Geometric Searching in Walkthrough Animations with Weak Spanners in Real Time*

Matthias Fischer Tamás Lukovszki Martin Ziegler

Department of Computer Science and Heinz Nixdorf Institute, University of Paderborn
D-33102 Paderborn, Germany,
{mafi,talu,ziegler}@uni-paderborn.de

Abstract

We study algorithmic aspects in the management of geometric scenes in interactive walkthrough animations. We consider arbitrarily large scenes consisting of unit size balls. For a smooth navigation in the scene we have to fulfill hard real time requirements. Therefore, we need algorithms whose running time is independent of the total number of objects in the scene and that use as small space as possible. In this work we focus on one of the basic operations in our walkthrough system: reporting the objects around the visitor within a certain distance.

Previously a randomized data structure was presented that supports reporting the balls around the visitor in an output sensitive time and allows insertion and deletion of objects nearly as fast as searching. These results were achieved by exploiting the fact that the visitor moves "slowly" through the scene. A serious disadvantage of the aforementioned data structure is a big space overhead and the use of randomization.

Our first result is a construction of weak spanners that leads to an improvement of the space requirement of the previously known data structures. Then we develop a deterministic data structure for the searching problem in which insertion of objects are allowed. Our incremental data structure supports $O(1+k)$ reporting time, where k is a certain quantity close to the number of reported objects. The insertion time is similar to the reporting time and the space is linear to the total number of objects.

1 Introduction

A walkthrough animation is a simulation and visualization of a three-dimensional scene. The visitor of such a scene can see a part of the scene on the screen or a special output device. By changing the orientation of the camera she can walk to an arbitrary position of the scene. In todays computer systems, the scene is modeled by many triangles. The triangles are given by the three-dimensional position of their points. For every position of the visitor the computer has to compute a view of the scene. It has to eliminate hidden triangles (hidden surface removal), to compute the color and brightness of the triangles (the objects resp.), and so on. This process is called rendering.

*Partially supported by EU ESPRIT Long Term Research Project 20244 (ALCOM-IT), DFG Graduiertenkolleg "Parallele Rechnernetzwerke in der Produktionstechnik" Me872/4-1, and DFG Research Cluster "Efficient Algorithms for Discrete Problems and their Applications" Me872/7-1.

For a smooth animation we have hard *real time* requirements. The computer has to render at least 20 pictures (frames) per second. If the animation is computed with less than 20 frames per second, navigation in the scenes is hard or impossible. The time for the rendering of a picture depends on the complexity of the scene, i.e., the number of triangles and the number of pixels which are needed for drawing a triangle. Therefore the graphic workstation cannot guarantee the 20 frames per second for large geometric scenes. In order to control this situation real time and approximation algorithms are necessary to reduce the complexity of those parts of the scene, which are far away and thus have a low influence on the quality of the rendered image.

Our goal is to develop real time algorithms for managing large and dynamic geometric scenes. Our scene is *dynamic* in the sense that a visitor can insert and/or delete objects. We are interested in theoretical and experimental aspects of the problem. The basis for our considerations is an abstract modeling of the problem introduced by Fischer et al. [12]. One of the most important problems to be solved in the walkthrough system is the *search problem*: In order to guarantee the real time behavior of our algorithms it is important that the time for the search, insertion, and deletion of objects is independent of the scene size.

This work is focused on the search problem. It is motivated by the fact that the visitor can only see a relatively small piece of the scene. Only the objects appearing from the position of the visitor in an angle of at least a fixed constant α are potentially visible. Our goal is to develop data structures that support the selection of the objects potentially visible for the visitor. We give efficient solutions to the following problems.

The static searching problem: We assume that our scene consists of n unit size balls. In this case the position of the visitor q and the angle α define a circle in the plane or an d-dimensional sphere in the d-dimensional Euclidean space E^d. All objects in this sphere are potentially visible and all objects outside are not.

Representing the objects by points of E^d we have to solve a *circular range searching problem*, in which we are given a set S of n points in E^d. For a query $query(q, r)$ we have to report the points of S in the interior of the sphere with center q and radius r. We want to develop data structures with $O(n)$ space requirement which support the reporting of these points efficiently. Our goal is that, after a certain point location for q, the reporting takes time not much larger than the output size. In addition we want to use as little space as possible.

The searching problem with moving visitor: We assume that the visitor moves slowly through the scene. We say that the visitor moves *slowly* if the quotient $\frac{\delta}{x}$ is a constant, where δ is the maximum distance between two consecutive query positions, and x is the distance between a *closest pair* of S. We utilize the slow motion of the visitor in order to obtain $O(1)$ time for the point location for the query positions.

The dynamic searching problem: In the dynamic case the visitor can insert or delete an object at her current position. The problem is called *fully dynamic* if insertion and deletion are allowed; *incremental* if only insertion; and *decremental* if only deletion is allowed. Our goal is a linear space data structure with the same query time as in the static case and update time similar to the query time.

1.1 Related problems, state of the art

The objects potentially visible from the visitor are lying in the interior of a sphere whose center is the visitor's position q and radius r is the distance from which an object is in an

angle exactly α visible, i.e., $r = \frac{w}{2\sin(\alpha/2)}$.

In the circular range searching problem we are given a set S of n points in E^d. A query has the form $query(q, r)$, and we have to report the points of S which are lying in the sphere with the center q and radius r. Representing the objects of the scene by points in E^d, we can report the potentially visible objects by solving a circular range searching problem. Furthermore, the results about *nearest neighbor queries* and *graph spanners* are of particular interest.

Data structures for circular range searching in the plane: For an overview of different kinds of range searching problems we refer to the survey article of Agarwal and Erickson [1]. Time optimal solutions of the 2-dimensional circular range searching problem use *higher order Voronoi diagrams*. Bentley and Maurer [4] presented a technique which extracts the points of the Voronoi cell containing q in the kth order Voronoi diagram of S for $k = 2^0, 2^1, 2^2, ...$ consecutively. It stops, if a point in the kth order Voronoi cell of q is found whose distance from q is greater than r, or all the n points are extracted. An $O(\log n \log\log n + t)$ query time is obtained, where t is the number of the points of S lying in the query disc. The space requirement of the data structures is $O(n^3)$. Chazelle et al. [8] improved the query time to $O(\log n + t)$ and the space requirement to $O(n(\log n \log\log n)^2)$ by the aid of the algorithmic concept *filtering search*. Aggarwal et al. [2] reduced the space requirement with *compacting* techniques to $O(n \log n)$. This method results an optimal query time for the circular range searching problem, but it has a superlinear space requirement already in the two dimensional case.

Nearest neighbor queries: Finding the *nearest neighbor* of a query point q in a set S of n points is one of the oldest problems in Computation Geometry. As we will see, the solution of this and related problems are very important to answer the range queries in our walkthrough problem. The following results show that it is not possible to get a query time independent of n without any restrictions on this problem.

Finding the *nearest neighbor* of a query point q in S has an $\Omega(\log n)$ lower bound in the so called *algebraic computation tree* model (c.f. [14]). The planar version of this problem has been solved optimally with $O(\log n)$ query time and $O(n)$ space. But in higher dimensions no data structure of size $O(n \log^{O(1)} n)$ is known that answers queries in polylogarithmic time. The *approximate nearest neighbor* problem, i.e., finding a point $p \in S$ to the query point q such that $dist(q, p) \leq (1 + \epsilon) dist(q, q^*)$, where $q* \in S$ is the exact nearest neighbor of q, was solved optimally by Arya et al. [3]. They give a data structure in dimension $d \geq 2$ of size $O(n)$ that answers queries in $O(\log n)$ time and can be built in $O(n \log n)$ time. (The constant factor in the query time depends on ϵ.) For proximity problems on point sets in $\mathbb{R}^d$ a comprehensive overview is given by Smid [17].

Spanners: Spanners were introduced to computational geometry by Chew [10]. Let $f > 1$ be any real constant. Let the *weight* of an edge (p, q) be the Euclidean distance between p and q and let the *weight* of a path be the sum of the weights of its edges. A graph with vertex set $S \subset E^d$ is called a spanner for S with *stretch factor* f, or an f-spanner for S, if for every pair $p, q \in S$ there is a path in the graph between p and q of weight at most f times the Euclidean distance between p and q. We are interested in spanners with $O(n)$ edges. By aid of an f-spanner of S we can answer a circular range query $query(q, r)$ by finding a "near" neighbor and performing a *breadth first search (BFS)* on the edges of the spanner. The BFS procedure visits all the edges having at least one endpoint not farer from q than fr. (The problem how we can find an appropriate near neighbor will be discussed later.)

Chen et al. [9] proved that the problem of constructing any f-spanner for $f > 1$ has an

$\Omega(n \log n)$ lower bound in the algebraic computation tree model. The first optimal $O(n \log n)$ time algorithm for constructing an f-spanner on a set of n points in E^d for any constant $f > 1$, were done by Vaidya [18] and Salowe [16]. Their algorithms use a hierarchical subdivision of E^d. Callahan and Kosaraju [7] gave a similar algorithm that constructs an f-spanner based on a special hierarchical subdivision and showed that the edges can be directed such that the spanner has bounded out-degree.

Weak spanners: Fischer et al. [12] presented a fully dynamic data structure for real time management of large geometric scenes. They use a property of a certain dense graph weaker than the spanner property to answer circular range queries performing a point location and a BFS. We call this property *weak spanner* property (defined below).

The basic data structure in [12] for the searching problem is the graph $G_\gamma(S)$ called the γ-angle graph for S. (The same construction was called Θ-*graph* by Keil and Gutwin [13] and Ruppert and Seidel [15].) Fischer et al. [12] proved that for $\gamma \leq \frac{\pi}{3}$ the graph $G_\gamma(S)$ is a weak spanner. They applied $G_\gamma(S)$ and perfect hashing to provide a randomized solution for the "moving visitor searching problem" and the dynamic searching problem. A brief description of the data structures in [12] contained in the Appendix.

1.2 New results

Consider a point set $S \subset E^d$ of n points. A (directed) graph G with vertex set S is a *weak spanner* for S with stretch factor f, if, for any two points $p, q \in S$, there is a (directed) path P from p to q in G such that, for each point $x \in P$, the Euclidean distance $dist(p, x)$ between p and x is at most $f \cdot dist(p, q)$. (Note that each weak spanner is strongly connected.)

Our first contribution is a new, improved variant of the weak spanner for $S \subset E^2$ constructed in [12]. We construct a graph $G_{\pi/2}(S)$ for S whose outdegree is bounded by 4. (The weak spanner used in [12] has outdegree 6.) On the other side, our weak spanner has stretch factor $\sqrt{3+\sqrt{5}} \approx 2.288$, whereas the one from [12] is 2. Nevertheless, we argue that our weak spanner yields a faster static data structure for our search problem: If the points of S are distributed uniformly in a square range of E^2, the graph $G_{\pi/2}(S)$ is not only a more space efficient data structure for circular range queries than some graph $G_\gamma(S)$ for $\gamma \leq \frac{\pi}{3}$ but it yields also a better expected query time: If we have to report the points of S in a disc with radius r, then we must traverse the directed edges having the origin in a concentric disc with radius fr, where f is the stretch factor of the graphs. The query time is determined by the number of traversed edges. The number of traversed edges is the number of points in the disc multiplied by the outdegree of the points. The expected number of points in the disc is quadratic in the radius. For example, the graph $G_{\pi/3}(S)$ has a stretch factor 2 and outdegree 6. $G_{\pi/2}(S)$ has a stretch factor $\sqrt{3+\sqrt{5}}$ and outdegree 4. Therefore, the expected query time in $G_{\pi/2}(S)$ is $\frac{3+\sqrt{5}}{6} \approx 0.87$ times the expected query time in $G_{\pi/3}(S)$.

Our second contribution is the development of a *deterministic* dynamic data structure for the "moving visitor searching problem", that guarantees running times for $query(q, r)$ and $insert(p)$ independent of n. The data structure from [12] yields such results only using randomization (perfect hashing). Our result is based on a careful, deterministic choice of $O(n)$ Steiner points. They replace the randomized approach from [12], that uses a grid as Steiner points and applies perfect hashing for compacting it.

The paper is organized as follows. In Section 2 we present a variant of the γ-angle graph [12], for a point set $S \subset E^2$ of n points, which outdegree bounded by four. We prove that

this graph is a weak spanner for S with stretch factor $\sqrt{3+\sqrt{5}}$, and we give a plane sweep algorithm computing it in $O(n \log n)$ time.

In Section 3 we place $O(n)$ carefully chosen Steiner points into the scene supporting the point location for the visitor in $O(1)$ time. Here we assume that the visitor moves slowly, i.e., the distance between two consecutive positions of the visitor is at most a constant times the distance of the closest pair of S.

Finally, in Section 4, we show how we can insert a new object into the scene (a new point into S resp.) at the visitor's position q in $O(1+k)$ time, where k is the number of points (original points plus Steiner points) in the disc with center q and radius $r\sqrt{3+\sqrt{5}}$.

2 The two dimensional $\frac{\pi}{2}$-angle graph

In this section we construct a weak spanner $G_{\pi/2}(S)$ for the set $S \subset E^2$ of n points. The graph $G_{\pi/2}(S)$ contains at most $4n$ directed edges. For the definition and the construction of $G_{\pi/2}(S)$ we use a slightly more complicated distance measure than Fischer et al.[12]. This distance measure leads to some trickier plane sweep algorithm to obtain an $O(n \log n)$ construction time.

In the second part of the subsection we show that the graph $G_{\pi/2}(S)$ is a weak spanner with stretch factor $\sqrt{3+\sqrt{5}} \approx 2.288$ for the point set S. The proof uses different arguments than the proofs for the γ-angle graph for $\gamma \leq \frac{\pi}{3}$.

2.1 Construction

We define the graph $G_{\pi/2}(S)$ as follows: Rotate the positive x-axis over angles $i\frac{\pi}{2} - \frac{\pi}{4}$ for $0 \leq i < 4$. This gives 4 rays $h_0, ..., h_3$. Let $c_0, ..., c_3$ be the cones that are bounded by two successive rays. The ith ray belongs to cone c_i and the $(i+1)$st ray does not. For $0 \leq i < 4$, let l_i be the ray that emanates from the origin and halves the angle in the cone c_i, i.e. l_i is coincides with the positive/negative x-/y-axis respectively. For each point $p \in S$ translate the cones $c_0, ..., c_3$ and the corresponding rays $l_0, ..., l_3$ such that its apexes and starting points are at p. Denote by $c_0^p, ..., c_3^p$ and $l_0^p, ..., l_3^p$ these translated cones and rays, respectively (Figure 1.a). Now we build $G_{\pi/2}(S)$ such that for each $p \in S$ and $0 \leq i < 4$, if the cone c_i^p contains a point of S then add a directed edge pq from p to some closest point q in c_i^p w.r.t. the distance measure $dist_i : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R} \times \mathbb{R}$ defined as follows: $dist_i(p,q) = (|x(p) - x(q)|, |y(p) - y(q)|)$ if $i = 0, 2$, i.e. $q \in c_0^p \cup c_2^p$ and $dist_i(p,q) = (|y(p) - y(q)|, |x(p) - x(q)|)$ if $i = 1, 3$. The total order under the pairs is defined by the lexicographical order.

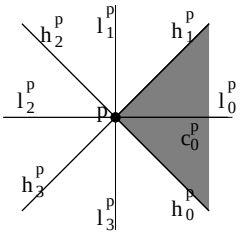

Figure 1: The definition of h_i^p, l_i^p and c_i^p.

Now we describe how the graph $G_{\pi/2}(S)$ can be efficiently built. Our construction consists of four phases. In the ith phase, $i = 0, .., 3$, we perform a plane sweep in the direction $i\frac{\pi}{2}$ in order to compute the neighbor in the ith translated cone c_i^p for each $p \in S$. Our plane sweep is a modified version of the plane sweep of Ruppert and Seidel [15]. We describe the 0th phase, the other phases perform analogously.

The Algorithm: First we sort the points of S non decreasing w.r.t. the x-coordinate. The sweepline moves from left to right. We maintain the invariant that for each point q left to the sweepline, its neighbor p in c_0^q has been computed if p is also left to the sweep line.

We initialize a data structure $D = \emptyset$. D will contain the points left to the sweepline whose neighbor has not yet been computed. The points in D are sorted increasing w.r.t. the y-coordinate. To handle the distance function $dist_i$ correctly, when the sweepline reaches a point p we put all points with the same x-coordinate as p in a data structure A and we work up these points in the same main step:

1. Let A be the set of points with the same x-coordinate as p ordered increasing w.r.t. the y-coordinate and let $A' = A$.

2. While $A \neq \emptyset$ we do the following:

   (a) Let p be the point of A with minimum y-coordinate. Determine the set of points $B(p) = \{q \in D \ : \ p \in c_0^q\}$. Let $B(p)$ be ordered also increasing w.r.t. the y-coordinate. Set $p' = p$. (The variable p' contain at each time the point of A with highest y-coordinate, such that the points of A with lower y-coordinate than p' cannot be a neighbor of any point of D.)

   (b) For each $q \in B(p)$ (in increasing order w.r.t. the y-coordinate) determine the point $p^* \in A$ with $|y(q) - y(p^*)|$ minimal and join q with p^* by a directed edge. Then delete q from D and set $p' = p^*$.

   (c) Delete p from A. Then delete all points from A having a lower y-coordinate than p'.

3. Insert the points of A' into D.

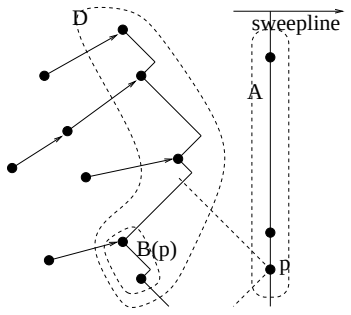

Figure 2: Illustration of the main step of the plane sweep.

Correctness: Now we prove by induction that the invariant is satisfied after the main step of the plane sweep. In the data structure D we maintain the points left to the sweepline l whose neighbor is not left to l. At the beginning of the algorithm no point of S is left to l, so $D = \emptyset$. In the main step we must determine the points of D whose neighbor is on

the sweepline l, compute the neighbor for these points, and update D, i.e. delete from D the points whose neighbor has been found and insert the points lying on l into D. In A we maintain the points on l, for which it is not yet decided if it is a neighbor of a point of D. Therefore, at the beginning A must contain all points of S on l. This is satisfied after step 1. We show that in step 2, for each point $q \in D$ with $c_0^q \cap A \neq \emptyset$ the neighbor is computed correctly. For the points $q \in D$ doesn't exist any point $q^* \in c_0^q$ left to l. On the other side, if a point $q \in D$ has a point $q^* \in A$ in c_0^q then q is contained in $B(p)$ in step 2.a for a $p \in A$, its neighbor computed correctly in step 2.b and then deleted from D. In step 2.c the points are deleted from A which are surely not a neighbor of a point of D. Therefore, after step 2 D contains the points left to l whose neighbor is not left to l and not on l. If we want to move the sweepline right to the next point, D mustn't contain any point whose neighbor is already computed (these points are deleted in step 2.b), but D must contain the points on l, because their neighbor is not yet computed. The points on l are inserted into D in step 3. So we can move the sweepline, the invariant is satisfied.

Analysis: For D, A and $B(p)$ we need data structures which support checking emptiness; insertion, deletion of a point; finding the next and the previous element w.r.t. y-coordinate for a given element of the data structure; and finding for a real number the point with nearest y-coordinate in the data structure (query). Using balanced search trees we obtain an $O(\log n)$ time for insertion, deletion, query and for finding the next and previous element, and $O(1)$ time for checking emptiness. If we link the nodes of the search tree in a doubly linked sorted list then we can find the next and the previous element in $O(1)$ time, too. For a point $q \in D$, let $next(q)$ and $prev(q)$ denote the next and the previous element of q in D w.r.t. the y-coordinate. In step 2.a we can determine $B(p)$ in the following way: Determine the nearest point $q \in D$ to p w.r.t. the y-coordinate. If q belongs to c_2^p (the reflected cone) then insert q into $B(p)$. Let $q' = q$. While $next(q)$ belongs to c_2^p, insert $next(q)$ into $B(p)$ and set $q = next(q)$. Than Let $q = q'$. While $prev(q)$ belongs to c_2^p insert $prev(q)$ into $B(p)$ and set $q = prev(q)$.

Note that each point $p \in S$ is inserted and deleted from D, A and $B(p)$ exactly once. For each point q the nearest point $p \in A$ is computed once, too. Therefore, we can summarize the above analysis in the following theorem:

Theorem 2.1 *The graph $G_{\pi/2}(S)$ can be computed in $O(n \log n)$ time and $O(n)$ space.* □

2.2 The weak spanner property

Theorem 2.2 *$G_{\pi/2}(S)$ is a weak spanner with stretch factor $\sqrt{3+\sqrt{5}}$.*

Proof: We prove that for each $s, t \in S$ there is a directed path P from s to t in $G_{\pi/2}(S)$, such that for each point $v \in P$ the Euclidean distance $dist(s, v)$ is at most $\sqrt{3+\sqrt{5}} \cdot dist(s, t)$.

Consider the following simple algorithm.

- Let $s_0 = s$ and $j = 0$. While $s_j \neq t$ do the following:
- Let $c_i^{s_j}$ be the cone of s_j containing t. Set s_{j+1} to the neighbor of s_j in $G_{\pi/2}(S)$ in this cone and set $j = j + 1$.

The above algorithm finds a path with the properties we wish. To prove this we first define a potential function Φ on the points of S and show that Φ decreases in 'almost' every step

of the above algorithm. Let $\phi_1(s)$ be the Manhattan distance $dist_M(t,s)$ from t to s and let $\phi_2(s)$ be $|x(t)-x(s)|$ if $t \in c_0^s \cup c_2^s$ and $|y(t)-y(s)|$ if $t \in c_1^s \cup c_3^s$. We define $\Phi(s)$ to be the pair $(\phi_1(s), \phi_2(s))$. The total order on the values of Φ is the lexicographical order of the pairs.

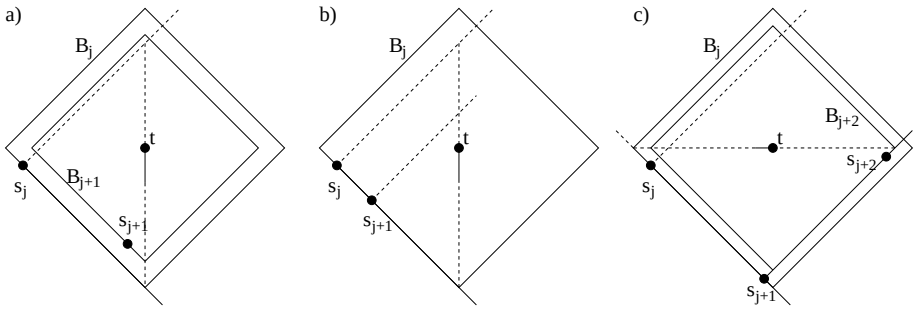

Figure 3: The potential Φ during the construction of P.

We show that the points of the path created by the above algorithm have the following property:

1. either $\Phi(s_j) > \Phi(s_{j+1})$,
2. or $\phi_1(s_j) = \phi_1(s_{j+1})$, $\phi_2(s_j) \leq \phi_2(s_{j+1})$ and $\phi_1(s_j) > \phi_1(s_{j+2})$.

This property implies that the algorithm terminates and finds t. Note, if $s_{j+1} = t$ then $\Phi(s_j) > \Phi(t) = (0,0)$. Let $c_i^{s_j}$ be the cone at s_j that contains t. By the definition of $G_{\pi/2(S)}$ either there is a directed edge $< s_j, t >$ in $G_{\pi/2}(S)$ or an other point $s_{j+1} \in c_i^{s_j}$ and a directed edge $< s_j, s_{j+1} >$ with $dist_i(s_j, s_{j+1}) \leq dist_i(s_j, t)$. Let $B_j = \{x \in \mathbb{R}^2 \ : \ dist_M(t,x) \leq dist_M(t,s_j)\}$. The point s_{j+1} is clearly contained in B_j (Figure 3) and so $\phi_1(s_j) \geq \phi_{j+1}$. We distinguish three cases.

Case 1: $dist_M(s_j,t) > dist_M(s_{j+1},t)$. Then $\phi_1(s_j) > \phi_1(s_{j+1})$ (Figure 3.a). Therefore, property 1 holds.

Case 2: $dist_M(s_j,t) = dist_M(s_{j+1},t)$ and $t \in c_i^{s_{j+1}}$. Then $\phi_1(s_j) = \phi_1(s_{j+1})$ and $\phi_2(s_j) > \phi_2(s_{j+1})$ (Figure 3.b). Therefore, property 1 holds.

Case 3: $dist_M(s_j,t) = dist_M(s_{j+1},t)$ and $t \in c_{i+1 \bmod 4}^{s_{j+1}}$. Then it can be happen, that $\phi_1(s_j) = \phi_1(s_{j+1})$ and $\phi_2(s_j) \leq \phi_2(s_{j+1})$. But in this case the next point s_{j+2} cannot be on the boundary of B_j (Figure 3.c). Therefore, property 2 holds.

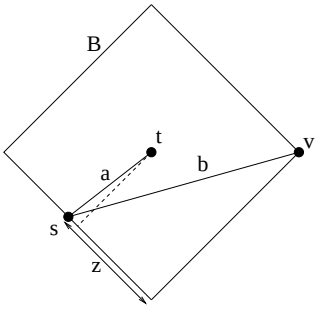

Figure 4: Definitions for the computing the weak spanner factor.

Now we prove that for each $v \in P$ the Euclidean distance $dist(s,v)$ is at most $\sqrt{3+\sqrt{5}} \cdot dist(s,t)$. We have seen, that $B = B_0$ contains each point s_j of the path P. Let $a = dist(s,t)$ and $b = dist(s,v)$. We have to maximize $\frac{b}{a}$ such that t is the center of B, s is on the boundary of B and x is contained in B. For a fixed point $p \in B$ one of the corners u of B satisfy $dist(p,u) = max(dist(p,x) : x \in B)$. So we can fix v at a corner of B and. Consider Figure 4. By aid of the theorem of Pythagoras and differentiation we obtain, that $\frac{b}{a}$ is maximal, when $z = (\sqrt{5}-1)l$, where l is the side length of B, and this maximum is $\sqrt{3+\sqrt{5}}$. □

Remark: Note that without the second component of $dist_i$, $0 \leq i < 4$ the obtained graph would be not necessary strongly connected. Consider the following example: let $S = \{p_1, p_2, p_3, p_4, p_5\}$ and let $(1,0)$, $(0,1)$, $(-1,0)$, $(0,-1)$ and $(0,0)$ be the coordinates of the points. Then the point in the center is not necessary reachable from the other points by a directed path. If we only have three cones (i.e. $\gamma = \frac{2\pi}{3}$) the construction of a non strongly connected example is quite easy.

3 Steiner points for navigation in the scene

In this subsection we present a deterministic method to find the nearest neighbor of the query point q in c_i^q w.r.t. $dist_i$, $i = 0,..,3$. The BFS procedure in a range query $query(q,r)$ starts with these points. In order to find a nearest neighbor we exploit that the visitor moves through the scene slowly. Furthermore, we extend the original point set S with $O(n)$ carefully placed Steiner points. In the extended point set S' we can take advantage of the fact that the nearest neighbors of the query position q is close to the nearest neighbors of the previous query position q_{prev} and we can find it in constant time.

3.1 The "moving visitor" structure

First we describe how we place the Steiner points. We proceed similarly to the *mesh generation* technique of Bern et al. [5][6]: First we produce a linear size, balanced quadtree and we take the corners of the boxes of this quadtree as Steiner points. We describe this technique briefly below.

Definitions [5]: A *quadtree* is a recursive subdivision of the plane into square boxes. The nodes of the quadtree are the boxes. Each box is either a *leaf* of the tree or is *split* into four equal-area children. A box has four possible *neighbors* in the four cardinal directions; a neighbor is a box of the same size sharing a side. A *corner* of a box is one of the four vertices of its square. The corners of the quadtree are the points that are corners of its boxes. A side of a box is *split* if either of the neighboring boxes sharing it is split. A quadtree is called *balanced* if each side of an unsplit box has at most one corner in its interior. An *extended neighbor* of a box is another box of the same size sharing a side or a corner with it.

Building balanced quadtree for S [5]: We start with a root box b concentric with and twice as large as the smallest bounding square of S. We recursively split b as long as b has a point of S in its interior and one of the following conditions holds: (i) b has at least two points or (ii) b has side length l and contains a single point $p \in S$ with nearest neighbor in S closer than $2\sqrt{2}l$ or (iii) one of the extended neighbors of b is split. Then we balance the quadtree. We remark that the balancing increases the space requirement of the quadtree only by a constant factor. After all splitting is done, every leaf box containing a point of S is

surrounded by eight empty leaf boxes of the same size.

Building linear size balanced quadtree [5]: The only nonlinear behavior of the above algorithm occurs, when a nonempty box is split without separating points of S. If this happens, we need to "shortcut" the quadtree construction to produce small boxes around a "dense" cluster without passing through many intermediate size of boxes. We construct the quadtree for the cluster recursively and we treat the cluster as an individual point. In this way we obtain a linear size quadtree for S, i.e. the number of the boxes is linear to n. The linear size quadtree for S can be constructed in $O(n \log n)$ time and $O(n)$ space. For some algorithmic details we refer to [6].

Here ends the part borrowed from Bern et al. [5].

We call a dense cluster with the containing box and the eight extended neighbor boxes an *extended cluster*. We remark that in the extended clusters the balance property is maintained by the above description. It will be crucial for the fast navigation that an extended cluster has only constant number of corners on the boundary.

Building $G_{\pi/2}$ from the quadtree: We extend S with the $O(n)$ corners of the linear size quadtree and construct the graph $G_{\pi/2}(S')$ for the extended point set S'. If we have the quadtree, we can determine for each point of S' its four neighbors in constant time. It follows from the fact that the neighbors of a point $p \in S'$ in $G_{\pi/2}(S')$ are in the interior of the leaf box $b(p)$ containing p or in the interior of a neighboring box of $b(p)$. Figure 5 illustrates the possible cases.

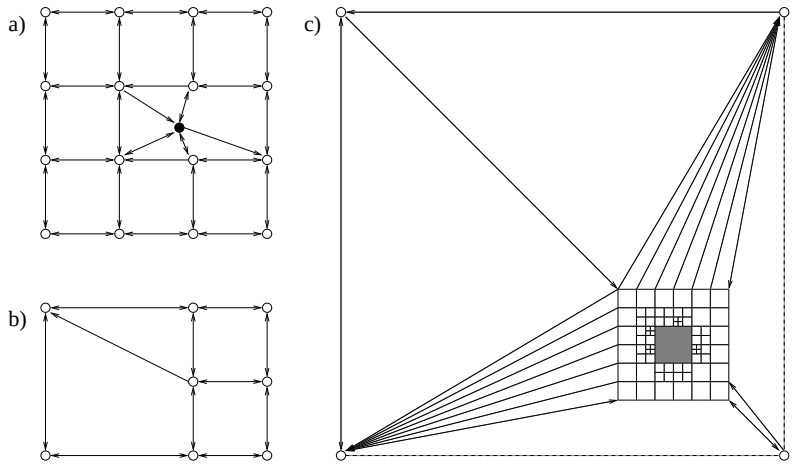

Figure 5: The scheme to construct $G_{\pi/2}(S')$ from the quadtree: a) An original point of S, the box containing it and the eight extended neighbor boxes. b) A split side. c) A shortcut.

Point location in constant time beyond slow motion of the visitor: If we have $G_{\pi/2}(S')$, we can determine for each point p' of S' the leaf box $b(p')$ of the quadtree containing p' in constant time. Furthermore, if x is the distance between the closest pair of S then the side length of the smallest box of the quadtree is a constant fraction of x. These observations imply that for the query position q the box $b(q)$ of the quadtree containing q can be computed in constant time via $G_{\pi/2}(S')$, if we know the box $b(q_{prev})$ containing the previous query position q_{prev}, since the line segment (q_{prev}, q) intersects only constant many leaf boxes of the quadtree. The intersections and so the box $b(q)$ can be computed in constant time. Then the nearest neighbor of q in S' in each cone c_i^q, $i = 0, ..., 3$ can be determined in constant time,

too. Knowing these nearest neighbors we can start the BFS in order to answer the range query $query(q, r)$. We conclude:

Theorem 3.1 *Let S be a scene of n equal size objects. Assume that the visitor moves slowly. There is a data structure which provides to report the potentially visible objects in time $O(1+k)$ time, where k is the number of the edges of $G_{\pi/2}(S')$ having the origin in the interior of a circle which is concentric with the query circle and has a radius $\sqrt{3+\sqrt{5}}$ times of the query circle. The space requirement of the data structure is $O(n)$ and can be built in $O(n \log n)$ time.* □

4 The incremental data structure, lazy updates

In this subsection we study the incremental version of the search problem. Here insertion of a new object at the current position of the visitor is allowed. We show how we can insert a point into the graph $G_{\pi/2}(S')$ in a time which match to the query time.

If we insert a new point into $G_{\pi/2}(S')$ we first must maintain the balance property in the quadtree. Then we must update the adjacencies at the affected vertices of $G_{\pi/2}(S')$ corresponding to corners of the changed boxes or they neighboring boxes. Updating $G_{\pi/2}(S')$ at the affected vertices costs only a constant time per box even in the case of a changed short cut. Therefore the update time is determined by the number of affected boxes. The main problem is to maintain the balance property. Let $l(b)$ be denote the *level* of box b, it is the length of the tree path from the root box of the quadtree to the box b. If a leaf box b is splitted then the balance condition can be violated in each level higher than $l(b)$ (Figure 6). Therefore, the worst case update time for the rebalancing of the quadtree depends on the depth of the quadtree and so on the total scene.

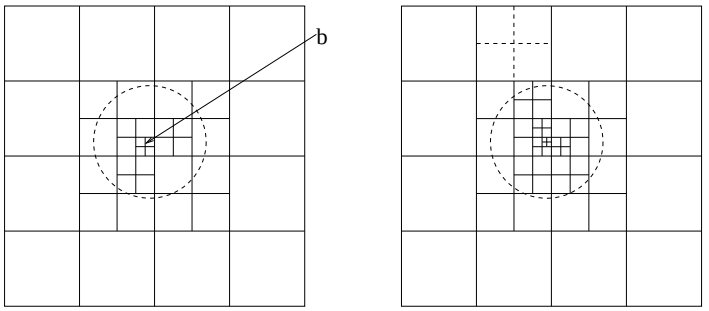

Figure 6: Rebalancing of the quadtree

Let C_r be query disc having a radius r and C_{rf} the disc concentric with C_r and having a radius $rf = r\sqrt{3+\sqrt{5}}$. In order to made the rebalancing independent from the total size of the scene we make *lazy updates*, i.e. we split only the boxes intersecting C_{rf}. After the splittings we update only the vertices of $G_{\pi/2}(S')$ in C_{rf}. The remainder splittings will be performed later, when the current C_{rf} intersects the according boxes. So, at the beginning of each query we must check, whether the disc C_{rf} affects new leaf boxes violating the balance property. In this case we perform the necessary splittings. In this way the update time is at most linear to the number of vertices of the updated graph $G_{\pi/2}(S')$ in the disc C_{rf}. This time matches to the time of the searching. We summarize:

Theorem 4.1 *Let S be a scene of n equal size objects. Assume that the visitor moves slowly. There is a incremental data structure which provides inserting a new object at the position of the visitor in time $O(1+k)$, where k is the number of the edges of $G_{\pi/2}(S')$ having the origin in the interior of a circle which is concentric with the query circle and has a radius $\sqrt{3+\sqrt{5}}$ times of the query circle.* □

5 Conclusions

We considered special range searching problems motivated by a walkthrough simulation of a large geometric scene.

We constructed a directed graph $G_{\pi/2}(S)$ for a set $S \subset E^2$ of n points whose outdegree is bounded by 4. We gave a plane sweep algorithm to compute $G_{\pi/2}(S)$ in $O(n \log n)$ time. Then we proved that $G_{\pi/2}(S)$ is a weak spanner with stretch factor $\sqrt{3+\sqrt{5}} \approx 2.288$. Our weak spanner decreases the space requirement of the data structures of Fischer et al. [12] drastically and yields a faster static data structure for the search problem in expected sense.

To solve the incremental "moving visitor searching problem" we presented a deterministic linear space data structure. Our data structure guarantees running times for $query(q, r)$ and $insert(p)$ independent of n. It is based on careful choice of $O(n)$ Steiner points.

The main directions of the future research are the following. We want to generalize the graph $G_{\pi/2}(S)$ in $d \geq 3$-dimensions, such that we obtain a weak spanner in E^d.

Further, we are interested in other linear size data structures to solve our searching problems. In particular, does a data structure exist solving the "moving visitor search problem" without Steiner points?

Acknowledgment: We would like to thank Artur Czumaj and Friedhelm Meyer auf der Heide for the helpfull comments and suggestions.

References

[1] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. Technical Report CS-1997-11, Duke Univ., Dep. of Comp. Sci., 1997. To appear: in Discrete and Computational Geometry: Ten Years Later.

[2] A. Aggarwal, M. Hansen, and T. Leighton. Solving query-retrival problems by compact voronoi diagrams. In *22nd ACM Symposium on Theory of Computing (STOC'90)*, pages 331–340, 1990.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *5th ACM-SIAM Symposium on Discrete Algorithms (SODA'94)*, pages 573–582, 1994.

[4] J. L. Bentley and H. A. Maurer. A note on the euclidean near neighbor searching in the plane. *Information Processing Letters*, 8:133–136, 1979.

[5] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *J. Comp. Syst. Sci.*, 48:384–409, 1994.

[6] M. Bern, D. Eppstein, and S. H. Teng. Parallel construction of quadtrees and quality triangulations. In *3rd Workshop on Algorithms and Data Structures (WADS'93)*, pages 188–199, 1993.

[7] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 291–300, 1993.

[8] B. Chazelle, R. Cole, F. P. Preparata, and C. Yap. New upper bounds for neighbor searching. *Information and Control*, 68:105–124, 1986.

[9] D. Z. Chen, G. Das, and M. Smid. Lower bounds for computing geometric spanners and approximate shortest paths. In *8th Canadian Conference in Computational Geometry (CCCG'96)*, pages 155–160, 1996.

[10] L. P. Chew. There is a planar graph almost as good as the complete graph. In *2nd Annual ACM Symposium on Computational Geometry*, pages 169–177, 1986.

[11] M. Dietzfelbinger and F. Meyer auf der Heide. Dynamic hashing in real time. In *Informatik: Festschrift zum 60. Geburtstag von Günter Hotz*, pages 95–119. Teubner, Stuttgart, 1992.

[12] M. Fischer, F. Meyer auf der Heide, and W.-B. Strothmann. Dynamic data structures for realtime management of large geometric scenes. In *5th Annual European Symposium on Algorithms (ESA'97)*, pages 157–170, 1997.

[13] J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete euclidean graph. *Discrete and Computational Geometry*, 7:13–28, 1992.

[14] F. P. Preparata and M. I. Shamos. *Computational Geometry An Introduction*. Springer Verlag, New York, 1985.

[15] J. Ruppert and R. Seidel. Approximating the d-dimensional complete euclidean graph. In *3rd Canadian Conference in Computational Geometry (CCCG'91)*, pages 207–210, 1991.

[16] J. S. Salowe. Constructing multidimensional spanner graphs. *International Journal of Computational Geometry and Applications*, 1:99–107, 1991.

[17] M. Smid. *Closest-Point Problems in Computational Geometry*. To appear in: Handbook on Computational Geometry, edited by J.-R. Sack, North Holand, Amsterdam.

[18] P. M. Vaidya. A sparse graph almost as good as the complete graph on points in k dimensions. *Discrete and Computational Geometry*, 6:369–381, 1991.

Appendix

A The γ-angle graph

Fischer et al. [12] presented a fully dynamic data structure for real time management of large geometric scenes. They use the weak spanner property of a dense graph to answer circular range queries performing a certain point location and a BFS.

The basic data structure in [12] is called γ-*angle graph*. The same construction was called Θ-*graph* by Keil and Gutwin [13] and Ruppert and Seidel [15].

We restrict the description to the two dimensional case. Let $k \geq 6$ be an integer constant and let $\gamma = \frac{2\pi}{k}$. Rotate the positive x-axis over angles $i\gamma$ for $0 \leq i < k$. This gives k rays $h_0, ..., h_{k-1}$. Let $c_0, ..., c_{k-1}$ be the cones that are bounded by two successive rays. The ith ray h_i belongs to cone c_i and h_{i+1} does not. Also, for $0 \leq i < k$, let l_i be the ray that emanates from the origin and halves the angle in the cone c_i. Define a graph in the following way. For each point $p \in S$ translate the cones $c_0, ..., c_{k-1}$ and the corresponding rays $l_0, ..., l_{k-1}$ such that its apexes and starting points are at p. Denote $c_0^p, ..., c_{k-1}^p$ and $l_0^p, ..., l_{k-1}^p$ this translated cones and rays respectively. For $p, q \in S$ and $0 \leq i < k$ define the distance $dist_i(p, q)$ as the Euclidean distance between p and the projection of q onto the translated ray l_i^p if q is contained in the translated cone c_i^p, and ∞ otherwise. For $0 \leq i < k$, if the cone c_i^p contains a point of S then add a directed edge from p to one of the closest points in c_i^p defined by the distance function $dist_i$.

The graph obtained in this way is called the γ-angle graph of S, denoted by $G_\gamma(S)$. It contains at most $kn = O(n)$ edges and can be build in $O(n \log n)$ time by performing k sorting and k plane sweeps. In [12] it is shown that $G_\gamma(S)$ is a weak spanner for S with stretch factor

$f = \max(\sqrt{1 + 48\sin^4\frac{\gamma}{2}}, \sqrt{5 - 4\cos\gamma})$. If we take $\gamma = \frac{\pi}{3}$ then we get $f = 2$. We remark that for $k \geq 7$ the γ-angle graph is a $(\frac{1}{1-2\sin(\gamma/2)})$-spanner (v. Ruppert and Seidel [15]).

If the center of the query circle q is a point of S the weak spanner property implies that we can perform a BFS beginning at q extracting only the points, whose distance from q is at most fr. If $q \notin S$ the BFS can be started at a point $p \in S$ not farer from q than fr, where f is the stretch factor. As mentioned, to find such a point in a time independent of n is impossible in the general case.

If the radius r of the query circles is known before the preprocessing of S, the root of the BFS can be found by adding the nodes of a grid as Steiner points to S. The size of the data structure remains linear if we store only those Steiner points (called the essential Steiner points) explicitly which have a neighbor in the original point set S. Using *dynamic perfect hashing* [11] to store the coordinates of the essential Steiner points we can find a Steiner point close to the current query position in constant time. This approach was used by Fischer et al. [12]. We remark that this method is randomized because of the hashing from [11] used. Furthermore, it uses the floor function and indirect addressing. Thus it is not captured by the algebraic computation tree model where an $(\log n)$ lower bound holds.

Using the hashing structure it is possible to insert and delete a point of S from the data structure of [12] in $\overline{O}(r'^2)$ and $\overline{O}(r'^2 \log r')$ time respectively, where r' is the step size of the grid, multiplied by a constant only dependent on γ.

The γ-angle graph can be generalized to E^d, as described in [12], in natural way. In this case, we have to divide the space at each point $p \in S$ into cones with a maximum angle $\gamma \leq \frac{\pi}{3}$. This cones must cover $\mathbb{R}^d$ but can overlap. The d-dimensional γ-angle graph for S can be computed in $O(n \log^{d-1} n)$ time and $O(n \log^{d-2} n)$ space by aid of *plane sweeps*. The generalization of the extension of S with Steiner points and the hashing structure to store only the essential Steiner points is straight forward.