another event. Each collision that does not correspond to a relevant or a redundant event is anticipated by another event. □

The importance of the design of the slabs becomes obvious if we observe that the consequence of an overly coarse simplification of the wavefront data would be collisions outside the scope of events and virtual events. Thus, Lemma 1 would not hold, and such events would have to be dealt with explicitly. Lemma 2 shows that we do not even have to identify virtual events explicitly.

Lemma 2 *Virtual events are never predicted globally.*

Proof. The global prediction maintains the next collision between a ray and a slab. By Lemma 1 this constitutes either an event or a virtual event. If it were virtual, then by the definition of a virtual event, it would not be the next collision to happen. Thus, virtual events are never globally predicted. □

4.4 Orthogonalized sublocal queries

We now define slabs to be *similar* if their sides pairwise enclose the same angle with the x-axis. For an example see Figure 13. Rays are similar if they merely point in the same direction and move at the same speed. These definitions at hand, the following holds for classes of similarity of slabs and of rays:

Lemma 3 *The number of classes of similarity of slabs and of rays is constant.*

The proof of Lemma 3 builds upon the fact that the expansion of a wavefront edge can only be accelerated, i.e. altered from the simple Manhattan metric expansion, by a single network segment.

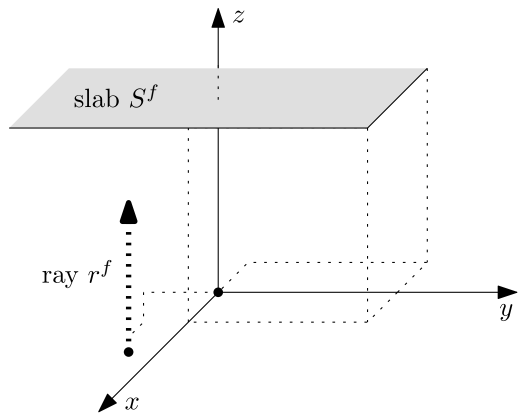


Figure 15: A transformed slab-ray pair in a sublocal data structure.

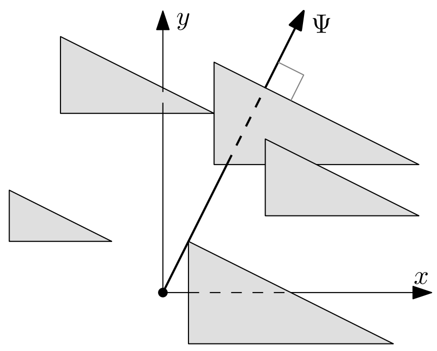


Figure 16: An additional axis (Ψ-axis) is introduced for range searching.

Since the number of different speeds and angular positions of segments is constant, the wavefront can expand only with a constant number of orientations and speeds. Since vertices of the wavefront are the intersection points of two edges, the same holds for wavefront vertices.

Let us now consider an arbitrary combination of one class of slabs with one class of rays. Figure 14 shows an exemplary sketch of a few such combinations. We call the result of a minimization query involving all objects of exactly these two classes a *sublocal* prediction. Since by Lemma 3 the number of ray and slab classes is constant, the number of pairs of ray and slab classes is constant, too. Therefore, any local prediction can easily be computed from sublocal predictions in constant time. Within a sublocal data structure considerable simplifications are possible. For each such data structure we can define a coordinate transformation f consisting of at most one rotation and four concatenated shearings. First the rotation aligns the rays with the z-axis and one side of the slabs with the x-axis. Then step by step each side of the slabs is orthogonalized to two of the three axes. As shown in Figure 15, we end up with simple orthogonal range queries instead of ray-shooting or lowest-intersection queries.

Observation 3 *Sublocal data structures can be implemented as multi-dimensional orthogonal range-query data structures.*

Since each sublocal data structure contains only similar rays and slabs, we can define two axes of the coordinate system to be parallel to slab edges, a third axis to be parallel to the rays. The same can be achieved by applying the abovementioned rotation and shearings to the data in a fixed coordinate system. Note that in order to orthogonalize slabs of types 3 and 4 (see Figure 12), we need to introduce an additional axis (see Figure 16), adding one more level to the range searching data structure, since these slabs comprise three pairwise non-parallel edges. Summarizing, we observe that the range-searching data structures are at most four-dimensional, see for example [9] or [12] for an overview of such data structures.

4.5 Feeding the global prediction

As stated earlier the global prediction relies on local predictions. Local predictions in turn are based on a constant number of sublocal queries, each being answered with a multi-dimensional orthogonal range query. The general picture of our algorithm is given in Figure 17. Making use of Theorem 1 and of well-known results about multi-level range trees and fractional cascading (see e.g. [9]) we can state the following (treating the number k of different network speeds as a constant).

Observation 4 *After an $O((n+c)\log^3(n+c))$-time preprocessing our sublocal data structures can each handle insertions, deletions and queries in $O(\log^3(c+n)\log\log(c+n))$ time using $O((c+n)\log^3(c+n))$ total space. The same holds for local data structures.*

Similar results for ray shooting in a fixed direction among general k-oriented polyhedra are obtained in [8].

Comparing slabs with rays we observe that while slabs are static, rays are not. Thus a ray cannot simply be represented by its static foot point p_{foot}. In order to do justice to the dynamic nature of rays we should in fact use the (moving) tip of the rays. However, instead of repeatedly advancing the tips of all rays we can simply apply a time correction when inserting the foot points into our lowest-intersection data structures and when querying our ray-shooting data structures: We set $p_{\text{foot}}^{\text{new}} := p_{\text{foot}} - (0,0,t)|\vec{v}|$, with t being the time elapsed since the start of the wavefront at $t_0 = 0$ and $\vec{v}$ being the direction of the ray. The key observation is that inside each individual sublocal data structure these modified foot points represent at all times the *relative* position of the tips of their rays, once the transformation f has been applied. Thus, time correction ensures that events are reported in the proper temporal order.

4.6 Ignoring redundant events

Since the statement of Observation 4 is applied to relevant events, all that is left to deal with after Lemmas 1 and 2 are redundant events. We can show that due to the careful design of our slabs we do not need to invest any time ignoring redundant events. As indicated in Figure 7, a redundant event is due to a wavefront vertex hitting a segment with equal or lower speed than segments hit by the same wavefront vertex earlier. If we simply refrain from forwarding local queries to sublocal data structures designed for segments with equal or lower speed, we implicitly ignore all redundant events. The crossed-out entries in Figure 14 illustrate this. This finally yields the following lemma:

Lemma 4 *The total number of globally predicted events is $O(k(c+n))$.*

5 Main result

The structure of our algorithm is shown in Figure 17, and in the previous sections we have described all its vital data structures and procedures. We now state our main result:

Theorem 3 *Given an isothetic transportation network C with c disjoint isothetic segments, a constant number of different speeds on these segments and a set P of n sites, the refined city Voronoi diagram can be computed in $O((c+n)\log^5(c+n)\log\log(c+n))$ time using $O((c+n)\log^5(c+n))$ storage. The refined city Voronoi diagram answers queries asking for the quickest path to S in $O(L+\log(c+n))$ time, where L is the complexity of the path.*

Proof. Recall that we assume that the number k of different network speeds is constant. According to Lemma 4, $O(c+n)$ events are treated. Using Observation 4, Theorem 2 allows us to predict each event in $O(\log^5(c+n)\log\log(c+n))$ time. Hence, the total time used for the global prediction

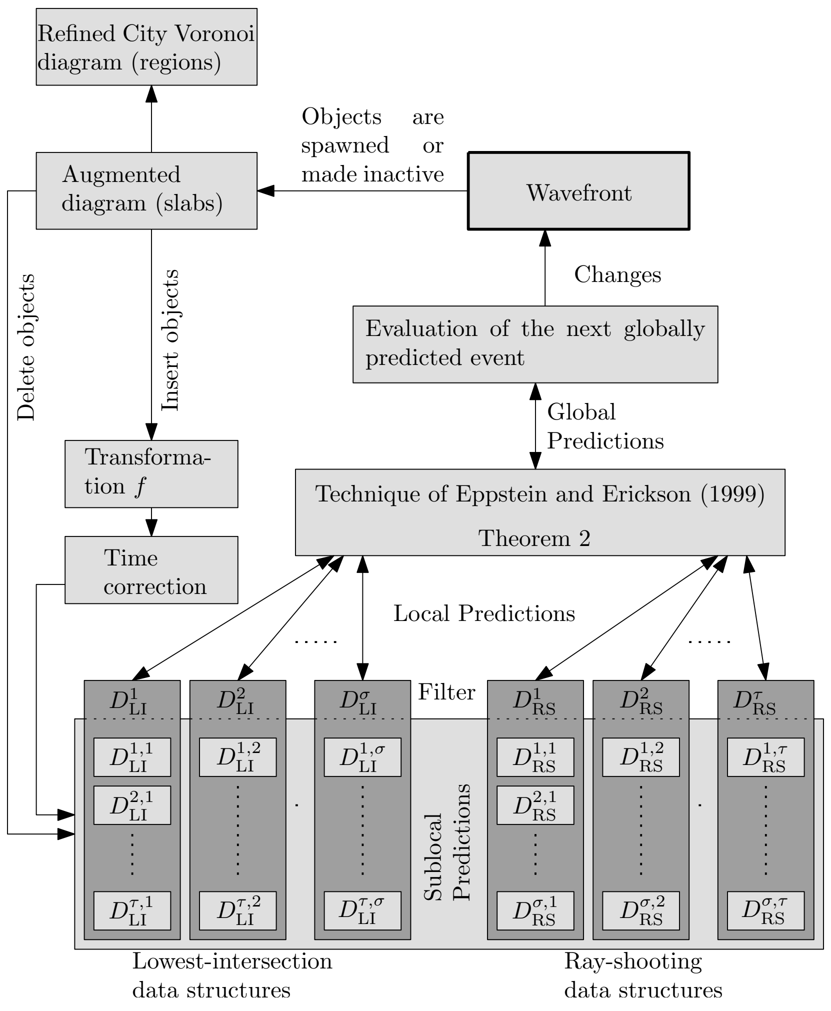

Refined City Voronoi diagram (regions)
Objects are spawned or made inactive
Augmented diagram (slabs)
Wavefront
Changes
Delete objects
Insert objects
Evaluation of the next globally predicted event
Global Predictions
Transformation f
Technique of Eppstein and Erickson (1999)
Theorem 2
Time correction
Local Predictions
Filter
D^1_{LI} D^2_{LI} D^σ_{LI} D^1_{RS} D^2_{RS} D^τ_{RS}
$D^{1,1}_{\mathrm{LI}}$ $D^{2,1}_{\mathrm{LI}}$ $D^{\tau,1}_{\mathrm{LI}}$ $D^{1,2}_{\mathrm{LI}}$ $D^{\tau,2}_{\mathrm{LI}}$ $D^{1,\sigma}_{\mathrm{LI}}$ $D^{\tau,\sigma}_{\mathrm{LI}}$
Sublocal Predictions
$D^{1,1}_{\mathrm{RS}}$ $D^{2,1}_{\mathrm{RS}}$ $D^{\sigma,1}_{\mathrm{RS}}$ $D^{1,2}_{\mathrm{RS}}$ $D^{\sigma,2}_{\mathrm{RS}}$ $D^{1,\tau}_{\mathrm{RS}}$ $D^{\sigma,\tau}_{\mathrm{RS}}$
Lowest-intersection data structures
Ray-shooting data structures

Figure 17: Sketch of the construction algorithm.

is $O((c+n)\log^5(c+n)\log\log(c+n))$, which dominates the time needed to handle events (see Observation 1 and Lemma 4). Observation 4 and thus Theorem 2 require $O((c+n)\log^3(c+n))$ space. The preprocessing time of $O((c+n)\log^3(c+n)\log\log(c+n))$ for the global prediction dominates the preprocessing times of the other data structures (see Observation 4). The total preprocessing time in turn is dominated by the running time of the global prediction. □

6 Conclusion and open problems

In this paper we have presented an algorithm for the construction of the refined city Voronoi diagram. By carefully simplifying the data objects involved, we were able to employ fast techniques for the simulation of a wavefront expansion. On the other hand, the space consumption of our simplified data and the data structures is only slightly superlinear. While our algorithm runs in subquadratic time it relies heavily on certain constraints in the setup. In particular these are the constant number of different network speeds and the isothetic network layout. Our results can be generalized to arbitrarily oriented network segments and to weighted sites as long as the number of different orientations and weights is constant. However, it is a challenge to find a general solution that runs in subquadratic time and can handle arbitrary network speeds, site weights, and segment orientations.

Acknowledgments

We thank Jae-Hoon Kim for discussions about the city Voronoi diagram.

References

[1] Manuel Abellanas, Ferran Hurtado, Christian Icking, Rolf Klein, Elmar Langetepe, Lihong Ma, Belén Palop del Río, and Vera Sácristan. Proximity problems for time metrics induced by the L_1 metric and isothetic networks. In *Actas de los IX Encuentros de Geometría Computacional*, pages 175–182, Girona, 2001.

[2] Manuel Abellanas, Ferran Hurtado, Christian Icking, Rolf Klein, Elmar Langetepe, Lihong Ma, Belén Palop del Río, and Vera Sácristan. Voronoi diagram for services neighboring a highway. *Information Processing Letters*, 86:283–288, 2003.

[3] Pankaj Kumar Agarwal, Julien Basch, Leonidas J. Guibas, John E. Hershberger, and Li Zhang. Deformable free space tilings for kinetic collision detection. *Int. J. Robotics Research*, 21(3):179–197, 2002.

[4] Oswin Aichholzer, David Alberts, Franz Aurenhammer, and Bernd Gärtner. A novel type of skeleton for polygons. *Journal of Universal Computer Science*, 1(12):752–761, 1995.

[5] Oswin Aichholzer, Franz Aurenhammer, and Belén Palop del Río. Quickest paths, straight skeletons, and the city Voronoi diagram. *Discrete & Computational Geometry*, 31(1):17–35, 2004.

[6] Sang Won Bae and Kyung-Yong Chwa. Voronoi diagrams with a transportation network on the Euclidean plane. In Rudolf Fleischer and Gerhard Trippen, editors, *Proc. 15th International Symposium on Algorithms and Computation (ISAAC'04)*, volume 3341 of *Lecture Notes in Computer Science*, pages 101–112. Springer-Verlag, 2004.

[7] Sang Won Bae and Kyung-Yong Chwa. Shortest paths and Voronoi diagrams with transportation networks under general distances. In *Proc. 16th Annu. Internat. Sympos. Algorithms Comput. (ISAAC'05)*, volume 3827 of *Lecture Notes in Computer Science*, pages 1007–1018. Springer-Verlag, 2005.

[8] Mark de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal*, volume 703 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[9] Mark de Berg, Mark van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry*. Springer-Verlag, 2nd edition, 2001.

[10] David Eppstein and Jeffrey Gordon Erickson. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. *Discrete Computational Geometry*, 22(4):569–592, 1999.

[11] Laxmi P. Gewali, Alex C. Meng, Joseph S.B. Mitchell, and Simeon Ntafos. Path planning in 0/1/infinity weighted regions with applications. *INFORMS Journal on Computing*, 2(3):253–272, 1990.

[12] Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of discrete and computational geometry.* CRC Press, 2nd edition, 2004.

[13] Robert Görke and Alexander Wolff. Constructing the city Voronoi diagram faster. In *Proc. 2nd Int. Symp. on Voronoi Diagrams in Science and Engineering (VD'05)*, pages 162–172, Seoul, 10–13 October 2005.

[14] Ferran Hurtado, Belén Palop del Río, and Vera Sacristán. Diagramas de Voronoi con fonciones temporales. In *Actas de los VIII Encuentros de Geometría Computacional*, pages 279–287, Jaume, 1999.

[15] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–667, 1987.

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

[17] Yaron Ostrovsky-Berman. Computing transportation Voronoi diagrams in optimal time. In *Proc. 21st European Workshop on Computational Geometry (EWCG'05)*, pages 159–162, Eindhoven, 9–11 March 2005.

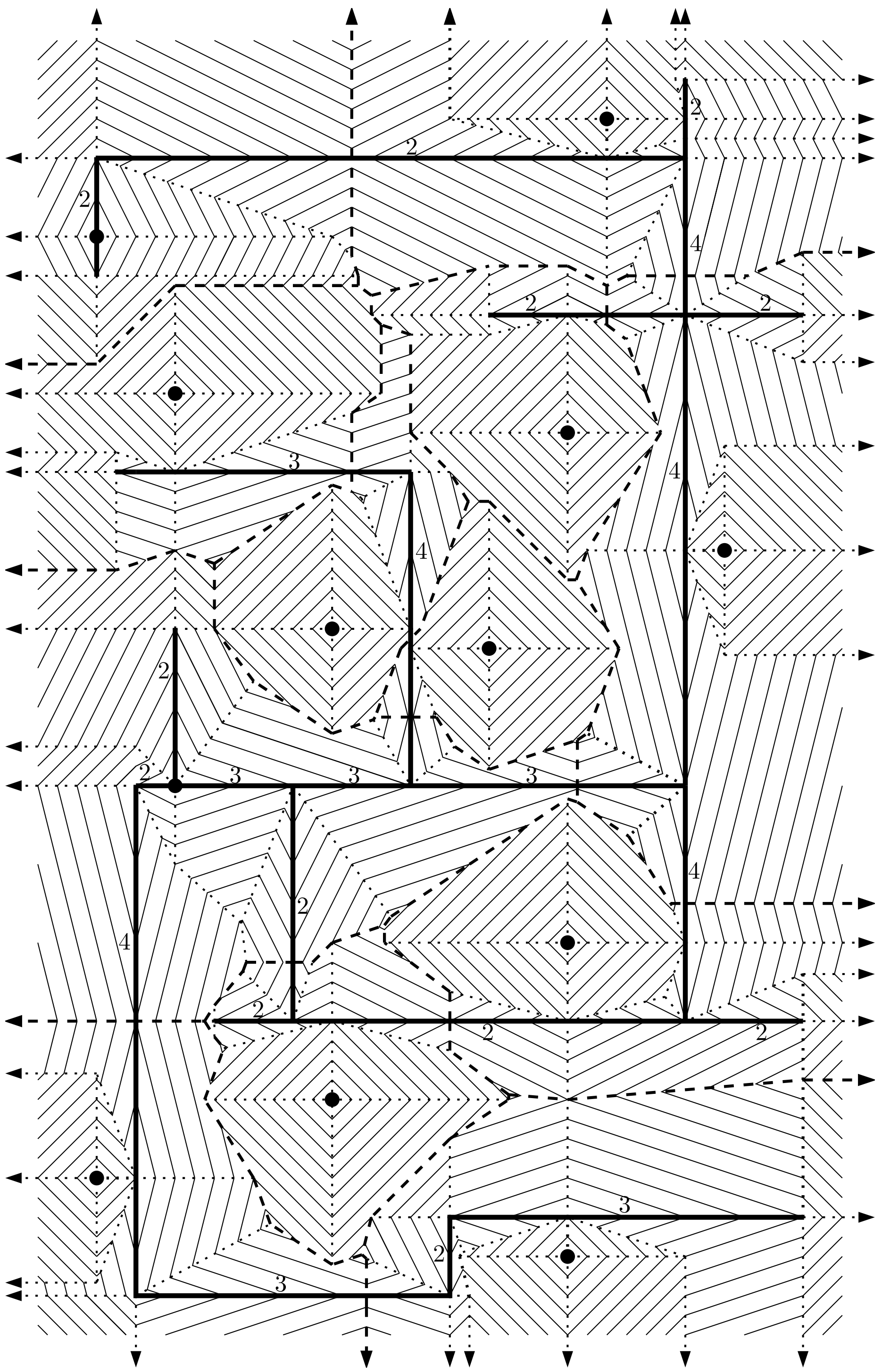


Figure 18: The city Voronoi diagram $V_C(P)$ (dashed) of a complex transportation network C (bold) and a set P of twelve sites (disks). The individual speeds of the network segments are given next to the segments. Snaphots of the expanding wavefront are given by thin solid lines, while the additional edges of the refined diagram $\mathcal{V}_C(P)$, being the traces of wavefront vertices, are indicated by dotted lines.