Geometric Data Structures for Computer Graphics

Elmar Langetepe
Bonn University, Germany
email: langetep@cs.uni-bonn.de

Gabriel Zachmann
Clauthal University, Germany
email: zach@in.tu-clauthal.de

July 9, 2009

Contents

Introduction

In recent years, methods from computational geometry have been widely adopted by the computer graphics community yielding elegant and efficient algorithms. This book aims at endowing practitioners in the computer graphics field with a working knowledge of a wide range of geometric data structures from computational geometry. It will enable readers to recognize geometric problems and select the most suitable data structure when developing computer graphics algorithms.

The book will focus on algorithms and data structures that have proven to be versatile, efficient, fundamental, and easy to implement. Thus practitioners and researchers will benefit immediately from this book for their everyday work.

Our goal is to familiarize practitioners and researchers in computer graphics with some very versatile and ubiquitous geometric data structures, enable them to readily recognize geometric problems during their work, modify the algorithms to their needs, and hopefully make them curious about further powerful treasures to be discovered in the area of computational geometry.

Figure 1: Boehme-Zitat Böhme [1993]

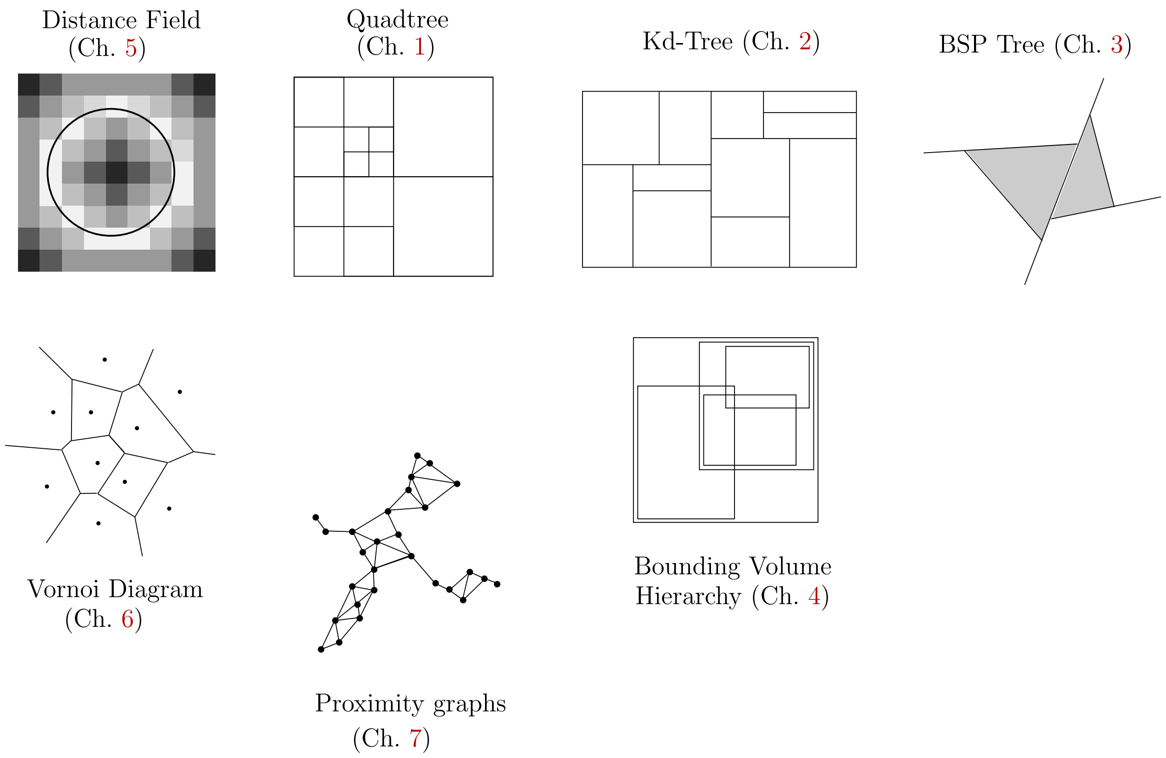


Figure 2: An overview of some of the data structures presented in this book.

In order to achieve these goals in an engaging yet sound manner, the general concept throughout the book is to present each geometric data structure in the following way: first, the data strucure will be defined and described in detail; then, some of its fundamental properties will be highlighted; after that, one or more computational geometry algorithms based on the data structure will be presented; and finally, a number of recent, representative and practically relevant algorithms from computer graphics will be described in detail, showing the utilization of the data structure in a creative and enlightening way.

We do not try to provide an exhaustive survey of the topics touched upon here — this would be far beyond the scope of this book. Neither do we aspire to present the latest and greatest algorithms for a given problem, for two reasons. First, the focus is on geometric data structures, and we do not want to obstruct the view by complicated algorithms. Second, we feel that for practical purposes a good trade-off between simplicity and efficiency is important.

The intended audience are practitioners working in 3D computer graphics (VR, CAD/CAM, entertainment, animation, etc.) and students from both computer graphics and computational geometry. Readers should be familiar with the basic principles of computer graphics and the type of problems in the area.

We have arranged the chapters in roughly increasing degree of difficulty. The hierarchical data structures are ordered by increasing flexibility, while the non-hierarchical chapters build on each other. Finally, the last three chapters present generic techniques for "kinetizing", "robustification", and dynamizing geometric data structures.

Figure 2 provides an overview of some of the data structures that will be discussed in the following chapters. Chapter 1 presents quadtrees and octrees, which are, arguably, the most popular data structure in computer graphics. Lifting one of the restrictions, thus makig the data structure more flexible, we arrive at kd-trees, which are presented in Chapter 2. We can make this even more flexible, which yields BSP trees, discussed in Chapter 3. From kd-trees, we can also derive bounding volume hierarchies, which are described in Chapter 4. Starting with a quadtree (or even a grid), we can store more information about the object(s), thus arriving at distance fields (Chapter 5). In a sense, distance fields are a discretized version of Voronoi diagrams, which are presented in Chapter 6.

Chapter 7

Geometric Proximity Graphs

In a previous chapter, we have already studied a data structure that provides a notion of proximity among its sites, without explicitely saying so, namely Delaunay diagrams (see Chapter 6). In this chapter, we will learn about other graphs that are defined over the notion of proximity (with different concrete meaning).

Interest in geometric proximity graphs (sometimes also referred to as *neighborhood graphs*) has been at a high level for about 20 years, in many diverse areas such as computational geometry, theoretical computer science, and graph theory, to name but a few. In a sense, polygonal meshes, which are an extremely common boundary representation of graphical objects, are a special kind of proximity graph.

These graphs can serve as a powerful tool to capture the *structure* or *shape* of otherwise unstructured point sets. Therefore, they have numerous applications in areas such as computer graphics, computer vision, geography, information retrieval, routing in ad-hoc networks, and computational biology, among many others (see Figure 7.1).

Even in psychology, they can help to explain some optical illusions [Sattar]. For instance, the well-known Mueller-Lyer illusion [Coren and Girgus, 1978] consists of two arrows, one with inward, the other with outward pointing arrow-heads (see Figure 7.2).

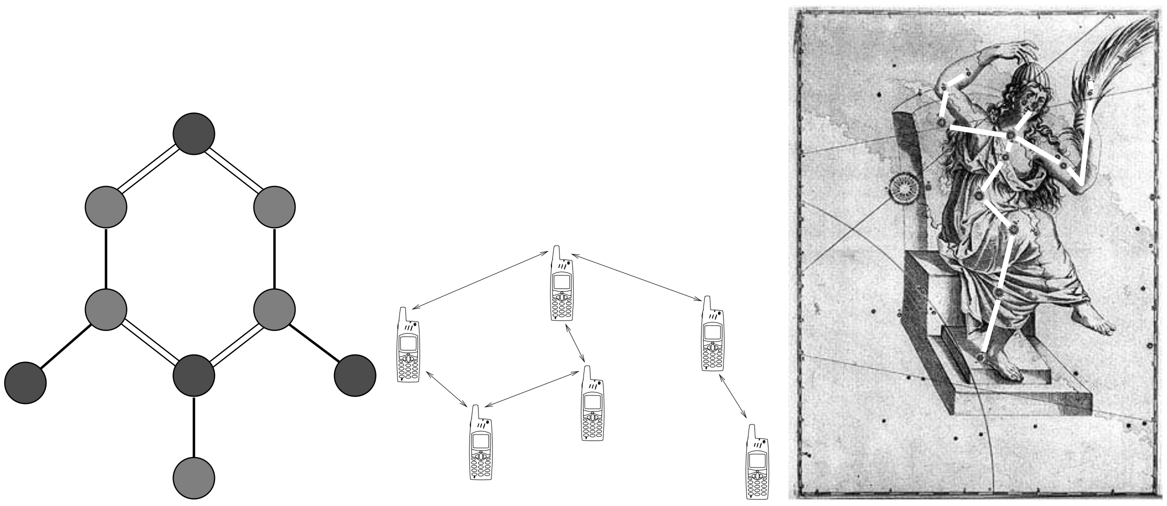


Figure 7.1: Identifying all pairs of close nodes in a set can reveal significant structure, such as a molecule or a mobile ad-hoc network. However, care must be exercised when establishing that proximity is indeed relveant (Cassiopeia courtesy of [Cassiopeia]).

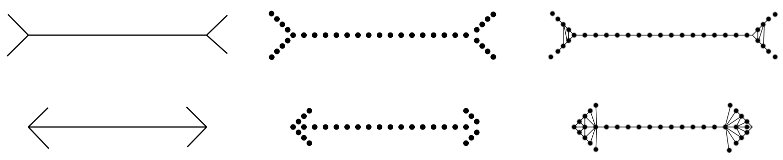


Figure 7.2: Proximity graphs can even help in psychology to explain some optical illusions.

In this chapter, we will present a small number of neighborhood graphs (other than polygonal meshes), and a few applications in computer graphics, where they can help to detect structure in point clouds.

7.1 A Small Collection of Proximity Graphs

In this section, we define a number of common proximity graphs and highlight some of their properties. This will usually be done by introducing a *neighborhood* that defines exactly when two nodes (i. e., points) are neighbors of each other. The following definitions and discussions build on [Jaromczyk and Toussaint, 1992].

7.1.1 Preliminary Definitions

Geometric graphs are graphs that are *embedded* in a metric space. Here, we will assume the space $\mathbb{R}^d$ together with an L_p norm, $1 \leq p \leq \infty$. The *length* between two points $x, y \in \mathbb{R}^d$ is defined as $d(x, y) := \|x - y\|_p = \left(\sum_{i=1}^{d} |x_i - y_i|^p\right)^{1/p}$.

Let V be a set of points in $\mathbb{R}^d$. Edges are (unordered) pairs of points $(p, q) \in V \times V$, denoted by pq.[1] In the following, the length of an edge is equal to the Euclidean distance between its two endpoints (any other metric could be used as well).

Proximity graphs are geometric graphs where the edges connect points that are in *proximity* to each other (or where at least one point is close to the other). If pq is an edge in such a proximity graph, then we say that p is a *neighbor* of q (and vice versa).

Exactly how proximity is defined depends on the type of neighborhood graph. It is always a geometric property, which, at least, involves the two points that are neighbors of each other (and, therefore, connected by an edge).

This property often involves spheres, so we define the sphere with center x and radius r as $S(x, r) := \{y \in \mathbb{R}^d \mid d(x, y) = r\}$. Analogously, we define the (closed) *ball* $B(x, r) := \{y \in \mathbb{R}^d \mid d(x, y) \leq r\}$.

7.1.2 Definitions of some proximity graphs

7.1.2.1 Unit disk graph

This is probably the neighborhood graph with the simplest definition. The set of edges of the unit disk graph, $UDG(V)$, is defined as

$$E := \{pq \mid d(p, q) \leq 1\}$$

i. e., two nodes are connected by an edge iff their distance is at most 1.

[1] Technically, one should distinguish between

1. the combinatorial structure of the graph, given by V and the set of edges $E \subset V \times V$, and
2. the *geometrical realization*, given by points that have an actual location in space and edges that are straight line segments connecting the points.

In the following, we will not differentiate between those two, hopefully without any confusion.

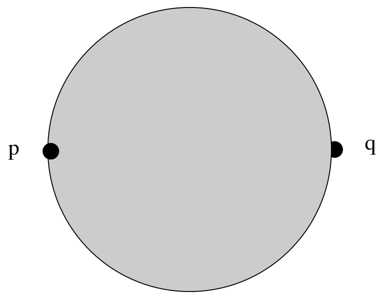


Figure 7.3: The defining neighborhood for the Gabriel graph is the *diameter sphere*.

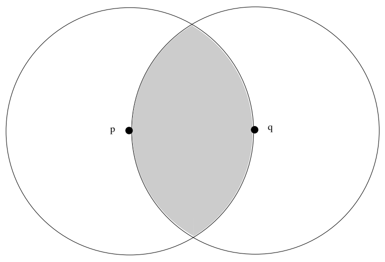


Figure 7.4: The *lune* is the defining neighborhood of the relative neighborhood graph.

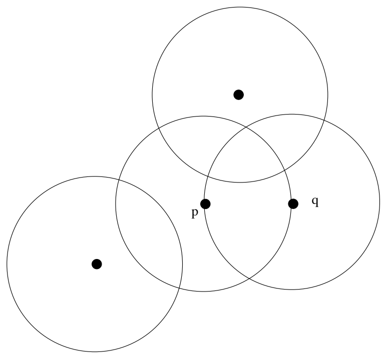


Figure 7.5: The sphere-of-influence graph is defined by *spheres-of-influence* around each point.

This definition is motivated by mobile ad-hoc networks, where each node (e. g., cell phone) can reach only other nodes within a certain radius (assuming that all phones have the same transmission power).

7.1.2.2 Relative neighborhood graph

We define a *lune* $L(p, q) := B(p, d) \cap B(q, d)$, where $d = \|p - q\|$ (see Figure 7.4).

The relative neighborhood graph of V, $RNG(V)$, is defined by the set of edges

$$E := \{pq \;\|\; L(p, q) \cap V = \emptyset\}$$

In other words,

$$pq \in E \;\Leftrightarrow\; \nexists v \in V : d(p, v) < d(p, q) \;\wedge\; d(q, v) < d(p, q).$$

Or, yet in other words,

$$pq \in E \;\Leftrightarrow\; \forall v \in V : d(p, q) \leq \max\{d(p, v), d(q, v)\}.$$

7.1.2.3 Gabriel graphs

The neighborhood here is a so-called *diameter sphere* $G(p, q) := B\big(\frac{p+q}{2}, \frac{d}{2}\big)$, where $d = \|p - q\|$ (see Figure 7.3).

The Gabriel graph over V, $GG(V)$, is defined by the set of edges

$$E := \{pq \;\|\; G(p, q) \cap V = \emptyset\}$$

In other words,

$$pq \in E \;\Leftrightarrow\; \forall v \in V : d(p, q) \leq \sqrt{d^2(p, v) + d^2(q, v)}\,.$$

7.1.2.4 β-skeletons

This is a family of neighborhood graphs, parameterized by $\beta, 1 \leq \beta < \infty$.

For a fixed β, the neighborhood is the intersection of two spheres:

$$U_\beta(p, q) := B\Big((1 - \frac{\beta}{2})p + \frac{\beta}{2}q, \frac{\beta}{2}d\Big) \;\cap\; B\Big((1 - \frac{\beta}{2})q + \frac{\beta}{2}p, \frac{\beta}{2}d\Big)$$

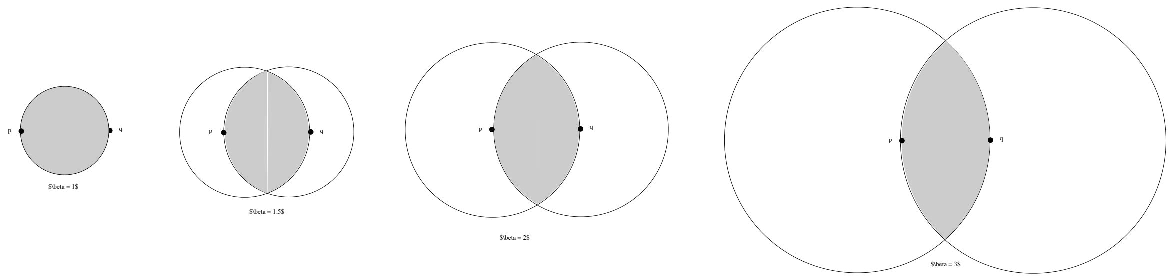


Figure 7.6: Examples of the family of lunes that define the β-skeleton.

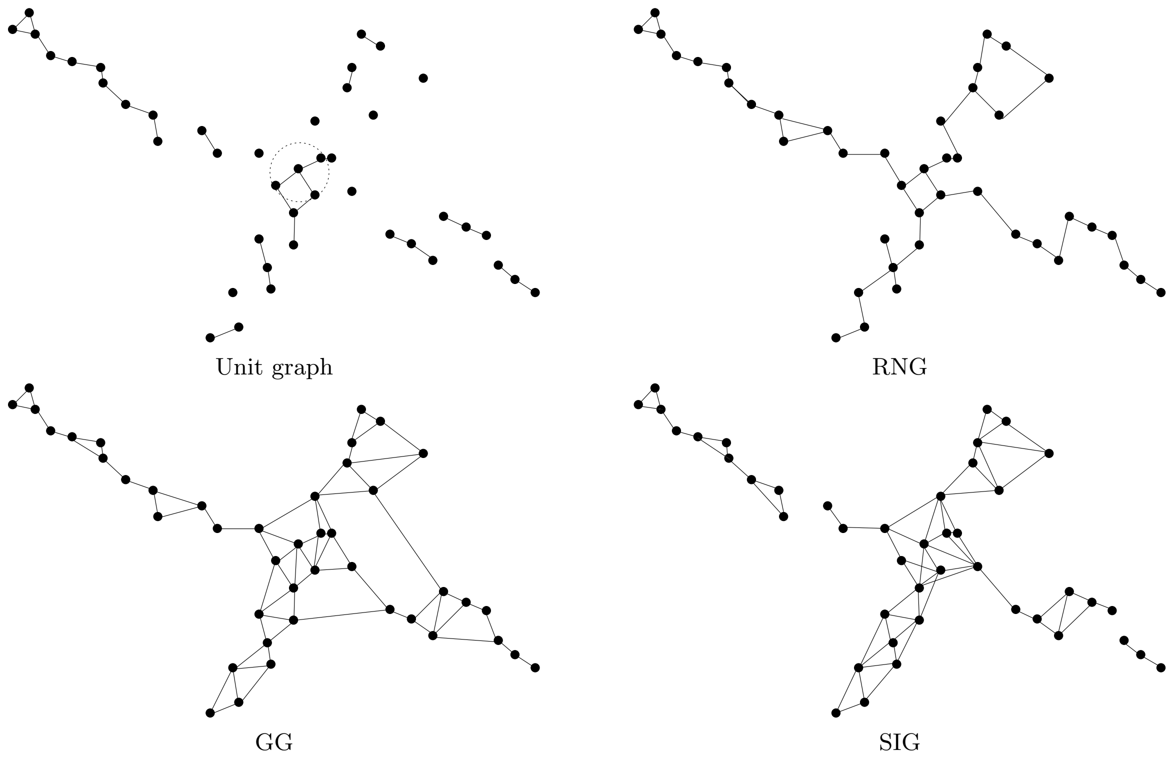


Figure 7.7: Example proximity graphs.

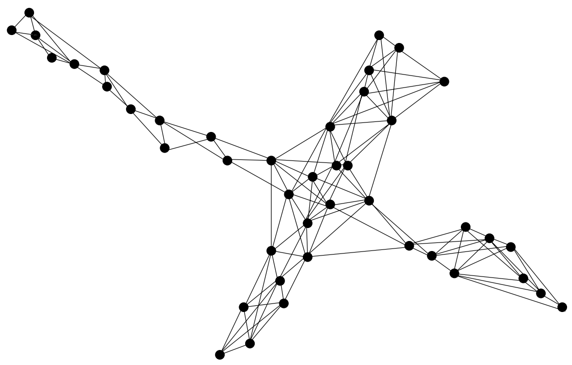


Figure 7.8: The 3-SIG over the same point cloud as in Figure 7.7.

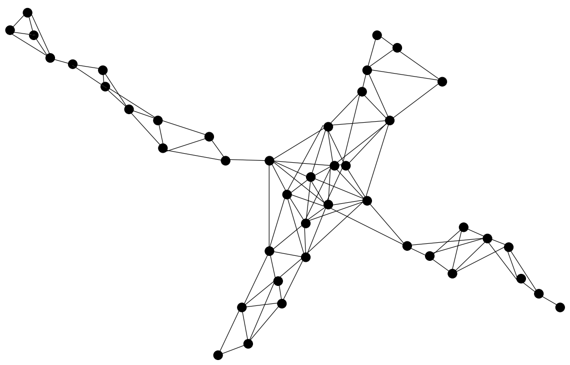


Figure 7.9: The 3-SIG with additional pruning.

where $d = \frac{\beta}{2}\|p - q\|$. The β-skeleton over V, $BG_\beta(V)$, is defined by the set of edges

$$E := \{pq \mid\mid U_\beta(p, q) \cap V = \varnothing\}$$

It is easy to see that $RNG(V) = BG_2(V)$ and $GG(V) = BG_1(V)$. Moreover, this family of proximity graphs is monotonic with respect to β, i. e., $\beta_1 > \beta_2 \Rightarrow BG_{\beta_1} \subset BG_{\beta_2}$. In other words, lower values of β give denser graphs.

7.1.2.5 Sphere-of-Influence graph

The sphere-of-influence graph (SIG) seems to be less well-known [Michael and Quint, 2003, Boyer et al., 2000, Jaromczyk and Toussaint, 1992].

While in the previous two graphs, the neighborhood is defined between *pairs* of points, here the neighborhood, its "sphere of influence", is defined for each point individually. More precisely, for each point $p \in V$, the radius r_p to its nearest neighbor is determined. Then, the SIG over V, $SIG(V)$, is defined by the set of edges

$$E := \{pq \mid\mid d(p, q) \leq r_p + r_q\}$$

The SIG tends to connect points that are "close" to each other relative to the local point density. In contrast, the RNG and the GG tend to connect points such that the overall edge length is small (think of cities and highways: it makes sense to connect two cities directly by a highway, *unless* there is a third one close by that could be connected as well by a small detour). The extreme along that line is the minimum spanning tree (see below) that has the least overall edge length.

Another difference is that the SIG, unlike the RNG or GG, can be disconnected (see Figure 7.7). This may or may not be desired, depending on the application.

A straight-forward way to reduce the likeliness or the number of "gaps" in the SIG is an extension, the r-SIG, $r \in \mathbb{N}$ [Klein and Zachmann, 2004a]. Here, the sphere-of-influence is not determined by the nearest neighbor, but by the r-th nearest neighbor. Obviously, the larger r, the more points are directly connected by an edge (see Figure 7.8).

A further extension can be applied to the r-SIG, if it is too dense or contains edges that are too long. This is *pruning* of edges based on a statistical outlier detection method over the lengths of edges [V. Barnett, 1994]. In statistics, an outlier is a single observation which is far away from the rest of the data. One definition of "far away" in this context is "larger than $Q_3 + 1.5 \cdot IQR$" where Q_3 is the third quartile (Q_2 would be the median), and IQR is the interquartile range $Q_3 - Q_1$. Our experiments showed that best results are achieved by pruning edges with length of at least $Q_3 + IQR$ [Klein and Zachmann, 2004a]. Figure 7.9 shows the pruned 3-SIG of the example point set.

Another extension could be to use ellipsoids instead of spheres [Klein and Zachmann, 2004a]. Then, we have the additional degree-of-freedom of the orientation of the ellipsoids. For instance, we could

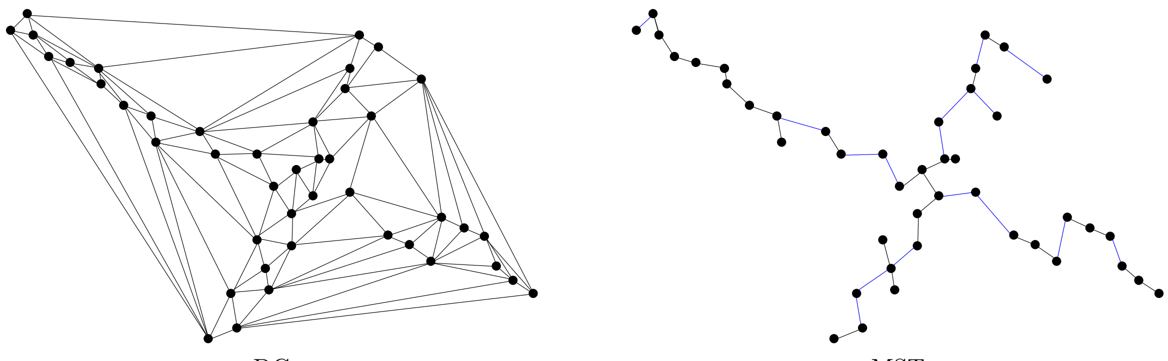


Figure 7.10: Example DG and MST for the same set of points as in Figure 7.7.

orient them along the direction of local largest variance. Depending on the application, this could have the advantage of better separating close sheets.

7.1.2.6 Other geometric graphs

There are other geometric graphs that are more or less closely related to proximity graphs, such as the minimum spanning tree and the Delaunay graph.

The *minimum spanning tree* (MST) spans (i.e., connects) all points by a tree (contains no cycles) of minimal length. Thus, although this tree can reveal interesting structure in the point set, there is no local proximity criterion for a pair of points.

The Delaunay graph (DG) is the dual of the Voronoi diagram (see Section 6.1.2). Alternatively, we can define the DG as follows.

Definition 6
Two points p and q are connected by an edge, if they satisfy the *empty circle property*. P and q satisfy the empty circle property iff there is a circle (or, in $\mathbb{R}^d$, a hypersphere) such that p and q are on its boundary and no point of V is in the interior of this circle.[2]

Actually, this definition is equivalent to the one given in Section 6.1.2. For the sake of uniqueness, we assume that all points are in *general position*, meaning that in $\mathbb{R}^d$ no $d+1$ points lie on a common hyperplane, and no $d+2$ points lie on a common hypersphere.

Figure 7.10 shows the DG and the MST for the same point set as before in Figure 7.7.

7.1.3 Inclusion property

Here, we demonstrate the following

Lemma 3
For a given set of points V,

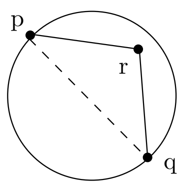
$$\mathsf{MST}(V) \subseteq \mathsf{RNG}(V) \subseteq \mathsf{GG}(V) \subseteq \mathsf{DG}(V).$$

This implies, in particular, that $\mathsf{RNG}(V)$ and $\mathsf{GG}(V)$ are connected, and that $|\mathsf{MST}(V)| \le |\mathsf{RNG}(V)| \le |\mathsf{GG}(V)| \le |\mathsf{DG}(V)|$. Furthermore, in $\mathbb{R}^2$, the number of edges is linear in the number of points.

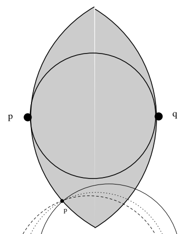
[2] Then, there is also a maximal empty hypersphere that has exactly $d+1$ points on its boundary, two of which are p and q.

```
construct DG(V), set GG := DG(V)
for all edges $pq \in$ GG do
    for all neighbors $r$ of $p$ or $q$ do
        if $r$ inside diameter circle of $p$ and $q$ then
            delete edge $pq$ from GG
        end if
    end for
end for
```

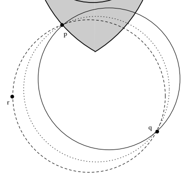
Algorithm 7.1: Simple algorithm to construct the Gabriel graph, starting from the Delaunay graph of a set of points.



The proof is fairly simple. Assume for the moment that $V \subseteq \mathbb{R}^2$. $\mathsf{MST}(V) \subseteq \mathsf{RNG}(V)$**:** Assume, for the sake of contradiction, that some edge pq is in the MST but not in the RNG. Then, the lune formed by p and q is not empty, i. e., there is some other point $r \in L(p,q)$. But then, $d(p,r) < d(p,q)$ and $d(q,r) < d(p,q)$. As a consequence, the original MST is not minimal, which can be seen as follows. The original MST cannot contain both eges pr and qr (otherwise it would contain a cycle). If edge pr is not there, we can replace pq by pr, and have constructed a tree with smaller total edge length. Similarly, we can replace pq by qr if edge qr is not there yet.



$\mathsf{RNG}(V) \subseteq \mathsf{GG}(V)$**:** Assume that pq is an edge in $\mathsf{RNG}(V)$. Then, the lune defined by p and q is empty, and so is the diametric circle defined by p and q, since it is contained in the lune.



$\mathsf{GG}(V) \subseteq \mathsf{DG}(V)$**:** Assume that pq is an edge in $\mathsf{GG}(V)$. Then, the diametric circle defined by p and q is empty. Thus, the pair (p,q) also satisfies the empty circle property as defined by Definiton 6. Usually, the DG has more edges than the GG. This can be seen as follows. Imagine that the circle can slide to the left or right, such that its diameter increases just so that p and q always stay on its boundary (of course, it won't be a diametric any more). Because of the definition of the GG, we can slide the circle, at least a little bit, and at least to one side, without hitting another point. Now, imagine that we slide the circle until it hits a third point, r. So we have now found two more pairs of points, (p,r) and (q,r), that satisfy the empty circle property.[3]

In higher dimensions, the proofs work just the same (with hyperspheres instead of circles).

7.1.4 Construction Algorithms

7.1.4.1 GG and RNG

We start with a simple algorithm for constructing the GG over a set V of points. From Lemma 3, we know that we can start with $\mathsf{DG}(V)$, and then remove those edges that are not neighbors according to the diameter circle property. In a brute force algorithm, we could check this property for an edge pq of $\mathsf{DG}(V)$ by just testing each point of V, whether it is inside the diameter circle of pq. A more efficient algorithm is to test only the neighbors of p, and those of q, against a candidate diameter circle. This is summarized in Algorithm 7.1.

Similarly, we can construct the $\mathsf{RNG}(V)$ by starting with the $\mathsf{DG}(V)$. Then we consider each edge pq of it in turn and check if there is any other point $r \in V$ that falls inside the lune made by p and q, i. e., if

$$d(p,r) < d(p,q) \ \wedge \ d(q,r) < d(p,q). \tag{7.1}$$

[3] Because we always assume V to be in general position (see Definiton 6).

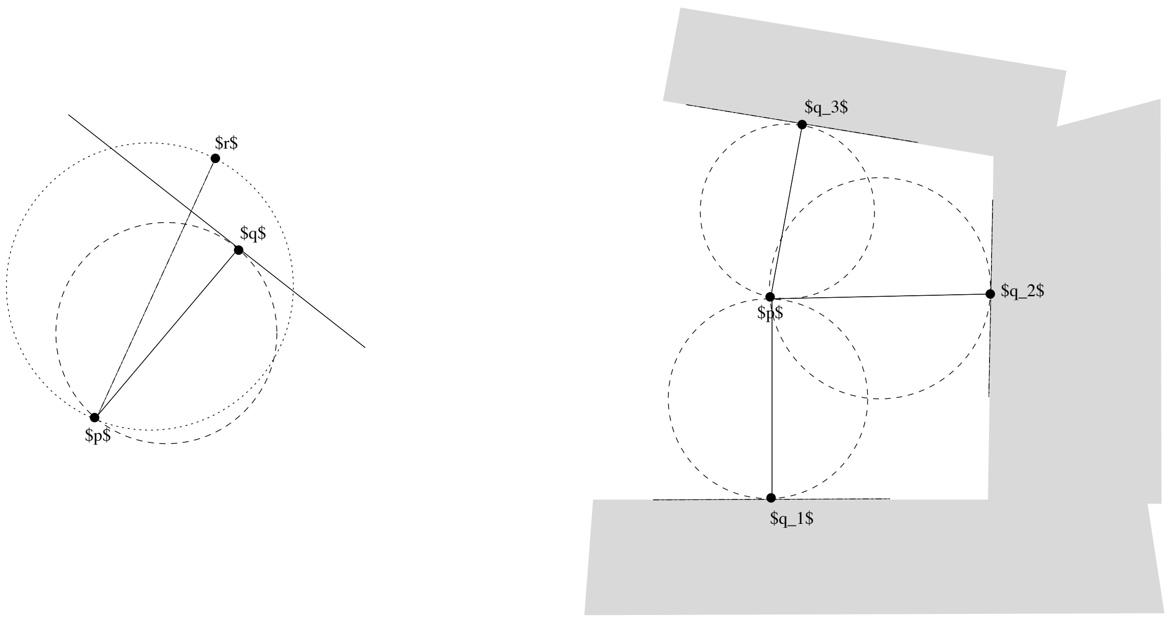


Figure 7.11: A simple heuristic leads to an $O(dn^2)$ algorithm to construct the Gabriel graph.

Another way to construct the GG would be the brute force approach. We can consider each potential pair of points (p, q) (there are $O(n^2)$ of them). For each of them, we check whether there is any other point r (there are $O(n)$) that is inside their diametric sphere. This test essentially involves two distance computations, i. e.,

$$\|p, r\|^2 + \|q, r\|^2 < \|p, q\|^2$$

which takes time $O(d)$ in d-dimensional space. Overall, this brute-force approach would take $O(dn^3)$.

We can easily improve upon this by the following observation. When checking the pair (p, q) whether they are neighbors in the GG, we must test whether any other point r is inside the diametric sphere (see Figure 7.11 left). At the same time, we can check whether r is in the right half-space of the plane H_{qp} that is orthogonal to the diameter pq and goes through q. If it is, then it cannot be a Gabriel neighbor to point p (because q would be inside the diametric sphere of pr).

So a better algorithm is the following. For each point p we keep a list of neighbor candidates, N_p (initially, this is the whole point set). Then, as we test $q_i \in N_p$ for being a real neighbor, we remove all r from N_p that are to the right of $H_{q_i p}$ (see Figure 7.11 right). This reduces the average compexity to $O(dn^2)$. Algorithm 7.2 summarizes this heuristic.

The algorithm to construct the relative neighborhood graph is analogous to the one for constructing the Gabriel graph.

7.1.4.2 SIG

The sphere-of-influence graph can be constructed much more efficiently than the previous one (on average), which is stated by the following

Lemma 4
The r-SIG can be determined in time $O(n)$ on average for uniformly and independently point-sampled models with size n in any fixed dimension. Moreover, it consumes only linear space in the worst case.

Proof. Dwyer [1995] proposed an algorithm to determine a SIG in linear time in the average case for uniform point clouds. As r is constant, this algorithm can easily be modified so that it can also compute the r-SIG in linear time. The algorithm consists of three steps.

```
for all $p \in V$ do
    $N_p \leftarrow V \setminus p$
    for all $q \in N_p$ do
        for all $r in V$ do
            if Eq. 7.1 true then
                consider next $q$
            else
                if $r$ is to the right of $H_{qp}$ then
                    remove $r$ from $N_p$
                end if
            end if
        end for
    end for
end for
```

Algorithm 7.2: Simple algorithm and heuristic to build the Gabriel graph.

First, the algorithm identifies the r-nearest neighbors of each point by utilizing the *spiral search* proposed by Bentley et al. [1980]: the space is subdivided into $O(n)$ hypercubic cells, the points are assigned to cells, and the r-nearest neighbors of each point p are found by searching the cells in increasing distance from the cell containing p. As $O(1)$ cells are searched for each point on average and a single query can be done in $O(1)$ [Bentley et al., 1980], this first step can be done in time $O(n)$.

Second, each point is inserted into every cell that intersects the r-nearest-neighbor sphere. On average, most spheres are small so that each point is inserted into a constant number of cells, and a constant number of points is inserted into each cell.

Finally, within each cell, all pairs of points that have been assigned to this cell are tested for intersection of their spheres-of-influence. Because each cell contains only a constant number of points, this can also be done in time $O(n)$.

Avis and Horton [1985] have shown that the 1-SIG has at most $c \cdot n$ edges where c is a constant. This c is always bounded by 17.5 [Avis and Horton, 1985, Edelsbrunner et al., 1989]. Guibas et al. [1992a] extended this result to the r-SIG over a point cloud from $\mathbb{R}^d$ and showed that the number of edges is bounded by $c_d \cdot r \cdot n$ where the constant c_d depends only on the dimension d. That means, the r-SIG consumes $O(n)$ space in the worst case. □

Moreover, as mentioned in Toussaint [1988], ElGindy has observed that the line-segment intersection algorithm introduced by Bentley and Ottmann [1979a] can be used to construct a SIG in the plane in $O(n \log n)$ time in the worst case. The algorithm of Guibas et al. [1992a] constructs the r-SIG in time

$$O(n^{2-\frac{2}{1+\lfloor(d+2)/\rfloor}+\epsilon} + rn\log^2 n),$$

for any $\epsilon > 0$ in the worst case.

7.2 Classification

7.2.1 Problem statement

Classification is a fundamental class of techniques that has applications in a huge number of areas, such as pattern recognition, machine learning, and robotics.

It is used to partition a "universe" of all possible objects, each described as a bunch of data, into a set of *classes*. It is mainly characterized by its *decision rule*. This rule determines, for a given object, which class it belongs to.

There are two fundamental classes of decision rules: *parametric* and *non-parametric* rules.

```
initialize grid with $n$ cells
for all $p \in V$ do
    assign $p$ to its grid cell
end for
for all $p \in V$ do
    find $r$-th nearest neighbor to $p$ by searching the grid cells in spiral order around $p$ with increasing
        distance
end for
for all $p \in V$ do
    for all cells around $p$ that intersect the sphere-of-influence around $p$ (in spiral order) do
        assign $p$ to cell
    end for
end for
for all cells in the grid do
    for all pairs $p_i, p_j$ of points assigned to the current cell do
        if spheres-of-influence of $p_i$ and $p_j$ intersect then
            create edge $p_i p_j$
        end if
    end for
end for
```

Algorithm 7.3: Simple algorithm to compute the r-SIG in $O(n)$ time on average.

Parametric decision rules make the membership classification based on the knowledge of the a priori probabilities of occurrence of objects belonging to some class C_i. This is captured by the probability density functions, $p(X|C_i)$, which characterize how probable is the measurement X should the membership class be C_i.

Very often, however, it is difficult to describe these probability density functions a priori, and often they are just unknown.

In such cases, non-parametric decision rules are attractive, because they do not require such a priori knowledge. Instead, these rules rely directly on the *training set* of objects. For this set, the class membership is known precisely for each object in this set. This is a priori knowledge, too, but much less difficult to acquire. (For instance, it could be provided by a human.) Therefore, this is also called *supervised learning*. The idea is, that a "teacher" feeds a number of example objects to the "student", and provides the correct answer for each of them. After the learning phase, the student tries to determine the correct answer for unseen objects, based on the examples it has seen so far.

Usually, objects are represented by a set of *features*, each of which can be represented usually by a real number. Thus, objects can be represented as points in the so-called *feature space*, and classes are subsets of this space. Since we work with points, we can usually define a measure of *distance* between points (which might be more or less well adapted to the objects).

A very simple, non-paramtric decision rule is the *nearest-neighbor classifier*. As its name suggests, it is based on the distance measure, and it assigns an unknown object to the same class as the nearest object from the (known) training set (see Figure 7.12). Most often, the Euclidean metric is used as a distance measure, although it is not always clear that this is best suited for the problem at hand.

Although this rule is extremely simple, it is fairly good (see Figure 7.14). More precisely, as the size of the trainig set goes to infinity, the asymptotic probability of error for the nearest-neighbor rule, P_e^{NN}, can be bounded by

$$\frac{P_e^{NN}}{P_e^{opt}} < 2 - P_e^{opt} \frac{N}{N-1}$$

where P_e^{opt} is the optimal, so-called Bayes probability of error, and N is the size of the trainig set [Cover and Hart, 1967]. In other words, the NN error is never two times worse than the optimal error.

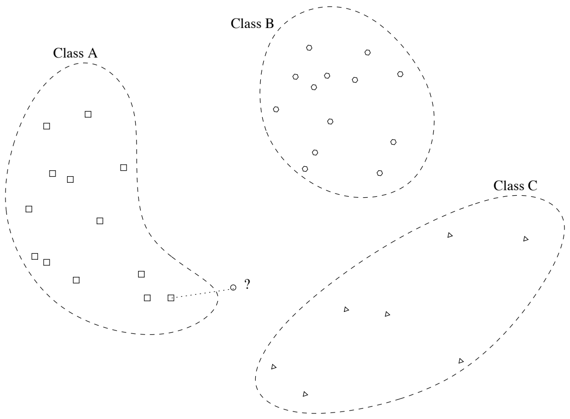


Figure 7.12: The nearest-neighbor yields a simple classifier.

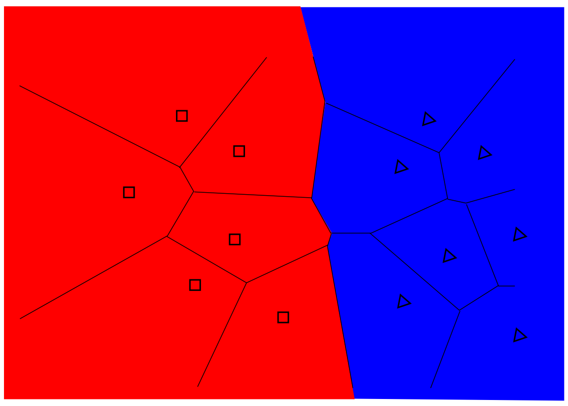


Figure 7.13: Conceptually, the nearest-neighbor classifier partitions the feature space into Voronoi cells.

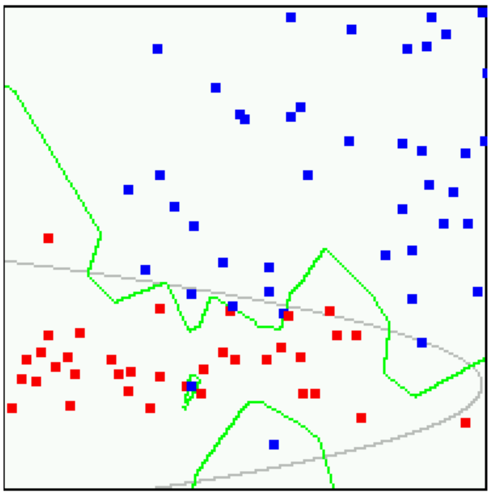


Figure 7.14: Comparison of the decision boundaries induced by the (optimal) Bayes classifier (squares), and the nearest-neighbor classifier (triangles).

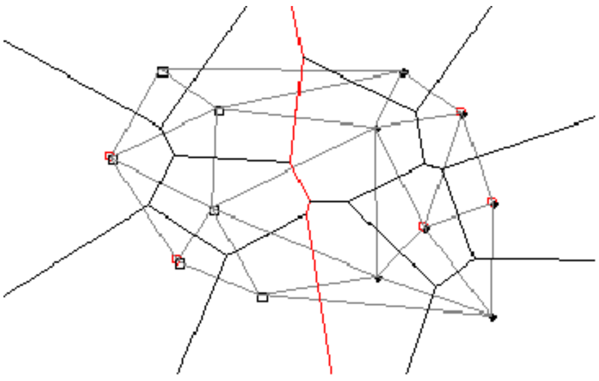


Figure 7.15: The decision boundary of the nearest-neighbor classifier runs between Delaunay neighbor with different color.

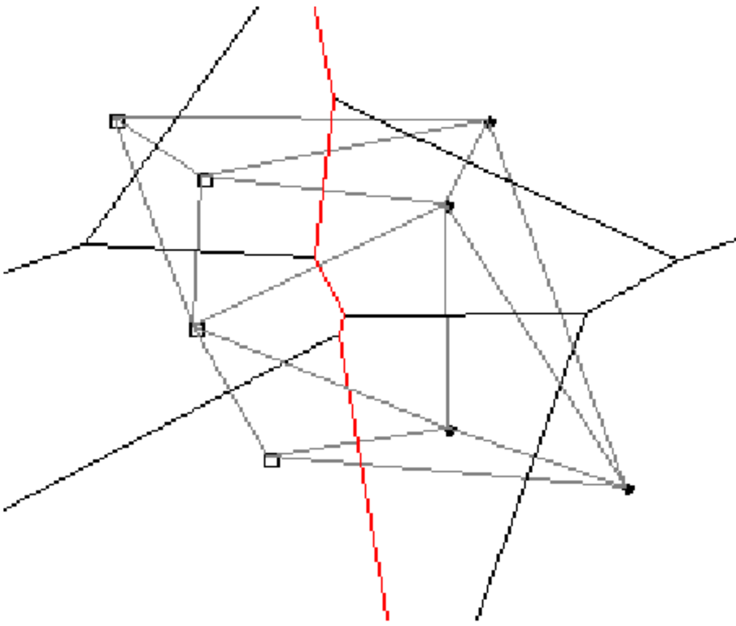


Figure 7.16: The edited training set has the same decision boundary, but less nodes.

Although the nearest-neighbor rule offers remarkably good performance (largely due to its simplicity), it is by no means a silver bullet. Some of its problems are:

- large space requirements (to store the complete trainging set);
- in high dimensions, it is very difficult to find the nearest neighbor in time better than $O(N)$ (this is called the *curse of dimensionality*.

In addition, there are two more problems that all classification methods must deal with

- classes may overlap, so there are regions in feature space that are populated by representatives from two or more classes;
- some representatives might be mis-labeled, i. e., they are classified into the wrong class (for instance, because they are outliers, or because the human generating the trainig set made a mistake.

In the next two sections, we will learn about some simple methods to correct some of these mistakes.

7.2.2 Editing and reducing the set

Imagine that the classes correspond to colors, i. e., each point in the training set is labeled with a color. The nearest-neighbor rule induces a partitioning of the feature space into a number of *cells*, each of which belongs to exactly one point of the training set. These cells are exactly the Voronoi cells of the Voronoi diagram induced by the training set (see Chapter 6 and Section 6.4.1).

So a class is exactly the union of all Voronoi cells that have the same color (see Figure 7.13). The *decision boundary* of a decision rule are those points in feature space that separate classes. More precisely, a point is on the decision boundary, iff any arbitrarily small, closed ball centered on this point contains points of two different classes. With the nearest-neighbor rule partitioning feature space into a set of Voronoi cells, the decision boundary are exactly those boundaries of the Voronoi cells that lie between cells of different colors.

In practice, the Voronoi diagram and the decision boundary are rarely computed explicitely, especially in higher dimensions. But it is clear that the decision boundary alone is sufficient for classification. So we can reduce the training set by removing those points that won't change the decision boundary. This is called *editing* the training set.

Instead of thinking in terms of Voronoi diagram and Voronoi cells, we can also think in terms of the Delaunay graph (see Figure 7.15). The decision boundary then runs between neighbors in this graph that have different color.

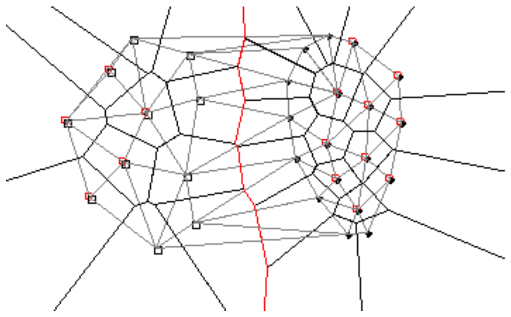


Figure 7.17: Delaunay editing often leaves too many points in the edited set. The unfilled points will be removed by Delaunay editing, but the contribution of the circled points is questionable.

With the Delaunay graph, it is very simple to edit the training set: we just remove all those nodes (i. e., points) that have all neighbors with the same color. This will change the Delaunay graph and the Voronoi diagram, but it will not change the decision boundary (see Figure 7.16).

7.2.3 Proximity graphs for editing

One problem with Delaunay editing is that it can still leave too many points (see Figure 7.17). These are usually points that are well separated from other classes, and that contribute only portions of the decision boundary very remote from the cluster of class representatives. Thus, these points are usually less important for the classification of unknown points, which can be expected to come from the same distribution as the training set.

Another problem is that computing the Delaunay graph takes, in the worst case, $\Theta(n^{\lceil d/2 \rceil})$, which is prohibitively expensive in higher dimensions (sometimes even in 3 dimensions).

So it makes sense to pursue the following, approximate solution [Bhattacharya et al., 1981]: compute a proximity graph over a given training set, which can be achieved more efficiently, edit the training set according to this graph, then classify unknown points using this edited training set. Of course, the decision boundary will be different, but hopefully not too much, so that classification of unknown points will still yield (mostly) the same result.

Figure 7.18 shows the same training set edited with the Delaunay graph (DG), the Gabriel graph (GG), and the RNG (see Section 7.1.2). Because the GG contains (usually) less edges than the DG, more nodes are marked having neighbors of the same color and are, thus, removed. So the GG usually yields a smaller training set than the DG. A similar relation holds between RNG and GG.

Consequently, the decision boundary changes, because it is, being induced by the nearest-neighbor rule, the boundary between Voronoi cells of different color in the Voronoi diagram over the edited training set. As you can see in Figure 7.18, the GG does not change the decision boundary much, but the RNG can change it quite substantially.

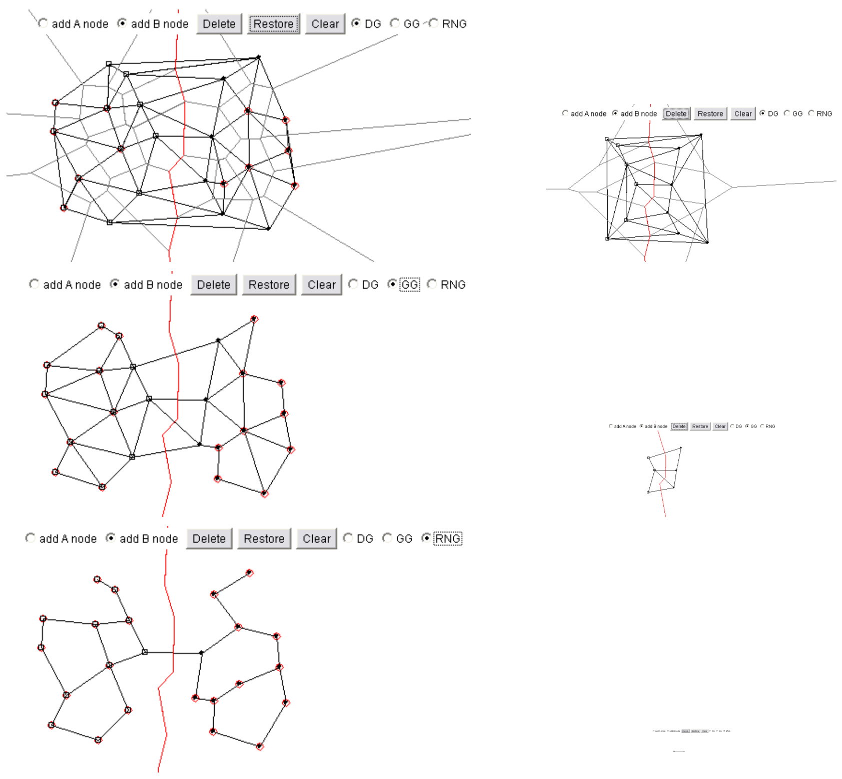


Figure 7.18: Different proximity graphs yield different edited (i. e., reduced) trainin sets and, thus, different approximations to the decision boundary

7.2.4 Cleaning the training set

In general, editing can serve several purposes. One of them is to reduce the size of the training set. This is the meaning we have understood so far. Another one is to remove samples from the training set that have been mis-labeled, that overlap with other classes (see Figure 7.14), or that are just outliers.

This task can be solved by utilizing proximity graphs, too. A very simple method will be described in the following [Sanchez et al., 1997].

The basic idea is to identify “bad” samples by examining their neighborhood. If most of the samples are correctly labeled, and the training set is sufficiently dense, then an incorrectly labeled sample will likely have a large number of differently labeled neighbor samples. This idea is summarized in Algorithm 7.4.

This algorithm can be refined straight-forwardly a little bit by modifying the rule when to mark a sample. For instance, we could mark a sample only if the number of neighbors in the winning class is at least 2/3 of the number of all neighbors. In addition, this method can be applied repeatedly to the training set.

7.3 Surfaces defined by point clouds

In the past few years, point clouds have had a renaissance caused by the wide-spread availability of 3D scanning technology (see Figure 7.19). Such devices produce a huge number of points, each of which lies on the surface of the real object in the scanner (see Figure 7.19). These sets of points are

```
build the proximity graph of the entire training set
for all samples in the set do
    find all its neighbors according to the proximity graph
    determine the most represented class (the “winning” class)
    mark the sample if the winning class is different than the label of the sample
end for
delete all marked samples from the set
```

Algorithm 7.4: Simple algorithm to clean (edit) a training set from incorrectly labeled samples.

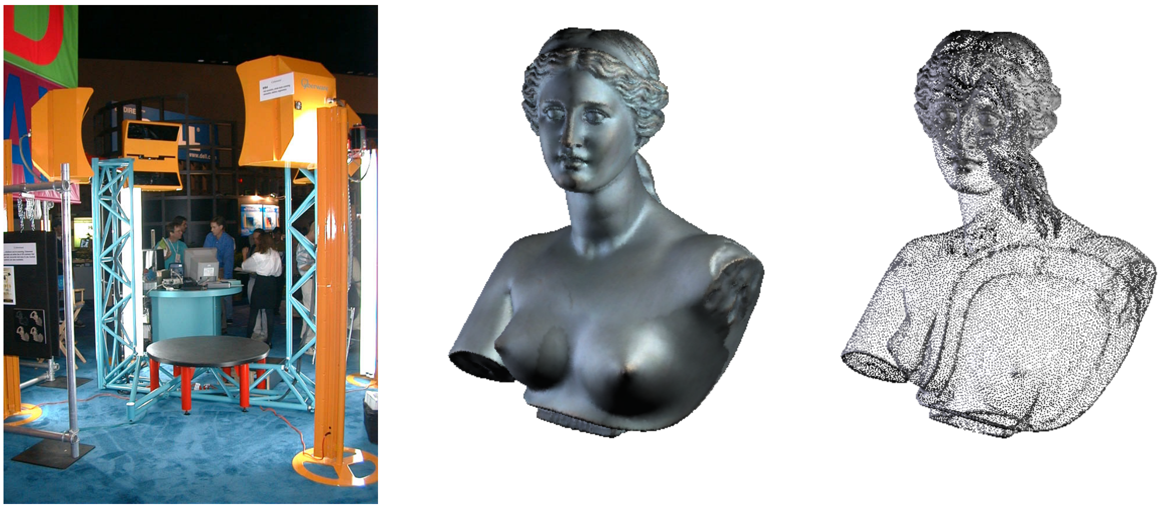


Figure 7.19: 3D scanners (left) produce large point clouds (right) from real objects (middle).

called *point clouds*. For one object there can be tens of millions of points. Usually, they are not ordered (or only partially ordered) and contain some noise.

In order to render [Pfister et al., 2000, Rusinkiewicz and Levoy, 2000, Zwicker et al., 2002, Bala et al., 2003] and interact [Klein and Zachmann, 2004b] with objects represented as point clouds, one must define an appropriate surface (even if it is not explicitly reconstructed).

This definition should produce a surface as close to the original surface as possible while being robust against noise (introduced by the scanning process). At the same time, it should allow to render and interact with the object as fast as possible.

In the following, we present a fairly simple definition [Klein and Zachmann, 2004a].

7.3.1 Implicit Surface Model

The surface definition begins with *weighted least squares interpolation* (WLS).

Let a point cloud $\mathcal{P}$ with N points $p_i \in \mathbb{R}^d$ be given. Then, an appealing definition of the surface from $\mathcal{P}$ is the zero set $S = \{x | f(x) = 0\}$ of an implicit function [Adamson and Alexa, 2003]

$$f(x) = n(x) \cdot (a(x) - x) \tag{7.2}$$

where $a(x)$ is the weighted average of all points $\mathcal{P}$

$$a(x) = \frac{\sum_{i=1}^{N} \theta(\|x - p_i\|) p_i}{\sum_{i=1}^{N} \theta(\|x - p_i\|)}. \tag{7.3}$$

Usually, a Gaussian *kernel* (weight function)

$$\theta(d) = e^{-d^2/h^2}, \quad d = \|x - p\|, \tag{7.4}$$

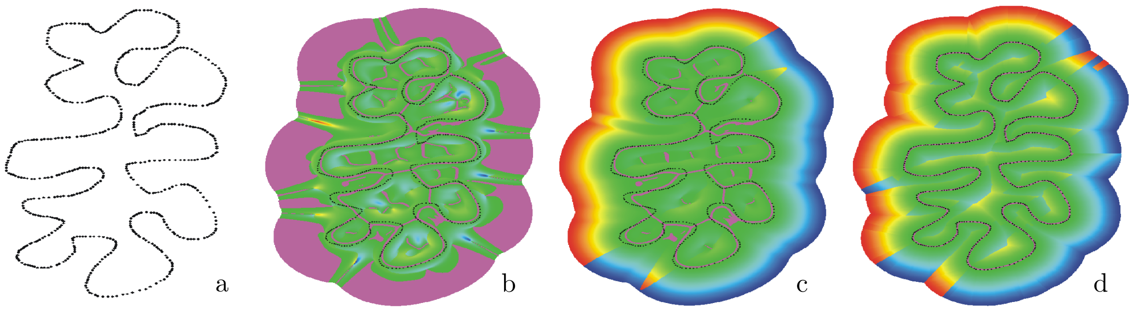


Figure 7.20: Visualization of the implicit function $f(x)$ over a 2D point cloud. Points $x \in \mathbb{R}^2$ with $f(x) \approx 0$, i.e., points on or close to the surface, are shown magenta. Red denotes $f(x) \gg 0$ and blue denotes $f(x) \ll 0$. (a) point cloud; (b) reconstructed surface using the definition of Adamson and Alexa [2003]; (c) utilizing the centered covariance matrix produces a better surface, but it still has several artifacts; (d) surface and function $f(x)$ based on the sphere-of-influence graph.

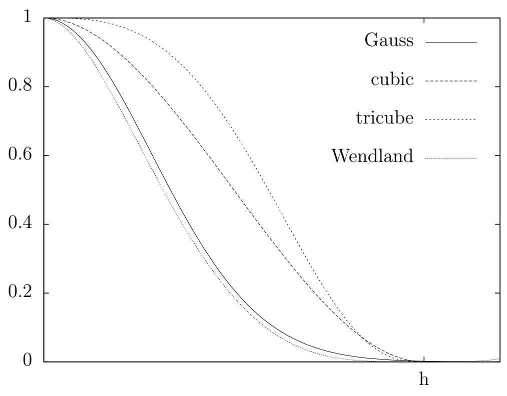


Figure 7.21: Different weight functions (kernels). Note that the horizontal scale of the Gauss curve is different, so that the qualitative shape of the curves can be compared better.

is used, but other kernels work as well (see below).

The bandwidth of the kernel, h, allows us to tune the decay of the influence of the points. It should be chosen such that no holes appear [Klein and Zachmann, 2004b].

Theoretically, θ's support is unbounded. However, it can be safely limited to the extent where it falls below the machine's precision, or some other, suitably small threshold θ_ε. Alternatively, one could use the cubic polynomial [Lee, 2000]

$$\theta(d) = 2\left(\frac{d}{h}\right)^3 - 3\left(\frac{d}{h}\right)^2 + 1,$$

or the tricube weight function [Cleveland and Loader, 1995]

$$\theta(d) = \left(1 - |\frac{d}{h}|^3\right)^3,$$

or the Wendland function [Wendland, 1995]

$$\theta(d) = \left(1 - \frac{d}{h}\right)^4 (4\frac{d}{h} + 1),$$

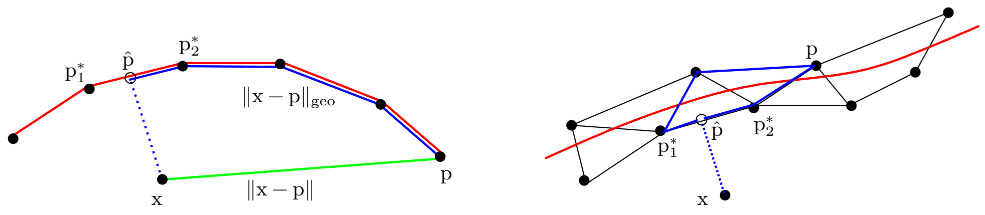


Figure 7.22: A proximity graph can help to improve the quality of the implicit surface by approximation of the geodesic distance instead of the Euclidean distance.

all of which are set to 0 for $d > h$ and, thus, have compact support (see Figure 7.21 for a comparison). However, the choice of kernel function is not critical [Härdle, 1990].

The normal $n(x)$ is determined by weighted least squares. It is defined as the direction of smallest weighted covariance, i. e., it minimizes

$$\sum_{i=1}^{N} \big(n(x)\cdot(a(x)-p_i)\big)^2 \theta(\|x-p_i\|) \tag{7.5}$$

for fixed x and under the constraint $\|n(x)\| = 1$.

Note that, unlike Adamson and Alexa [2003], we use $a(x)$ as the center of the *principal component analysis* (PCA), which makes $f(x)$ much more well-behaved (see Figure 7.20). Also, we do not solve a minimization problem like Levin [2003], Alexa et al. [2003], because we are aiming at an extremely fast method.

The normal $n(x)$ defined by (7.5) is the smallest eigenvector of the centered covariance matrix $B = (b_{ij})$ with

$$b_{ij} = \sum_{k=1}^{N} \theta(\|x-p_k\|)(p_{k_i} - a(x)_i)(p_{k_j} - a(x)_j). \tag{7.6}$$

There are several variations of this simple definition, but for sake of clarity, we will stay with this basic one.

7.3.2 Euclidean Kernel

This simple defintion (and many of its variants) has several problems. One problem is that the Euclidean distance $\|x - p\|$, $p \in \mathcal{P}$, can be small, while the distance from x to the closest point on S and then along the shortest path to p on S (the geodesic) is quite large.

This can produce artifacts in the surface S (see Figure 7.20); two typical cases are as follows. First, assume x is halfway between two (possibly unconnected) components of the point cloud; then it is still influenced by *both* parts of the point cloud, which have similar weights in Equ. 7.3 and 7.5. This can lead to an *artificial* zero subset $\subset$ S where there are no points from $\mathcal{P}$ at all. Second, let us assume that x is inside a cavity of the point cloud. Then, $a(x)$ gets “drawn” closer to x than if the point cloud was flat. This makes the zero set *biased* towards the “outside” of the cavity, away from the true surface. In the extreme, this can lead to cancellation near the center of a spherical point cloud, where all points on the sphere have a similar weight.

This thwarts algorithms based solely on the point cloud representation, such as collision detection [Klein and Zachmann, 2004b] or ray-tracing [Adamson and Alexa, 2004].

The problems just mentioned could be alleviated somewhat by restricting the surface to the region $\{x : \|x - a(x)\| < c\}$ (since $a(x)$ must stay within the convex hull of $\mathcal{P}$). However, this does not help in many cases involving cavities.

7.3.3 Geodesic Distance Approximation

As illustrated above, the main problems are caused by a distance function that does not take the topology of S into account. Therefore, we will try to approximate the geodesic distances on the surface S. Unfortunately, we do not have an explicit reconstruction of S, and in many applications, we do not even want to construct one. Instead, we utilize a geometric proximity graph where the nodes are points $\in \mathcal{P}$.

In principle, we could utilize any proximity graph, but the SIG seems to yield the best results.

We define a new distance function $d_{\text{geo}}(x, p)$ as follows. Given some location x, we compute its nearest neighbor $p_1^* \in \mathcal{P}$. Then, we compute the closest point $\hat{p}$ to x that lies on an edge adjacent to p_1^*. Now, conceptually, the distance from x to any $p \in \mathcal{P}$ could be defined as

$$d_{\text{geo}}(x, p) = \min_{n \in \{p_1^*, p_2^*\}} \left\{ d(n, p) + \|\hat{p} - n\| \right\},$$

where $d(p^*, p)$ for any $p \in \mathcal{P}$ is the accumulated length of the shortest path from p^* to p, multiplied by the number of "hops" along the path (see Figure 7.22 left). However, it is not obvious that this is always the desired distance. In addition, it is desirable to define the distance function with as few discontinuities as possible. Therefore, we take the weighted average (see Figure 7.22 right)

$$\begin{aligned} d_{\text{geo}}(x, p) = \; & (1 - a)\big(d(p_1^*, p) + \|\hat{p} - p_1^*\|\big) \\ & + a\big(d(p_2^*, p) + \|\hat{p} - p_2^*\|\big) \end{aligned} \tag{7.7}$$

with the interpolation parameter $a = \|\hat{p} - p_1^*\|$.

Note that we do *not* add $\|x - \hat{p}\|$. That way, $f(x)$ is non-zero even far away from the point cloud.

Of course, there are still discontinuities in d_{geo} and thus in function f. These can occur at the borders of the Voronoi regions of the cloud points, in particular at borders where the Voronoi sites are far apart from each other, such as points close to the medial axis.

The rationale for multiplying the path length by the number of hops is the following: if an (indirect) neighbor p is reached by a shortest path with many hops, then there are many points in $\mathcal{P}$ that should be weighted much more than p, even if the Euclidean distance $\|p^* - p\|$ is small. This is independent of the concrete proximity graph used for computing the shortest paths.

Overall, when computing f by (7.2)–(7.6), we use d_{geo} in (7.4).

7.3.4 Automatic bandwidth computation

Another problem of the simple surface definition (without proximity graph) is the bandwidth h in (7.4), which is a critical parameter.

On the one hand, if h is chosen too small, then the variance may be too large, i. e., noise, holes, or other artifacts may appear in the surface. On the other hand, if it is chosen too large, then the bias may be too large, i. e., small features in the surface will be smoothed out. To overcome this problem, Pauly et al. [2002] proposed to scale the parameter h adaptively.

Here, we can use the proximity graph (e. g., a SIG) to estimate the local sampling density, $r(x)$, and then determine h accordingly. Thus, h itself is a function $h = h(x)$.

Let r_1 and r_2 be the lengths of the longest edges incident to p_1^* and p_2^*, resp., (see Figure 7.22). Then, we set

$$r(x) = \frac{1}{r} \cdot \frac{\|\hat{p} - p_2^*\| \cdot r_1 + \|\hat{p} - p_1^*\| \cdot r_2}{\|p_2^* - p_1^*\|} \tag{7.8}$$

$$h(x) = \frac{\eta \, r(x)}{\sqrt{-\log \theta_\varepsilon}} \tag{7.9}$$

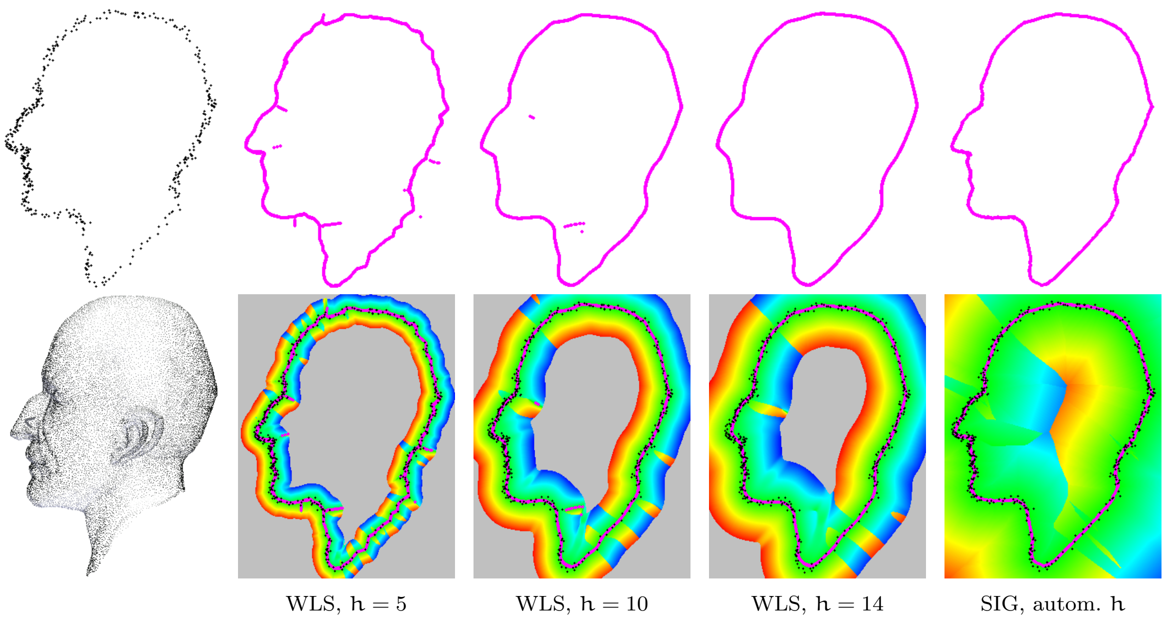


Figure 7.23: Reconstructed surface based on simple WLS and with proximity graph (rightmost) for a noisy point cloud obtained from the 3D Max Planck model (leftmost). Notice how fine details as well as sparsely sampled areas are handled without manual tuning.

where θ_ε is a suitably small value (see Section 7.3.1), and r is the number of nearest neighbors that determine the radius of each sphere-of-influence. (Note that $\log \theta_\varepsilon < 0$ for realistic values of θ_ε.) Thus, p_i with distance $\eta\, r(x)$ from $\hat{p}$ will be assigned a weight of θ_ε (see Equ. 7.4).

We have now replaced the scale- and sampling-dependent parameter h by another one, η, that is *independent* of scale and sampling density. Often, this can just be set to 1, or it can be used to adjust the amount of "smoothing". Note that this automatic bandwidth detection works similarly for many other kernels as well (see Section 7.3.1).

Depending on the application, it might be desirable to involve more and more points in (7.2), so that $n(x)$ becomes a least squares plane over the complete point set $\mathcal{P}$ as x approaches infinity. In that case, we can just add $\|x - \hat{p}\|$ to (7.8).

Figure 7.23 shows that the automatic bandwidth determination allows the WLS approach to handle point clouds with varying sampling densities without any manual tuning: the smoothing of the different sampling densities is very similar, compared to the "scale" (i. e., density). Notice the fine detail in the range from the tip of the nose throughout the chin, and the relatively sparse sampling in the area of the skull and at the bottom.

7.3.5 Automatic boundary detection

Another benefit of the automatic sampling density estimation is a very simple boundary detection method. This method builds on the one proposed by Adamson and Alexa [2004].

The idea is simply to discard points x with $f(x) = 0$, if they are "too far away" from $a(x)$ *relative* to the sampling density in the vicinity of $a(x)$. More precisely, we define a new implicit function

$$\hat{f}(x) = \begin{cases} f(x), & \text{if } |f(x)| > \varepsilon \vee \|x - a(x)\| < 2r(x) \\ \|x - a(x)\|, & \text{else} \end{cases} \tag{7.10}$$

Figure 7.24, right, shows that this simple method is able to handle different sampling densities fairly well.

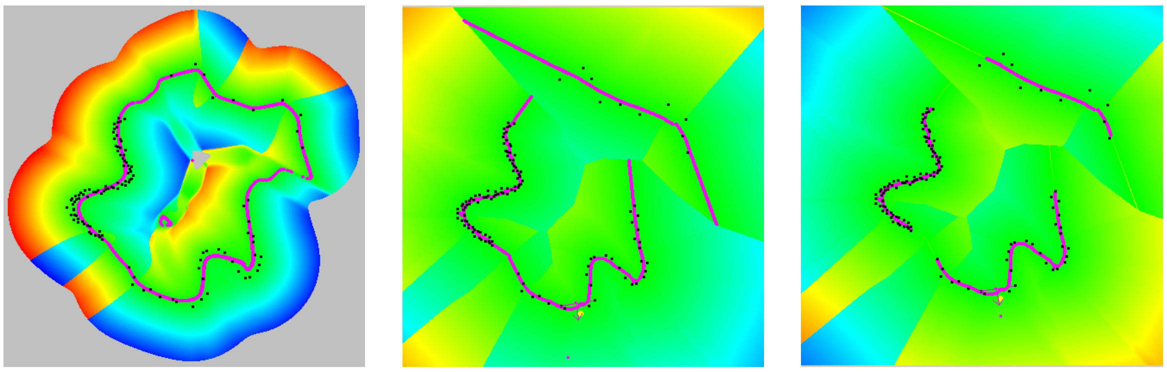


Figure 7.24: Automatic sampling density estimation for each point, allows to determine the bandwidth automatically and independently of scale and sampling-density (middle), and to detect boundaries automatically (right).

7.3.6 Complexity of function evaluation

The geodesic kernel needs to determine the nearest neighbor p^* of a point x in space (see Section Section 6.4.1). Using a simple kd-tree, an approximate nearest neighbor in 3D can be found in $O(\log^3 N)$ [Arya et al., 1998].[4]

Klein and Zachmann [2004a] show that all points p_i influencing x can be determined in constant time by a depth-first or breadth-first search through the graph.

Overall, $f(x)$ can be determined in $O(\log^3 N)$.

In order to achieve also a fast practical function evaluation, we also need to compute the smallest eigenvector quickly [Klein and Zachmann, 2004b]. First, we compute the smallest eigenvalue λ_1 by determining the three roots of the cubic characteristic polynomial $\det(B - \lambda I)$ of the covariance matrix B [Press et al., 1992]. Then, we compute the associated eigenvector using the Cholesky decomposition of $B - \lambda_1 I$.[5]

In our experience, this method is faster than the Jacobi method by a factor of 4, and it is faster than singular value decomposition by a factor 8.

7.4 Intersection Detection between Point Clouds

The surface defintion described in the previous sections is well suited for quick rendering, for instance, by ray-tracing. In addition, it can also be used to determine the intersection between two point cloud objects [Klein and Zachmann, 2005]. This will be explained in the following sections.

The problem is the following. Given two point clouds A and B, the goal is to determine whether or not there is an intersection, i. e., a common root $f_A(x) = f_B(x) = 0$, and, possibly, to compute a sampling of the intersection curve(s), i. e., of the set $\mathcal{Z} = \{x \mid f_A(x) = f_B(x) = 0\}$.

In principle, one could just utilize one of the many general root finding algorithms [Pauly et al., 2003, Press et al., 1992]. However, finding common roots of two (or more) nonlinear functions is extremely difficult. Even more so here, because the functions are not described analytically, but algorithmically.

---

[4] Under mild conditions, nearest neighbor can be done in $O(\log N)$ time by utilizing a Delaunay hierarchy [Devillers, 2002], but this may not always be practical.

[5] The second step is possible because $B - \lambda I$ is positive semi-definite. Let $\lambda(A) = \{\lambda_1, \lambda_2, \lambda_3\}$ with $0 < \lambda_1 \leq \lambda_2 \leq \lambda_3$. Then, $\lambda(B - \lambda I) = \{0, \lambda_2 - \lambda_1, \lambda_3 - \lambda_1\}$. Let $B - \lambda I = USU^T$ be the singular value decomposition. Then, $x^T(B - \lambda I)x = x^T USU^T x = y^T Sy \geq 0$.

Thus, the Cholesky decomposition can be performed if full pivoting is done [Higham, 1990].

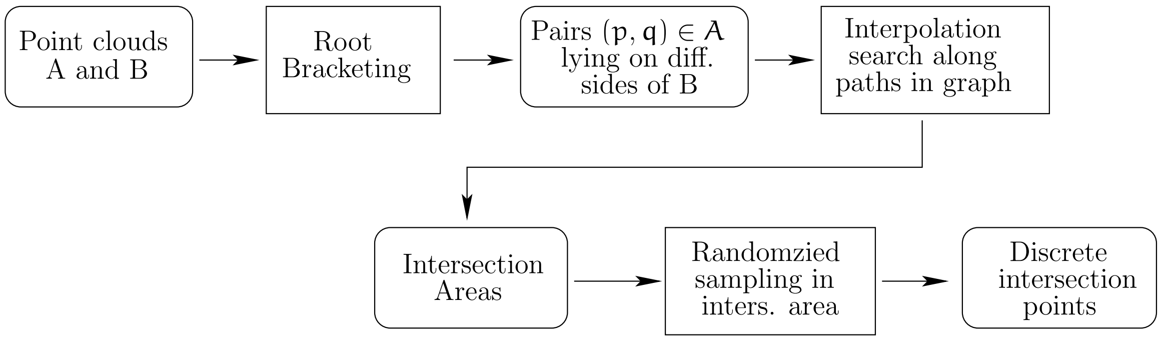

Point clouds A and B
Root Bracketing
Pairs $(p, q) \in A$ lying on diff. sides of B
Interpolation search along paths in graph
Intersection Areas
Randomzied sampling in inters. area
Discrete intersection points

Figure 7.25: Outline of the point cloud intersection detection.

Fortunately, here we can exploit the proximity graph we already have in the surface defintion of the objects and, thus, achieve much better performance.

First, the algorithm tries to bracket intersections by two points on one surface and on either side of the other surface (see Figure 7.25). Second, for each such bracket, it finds an approximate point in one of the point clouds that is close to the intersection (see Figure 7.26). Finally, this approximate intersection point is refined by subsequent randomized sampling. This last step is optional, depending on the accuracy needed by the application.

Steps 1 and 3 will be solved by a randomization approach. Both steps 1 and 2 will utilize the proximity graph. In the following, we describe each step in detail.

7.4.1 Root Bracketing

As mentioned before, the algorithm starts by constructing random pairs of points on different sides of one of the surfaces. The two points should not be too far apart, and, in addition, the pairs should evenly sample the surface.

An exhaustive enumeration of all pairs is, of course, prohibitively expensive. Therefore, we pursue the following, randomized (sub-)sampling procedure.

Assume that the implicit surface is conceptually(!) approximated by surfels (2D discs) of equal size [Pfister et al., 2000, Rusinkiewicz and Levoy, 2000]. Let $\text{Box}(A, B) = \text{Box}(A) \cap \text{Box}(B)$ and $\overline{A} = A \cap \text{Box}(A, B)$. Then, we want to randomly draw points $p_i \in \overline{A}$ such that each surfel s_i gets occupied by at least one p_i; here, "occupied by p_i" means that the projection of $a(p_i)$ along the normal $n(p_i)$ onto the supporting plane of s_i lies within the surfel's radius.

For each p_i we can easily determine another point $p_j \in \overline{A}$ in the neighborhood of p_i (if any), so that p_i and p_j lie on different sides of f_B. We represent the neighborhood of a point p_i by a sphere C_i centered at p_i.

An advantage of this is that the application can specify the density of the intersection points that are to be returned by our algorithm. From these, it is fairly easy to construct a discretization of the complete intersection curves (e. g., by utilizing randomized sampling again).

Note that we never need to actually construct the surfels, or assign the points from A explicitly to the neighborhoods, which we describe in the following. Section 7.4.2 describes how to choose the radius of the spheres C_i.

In order to find a $p_j \in A \cap C_i$ on the "other side" of f_B, we use $f_B(p_i) \cdot f_B(p_j) \leq 0$ as an indicator. This, of course, is reliable only if the normals $n(x)$ are consistent throughout space. If the surface is manifold, this can be achieved by a method similar to Hoppe et al. [1992]: utilizing a proximity graph (such as the SIG), we can propagate a normal to each point $p_i \in A$. Then, when defining $f(x)$, we choose the direction of $n(x)$ according to the normal stored with the nearest neighbor of x.[6]

[6] Surprisingly, the direction of $n(x)$ is consistent over fairly large volumes without any preconditioning.

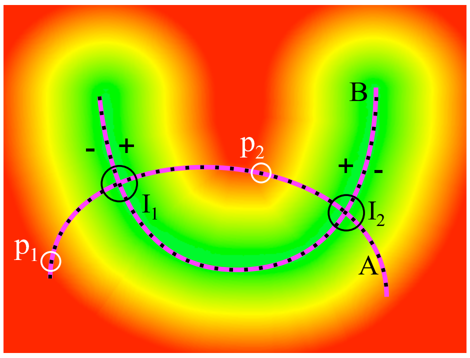


Figure 7.26: Two point clouds A and B and their intersection spheres I_1 and I_2. The root finding procedure, when initialized with $p_1, p_2 \in A$, will find an approximate intersection point inside the *intersection sphere* I_1.

In order to sample A such that each (conceptual) surfel is represented by at least one point in the sample, we use the following

Lemma 5
Let A be a uniformly sampled point cloud. Further, let S_A denote the set of conceptual surfels approximating the surface of A inside the intersection volume of A and B, and let $a = |S_A|$. Then, we can occupy each surfel with at least one point with probability $p = e^{-e^{-c}}$, where c is an arbitrary constant, just by drawing $n = O(a \ln a + c a)$ random and independent points from $\overline{A}$. These points will be denoted as A'.

Proof: see Klein and Zachmann [2005].

For instance, if we want $p \geq 97\%$, we have to choose $c = 3.5$, and if $a = 30$, then we have to draw $n \approx 200$ random points.

The next section will show how to choose an appropriate size for the neighborhoods C_i. After that, Section 7.4.3 will propose an efficient way to determine the other part p_j of the root brackets, given a point $p_i \in A'$.

7.4.2 Size of Neighborhoods

The radius of the spherical neighborhoods C_i has to be chosen so that, on the one hand, all C_i cover the whole surface defined by A. On the other hand, the intersection with each adjoining neighborhood of C_i has to contain at least one point in A' so as to not miss any collisions lying in the intersection of two neighborhoods. The situation is illustrated in Figure 7.27.

To determine the minimal radius of a spherical neighborhood C_i, we introduce the notion of *sampling radius*.

Definition 7 (Sampling radius)
Let a point cloud A as well as a subset $A' \subseteq A$ be given. Consider a set of spheres, centered at A', that cover the surface defined by A (not A'), where all spheres have equal radius. We define the sampling radius $r(A')$ as the minimal radius of such a sphere covering.

The sampling radius $r(A')$ can obviously be estimated as the radius r of a surfel $s_i \in S_A$.

Let F_A denote the surface area of the implicit surface over $\overline{A}$. Then, the surfel radius r can be determined by

$$\frac{F_A}{a} = \pi r^2 \Rightarrow r = \sqrt{\frac{F_A}{a\pi}}.$$

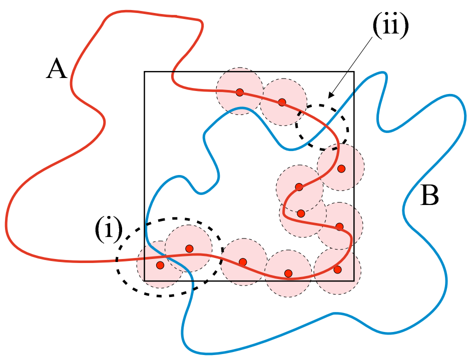


Figure 7.27: If the spherical neighborhoods C_i (filled disks) are too small, not all collisions can be found. (i) Adjoining neighborhoods might not overlap sufficiently, so their intersection contains no randomly chosen cloud point. (ii) The surface might not be covered by neighborhoods C_i.

Assume that the implicit surface over $\overline{A}$ can also be approximated by surfels of size $r(A)$. Then, F_A can be estimated by

$$F_A = |\overline{A}| \cdot \pi r(A)^2.$$

Overall, $r(A')$ can be estimated by

$$r(A') = r(A) \cdot \sqrt{\frac{|\overline{A}|}{a}} \approx r(A) \cdot \sqrt{\frac{\mathrm{Vol}(A,B)}{\mathrm{Vol}(A) \cdot a} \cdot |A|}\,.$$

The size of $\overline{A}$, $|\overline{A}|$, can easily be estimated as $|A|\frac{\mathrm{Vol}(A)}{\mathrm{Vol}(A,B)}$, and the sampling radius $r(A)$ can easily be determined in the preprocessing.

7.4.3 Completing the Brackets

Given a point $p_i \in A'$, we have to determine other points $p_j \in A \cap C_i$ on the other side of f_B in order to bracket the intersections. From a theoretical point of view, this could be done by testing $f_B(p_i) \cdot f_B(p_j) \leq 0$ for all points $p_j \in A' \cap C_i$ in time $O(1)$ because $|A'|$ can be chosen constant. In practice however, the set $A' \cap C_i$ cannot be determined quickly. Therefore, in the following, we propose an adequate alternative that works in time $O(\log \log N)$.

We observe that $A' \cap C_i \approx A' \cap A_i$, where $A_i := \{x \mid 2r(A') - \delta \leq \|x - p_i\| \leq 2r(A')\}$ is an *anulus* around p_i (or, at least, these are the p_j that we need to consider to ensure a certain bracket density). By construction of A', $A' \cap A_i$ has a similar distribution as $A \cap A_i$. Observe further, that we don't necessarily need $p_j \in A'$.

Overall, the idea is to construct a random sample $B_i \subset A \cap C_i$ such that $B_i \subset A_i$, $|B_i| \approx |A' \cap A_i|$, and such that B_i has a similar distribution as $A' \cap A_i$.

This sample B_i can be constructed quickly by the help of Lemma 5: we just choose randomly $O(b \ln b)$ many points from $A \cap A_i$, where $b := |A' \cap C_i|$.

We can describe the set $A \cap A_i$ very quickly, if the points in the CPSP map stored with p_i are sorted by their geodesic[7] distance from p_i. Then we just need to use interpolation search to find the first point with distance $2r(A') - \delta$ and the last point with distance $2r(A')$ from p_i. This can be done in time $O(\log \log |A \cap C_i|)$ per point $p_i \in A'$. Thus, the overall time to construct all brackets is in $O(\log \log N)$.

[7] By using the geodesic distance (or, rather, the approximation thereof) we basically impose a different topology on the space where A is embedded, but this is actually desirable.

```
$l, r = 1, n$
$d_{l,r} = f_B(P_1), f_B(P_n)$
**while** $|d_l| > \epsilon$ and $|d_r| > \epsilon$ and $l < r$ **do**
    $x = l + \lceil \frac{-d_l}{d_r - d_l}(r - l) \rceil$ {*}
    $d_x = f_B(P_x)$
    **if** $d_x < 0$ **then**
        $l, r = x, r$
    **else**
        $l, r = l, x$
    **end if**
**end while**
```

Algorithm 7.5: Pseudo-code for the root finding algorithm based on interpolation search. P is an array containing the points of the shortest path from $p_1 = P_1$ to $p_2 = P_n$, which can be precomputed. $d_i = f_B(P_i)$ approximates the distance of P_i to object B. (*) Note that either d_l or d_r is negative.

7.4.4 Interpolation Search

Having determined two points $p_1, p_2 \in A$ on different sides of surface B, the next goal is to find a point $\hat{p} \in A$ "between" p_1 and p_2 that is "as close as possible" to B. In the following, we will call such a point *approximate intersection point* (AIP). The true intersection curve $f_B(x) = f_A(x) = 0$ will pass close to $\hat{p}$ (usually, it does not pass through any points of the point clouds).

Depending on the application, $\hat{p}$ might already suffice. If the true intersection points are needed, then we refine the output of the interpolation search by the procedure described in Section 7.4.6.

Here, we can exploit the proximity graph: we just consider the points P_{12} that are on the shortest path between p_1 and p_2, and we look for $\hat{p}$ that assumes $\min_{p \in P_{12}}\{|f(p)|\}$.

Let us assume that f_B is monotonic along the path $\overline{p_1 p_2}$. Then, instead of doing an exhaustive search along the path, we can utilize interpolation search to look for $\hat{p}$ with $f(\hat{p}) = 0$.[8] This makes sense here, because the "access" to the key of an element, i.e., an evaluation of $f_B(x)$, is fairly expensive [Sedgewick, 1989]. The average runtime of interpolation search is in $O(\log \log m)$, m = number of elements.

Algorithm 7.5 for the interpolation search assumes that the shortest paths are precomputed and stored in a map.

However, in practice, the memory consumption, albeit linear, could be too large for huge point clouds. In that case, we can compute the path P on-the-fly at runtime by Algorithm 7.6. Theoretically speaking, the overall algorithm is now in linear time. However, in practice, it still behaves sublinear because the reconstruction of the path is negligible compared to evaluating f_B.

If f_B is not monotonic along the paths between the brackets, but the sign of $f_B(x)$ is consistent, then we can utilize binary search to find $\hat{p}$. (The complexity in that case is, of course, $O(\log m$.)

7.4.5 Models with Boundaries

If the models have boundaries and the sampling rate of the root bracketing algorithm is too low, not all intersections will be found (see Figure 7.29). In that case, some AIPs might not be reached, because they are not connected through the proximity graph.

Therefore, we modify the r-SIG for our purpose a little bit. After constructing the graph, we usually prune away all "long" edges by an outlier detection algorithm (see Section 7.1.2.5). Now, we only mark these edges as "virtual". Thus, we can still use the r-SIG for defining the surface as before. For our interpolation search, however, we can also use the "virtual" edges so that small holes in the model are bridged.

[8] In practice, the interpolation search will never find exactly such a $\hat{p}$, but instead a pair of adjacent points on the path that straddle B

```
q.insert($p_1$);   clear P
repeat
    $p = q.pop$
    P.append( p )
    for all  $p_i$ adjacent to p  do
        if  $d_{geo}(p_i, p_2) < d_{geo}(p_1, p_2)$  then
            insert $p_i$ into q with priority $d_{geo}(p_i, p_2)$
        end if
    end for
until  $p = p_2$
```

Algorithm 7.6: This algorithm can be used to initialize P for Algorithm 7.5 if precomputing and storing all shortest paths in a map is too expensive. (q is a priority queue.)

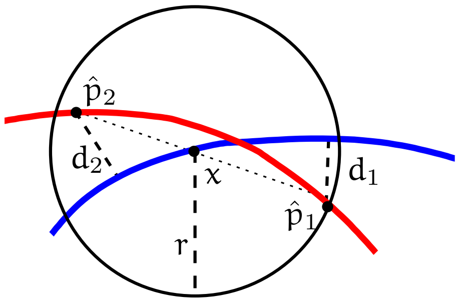


Figure 7.28: An intersection sphere centered at an AIP p_i. Its radius r can be determined approximately with the help of a second point on the other side of B.

7.4.6 Precise Intersection Points

If two point clouds are intersecting, the interpolation search computes a set of AIPs. Around each of them, an *intersection sphere* of radius $r = \min(\|x - \hat{p}_1\|, \|x - \hat{p}_2\|)$ contains a true intersection point, where

$$x = \frac{1}{d_1 + d_2}(d_2 p_1 + d_1 p_2),$$

the $\hat{p}_i$ have been computed by the interpolation search, lying on different sides of surface B, and $d_i = f_B(p_i)$. This idea is illustrated in Figure 7.28. So if the AIPs are not precise enough, we can sample each such sphere to get more accurate (discrete) intersection points.

More precisely, if we want a precise collision point's distance from the surfaces to be smaller than ϵ_2, we cover a given intersection sphere by s smaller spheres with *diameter* ϵ_2 and sample that volume by $s \ln s + cs$ many points so that each of the s spheres gets a point with high probability. For each of these, we just determine the distance to both surfaces.

Rogers [1963] showed that a sphere with radius $a{\cdot}b$ can be covered by at most $s = \lceil \sqrt{3}\, a \rceil^3$ smaller spheres of radius b. Since we would like to cover the intersection sphere by spheres with radius $b = \epsilon_2/2$, we have to choose $a = 2r/\epsilon_2$, so that $a{\cdot}b = r$. As a consequence,

$$s = \lceil \sqrt{3}\, \frac{2r}{\epsilon_2} \rceil^3.$$

For example, to cover an intersection sphere with spheres of *radius* ϵ, then $\epsilon_2 = 2\epsilon$ and $s = \lceil \sqrt{3}\, r/\epsilon \rceil^3$.

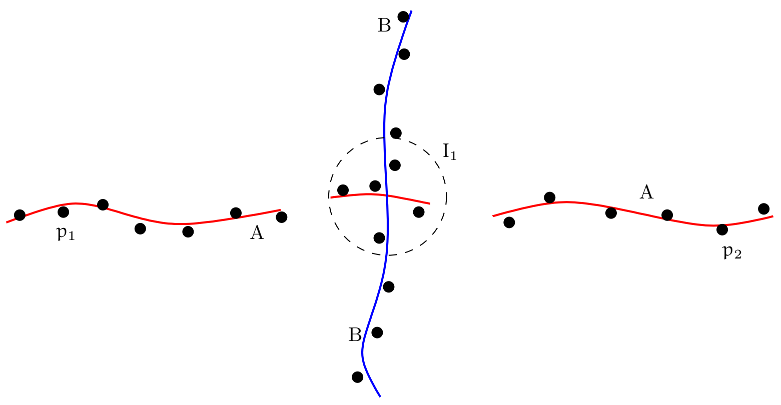


Figure 7.29: Models with boundaries can cause errors (I_1 could remain undetected), which can be avoided by "virtual" edges in the proximity graph.

7.4.7 Running time

Under some mild assumptions about the point cloud, the running time of the algorithm is in $O(\log \log N)$, where N is the number of points [Klein and Zachmann, 2005].

Empirically, it was found that about $n = O(a \ln a) = 200$ samples in phase 1 of the algorithm yields a sufficient bracket density so that the error is only about 0.1%.

The performance of a simple implementation in C++, running on a 2.8 GHz Pentium-IV, for detecting *all* intersections between two objects, depending on different densities of the point clouds, is shown in Figure 7.30. (We define the density of an object A with N points as the ratio of N over the number of volume units of the AABB of A (which is at most 8 as each object is scaled uniformly so that it fits into a cube of size 2^3). Computing the shortest paths between (p_i, p_j) on-the-fly during interpolation search (see Algorithm 7.6) incurs at most an additional 10% of the overall running time.

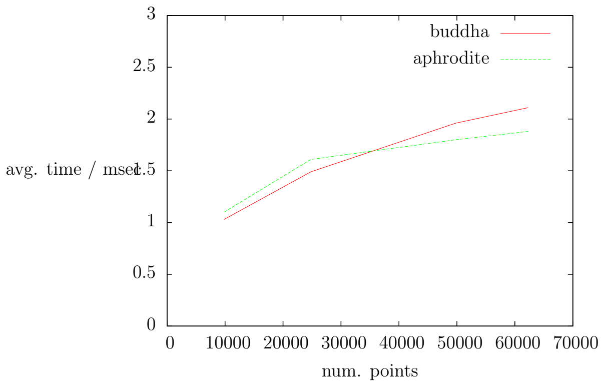


Figure 7.30: The plot shows the running time depending on the size of the point clouds for two objects (see Figures 7.19 and 7.31). The time is the average of all intersection detection times for distances between 0 and 1.5 and a lot of different orientations for two identical copies of each object.

Figure 7.31: One of the test objects for Figures 7.30.

Bibliography

Manuel Abellanas, Ferran Hurtado, Christian Icking, Rolf Klein, Elmar Langetepe, Lihong Ma, Belén Palop, and Vera Sacristán. The farthest color Voronoi diagram and related problems. In *Abstracts 17th European Workshop Comput. Geom.*, pages 113–116. Freie Universität Berlin, 2001a. URL `http://wwwpi6.fernuni-hagen.de/Publikationen/ahiklmps-fcvdr-01.pdf`. 96

Manuel Abellanas, Ferran Hurtado, Christian Icking, Rolf Klein, Elmar Langetepe, Lihong Ma, Belén Palop, and Vera Sacristán. Smallest color-spanning objects. In *Proc. 9th Annu. European Sympos. Algorithms*, volume 2161 of *Lecture Notes Comput. Sci.*, pages 278–289. Springer-Verlag, 2001b. URL `http://wwwpi6.fernuni-hagen.de/Publikationen/tr283.pdf`. 96

Anders Adamson and Marc Alexa. Approximating and intersecting surfaces from points. In *Proc. Eurographics Symp. on Geometry Processing*, pages 230–239, 2003. 127, 128, 129

Anders Adamson and Marc Alexa. Approximating bounded, non-orientable surfaces from points. In *Shape Modeling International*, 2004. accepted. 129, 131

Pankaj K. Agarwal, Leonidas J. Guibas, T. M. Murali, and Jeffrey Scott Vitter. Cylindrical static and kinetic binary space partitions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 39–48, 1997. 143

Pankaj K. Agarwal, Jeff Erickson, and Leonidas J. Guibas. Kinetic BSPs for intersecting segments and disjoint triangles. In *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, pages 107–116, 1998. 143

A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974. 195

M. Alexa, J. Behr, Daniel Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *IEEE Trans. on Visualization and Computer Graphics*, 9(1):3–15, 2003. 129

Helmut Alt and Otfried Schwarzkopf. The Voronoi diagram of curved objects. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 89–97, 1995. URL `http://www.cs.ruu.nl/~otfried/ps/as-vdco-95.ps.gz`. 93

Sigal Ar, Bernard Chazelle, and Ayellet Tal. Self-customized BSP trees for collision detection. *Computational Geometry: Theory and Applications*, 15(1–3):91–102, 2000. 47

Sigal Ar, Gil Montag, and Ayellet Tal. Deferred, self-organizing BSP trees. In *Eurographics*, pages 269–278, September 2002. URL `http://www.ee.technion.ac.il/~ayellet/papers.html`. 48, 49

J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In A. Glassner, editor, *An Introduction to Ray Tracing*, pages 201–262. Academic Press, San Diego, CA, 1989. ISBN 0-12-286160-4. 11, 52

James Arvo and David B. Kirk. Fast ray tracing by ray classification. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 55–64, July 1987. 11

S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM*, 45:891–923, 1998. 132

F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991. 75

Franz Aurenhammer and Rolf Klein. Voronoi diagrams. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000. URL `http://wwwpi6.fernuni-hagen.de/Publikationen/tr198.pdf`. 75, 76, 77, 80

David Avis and Joe Horton. Remarks on the sphere of influence graph. *Discrete geometry and convexity, Annals of the New York Academy of Sciences*, 440:323–327, 1985. 121

Francis Avnaim, Jean-Daniel Boissonnat, Olivier Devillers, Franco P. Preparata, and Mariette Yvinec. Evaluating signs of determinants using single-precision arithmetic. *Algorithmica*, 17(2): 111–132, 1997. URL `http://www-sop.inria.fr/prisme/publis/abdpy-esdus-97.ps.gz`. 183, 194

R. Baker, M. Boilen, M. T. Goodrich, R. Tamassia, and B. A. Stibel. Testers and visualizers for teaching data structures. In *Proc. of ACM SIGCSE '99*, 1999. 211

Kavita Bala, Bruce Walter, and Donald P. Greenberg. Combining edges and points for interactive high-quality rendering. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):631–640, 2003. 127

Laurent Balmelli, Jelena Kovacevic, and Martin Vetterli. Quadtrees for embedded surface visualization: Constraints and efficient data structures. In *Proc. of IEEE International Conference on Image Processing (ICIP)*, volume 2, pages 487–491, October 1999. 6

Laurent Balmelli, Thmoas Liebling, and Martin Vetterli. Computational analysis of 4-8 meshes with application to surface simplification using global error. In *Proc. of the 13th Canadian Conference on Computational Geometry (CCCG)*, August 2001. 6

Gill Barequet, Bernard Chazelle, Leonidas J. Guibas, Joseph S. B. Mitchell, and Ayellet Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Computer Graphics Forum*, 15(3):C387–C396, C484, September 1996. ISSN 0167-7055. 52

Ronen Barzel, John Hughes, and Daniel N. Wood. Plausible motion simulation for computer graphics animation. In R. Boulic and G. Hégron, editors, *Proceedings of the Eurographics Workshop Computer Animation and Simulation*, pages 183–197. Springer, 1996. 62

J. Basch, L. J. Guibas, C. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 388–390, 1997. 139

C. Batut, D. Bernadi, H. Cohen, and M. Olivier. User's guide to pari/gp, 2000. URL `www.gn-50uma.de/ftp/pari-2.1/manuals/users.pdf`. 210

N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Conf. on Management of Data*, pages 322–331, 1990. 56

M. Benouamer, P. Jaillon, D. Michelucci, and J.-M. Moreau. A lazy solution to imprecision in computational geometry. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 73–78, 1993. 180, 210

J. L. Bentley. Decomposable searching problems. *Inform. Process. Lett.*, 8:244–251, 1979. 213, 216

J. L. Bentley. Solutions to Klee's rectangle problems. Technical report, Carnegie-Mellon Univ., Pittsburgh, PA, 1977. 23

J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *Transactions on Computing*, 28(9):643– 647, 1979a. 121

J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, September 1979b. 151, 202

Jon Louis Bentley, Bruce W. Weide, and Andrew C. Yao. Optimal expected-time algorithms for closest point problems. *ACM Transactions on Mathematical Software*, 6(4):563–580, 1980. 121

J. Bernal. Bibliographic notes on Voronoi diagrams. Technical report, National Institute of Standards and Technology, Gaithersburg, MD 20899, 1992. 75

Binay K. Bhattacharya, Ronald S. Poulsen, and Godfried T. Toussaint. Application of proximity graphs to editing nearest neighbor decision rule. In *International Symposium on Information Theory*, Santa Monica, 1981. 125

Hartmut Böhme. *Albrecht Dürer: Melencolia I. Im Labyrinth der Deutung*. Frankfurt, Main, Germany, third edition, 1993. vii

J.-D. Boissonnat and F. Cazals. Natural neighbor coordinates of points on a surface. *Comput. Geom. Theory Appl.*, 19:155–173, 2001. 112

Jean-Daniel Boissonnat and Frédéric Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 223–232, 2000. 112

Jean-Daniel Boissonnat and Monique Teillaud. On the randomized construction of the Delaunay tree. *Theoret. Comput. Sci.*, 112:339–354, 1993. URL `http://www.inria.fr/cgi-bin/wais_ra_sophia?question=1140`. 83

G. Borgefors. Distance transformations in arbitrary dimensions. In *Computer. Vision, Graphics, Image Processing*, volume 27, pages 321–345, 1984. 69

Elizabeth D. Boyer, L. Lister, and B. Shader. Sphere-of-influence graphs using the sup-norm. *Mathematical and Computer Modelling*, 32(10):1071–1082, 2000. 116

Peer-Timo Bremer, Serban D. Porumbescu, Falko Kuester, Bernd Hamann, Kenneth I. Joy, and Kwan-Liu Ma. Virtual clay modeling using adaptive distance fields. In H. R. Arambnia et al., editor, *Proceedings of the 2002 International Conference on Imaging Science, Systems, and Technology (CISST 2002)*, volume 1, Athens, Georgia, 2002. 72, 73

K. Q. Brown. Voronoi diagrams from convex hulls. *Inform. Process. Lett.*, 9(5):223–228, 1979. 90

C. Burnikel, K. Mehlhorn, and S. Schirra. On degeneracy in geometric computations. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 16–23, 1994. 149, 201, 202

C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. Efficient exact geometric computation made easy. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 341–350, 1999. 149, 157

C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. A strong and easily computable separation bound for arithmetic expressions involving radicals. *Algorithmica*, 27(1):87–99, 2000. 181, 182

C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. In Friedhelm Meyer auf der Heide, editor, *Proc. 9th Annu. European Sympos. Algorithms*, volume 2161 of *Lecture Notes Comput. Sci.*, pages 254–265. Springer-Verlag, 2001. 181, 182

Christoph Burnikel, Jochen Könnemann, Kurt Mehlhorn, Stefan Näher, Stefan Schirra, and Christian Uhrig. Exact geometric computation in LEDA. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C18–C19, 1995. URL `http://www.mpi-sb.mpg.de/guide/staff/uhrig/leda.html`. 183

Cassiopeia. Cassiopeia constellation from uranometreia. URL `http://www.ras.org.uk/html/library/rare.html`. 113

L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989. 92

L. P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 274–280, 1993. 92

Norman Chin. Partitioning a 3D convex polygon with an arbitrary plane. In David Kirk, editor, *Graphics Gems III*, chapter V.2, pages 219–222. Academic Press, 1992. 44

K. L. Clarkson. Safe and effective determinant evaluation. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 387–395, October 1992. 183, 194

W. S. Cleveland and C. L. Loader. Smoothing by local regression: Principles and methods. In W. Haerdle and M. G. Schimek, editors, *Statistical Theory and Computational Aspects of Smoothing*, pages 10–49. Springer, New York, 1995. 128

Daniel Cohen-Or, Amira Solomovici, and David Levin. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, April 1998a. ISSN 0730-0301. 72, 73

Daniel Cohen-Or, Amira Solomovici, and David Levin. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, 1998b. URL `http://visinfo.zib.de/EVlib/Show?EVL-1998-152`. 72

R. Cole. Searching and storing similar lists. *J. Algorithms*, 7:202–220, 1986. 97

João Luiz Dihl Comba. *Kinetic Vertical Decomposition Trees.* PhD dissertation, Stanford University, September 1999. URL `http://graphics.stanford.edu/~comba/kvd/kvd.html`. 143

S. Coren and J. S. Girgus. *Seeing is Deceiving: The Psychology of Visual Illusions.* Lawrence Erlbaum Associates, 1978. 113

T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, IT-13(1):21–27, 1967. 122

G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, NJ, 1963. 196

J. H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra: systems and algorithms for algebraic computation.* Acacdemic Press, 1988. 168

Mark de Berg. Linear size binary space partitions for uncluttered scenes. *Algorithmica*, 28:353–366, 2000. 45

Mark de Berg. Linear size binary space partitions for fat objects. In *Proc. 3rd Annu. European Sympos. Algorithms*, volume 979 of *Lecture Notes Comput. Sci.*, pages 252–263. Springer-Verlag, 1995. 45

Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000. 3, 27, 28, 39, 45, 80

Mark de Berg, João Comba, and L. J. Guibas. A segment-tree based kinetic BSP. In *7th Annual Symposium on Computational Geometry (SOCG)*, 2001. 141, 143, 148

P. de Rezende and W. Jacometti. Geolab: An environment for development of algorithms in computational geometry. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 175–180, 1993a. 211

P. J. de Rezende and W. R. Jacometti. Animation of geometric algorithms using GeoLab. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 401–402, 1993b. 211

F. Dehne and H. Noltemeier. A computational geometry approach to clustering problems. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 245–250, 1985. 103

T. J. Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18:224–242, 1971. 173

Oliver Deussen, Stefan Hiller, Cornelius van Overveld, and Thomas Strothotte. Floating points: A method for computing stipple drawings. *Computer Graphics Forum*, 19(3), August 2000. ISSN 0167-7055. URL `http://www.blackwellpublishers.co.uk/asp/journal.asp?ref=0167-7055&src=ard&&aid=396&&iid=3&&vid=19`. 106

Olivier Devillers. The Delaunay hierarchy. *Internat. J. Found. Comput. Sci.*, 13:163–180, 2002. 132

L. Devroye, C. Lemaire, and J. M. Moreau. Expected time analysis for delaunay point location. *Comput. Geom. Theory Appl.*, 22:61–89, 2004. 102

Luc Devroye, Ernst Peter Mücke, and Binhai Zhu. A note on point location in Delaunay triangulations of random points. *Algorithmica*, 22:477–482, 1998. 102

A. K. Dewdney and J. K. Vranch. A convex partition of R^3 with applications to Crum's problem and Knuth's post-office problem. *Utilitas Math.*, 12:193–199, 1977. 87

H. Djidjev and A. Lingas. On computing the Voronoi diagram for restricted planar figures. In *Proc. 2nd Workshop Algorithms Data Struct.*, volume 519 of *Lecture Notes Comput. Sci.*, pages 54–64. Springer-Verlag, 1991. 79

D. P. Dobkin and M. J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. *Algorithmica*, 4:3–32, 1989.

D. P. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM J. Comput.*, 5:181–186, 1976. 96

Q. Du, V. Faber, and M. Gunzburger. Centroidal Voronoi tessellations: Applications and algorithms. *SIAM Reviews*, 41:637–676, 1999. 106

R. A. Dwyer. Higher-dimensional Voronoi diagrams in linear expected time. *Discrete Comput. Geom.*, 6:343–367, 1991. 87, 102

Rex A. Dwyer. The expected size of the sphere-of-influence graph. *Computational Geometry: Theory and Applications*, 5(3):155–164, 1995. 121

H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987. 75, 90, 98

H. Edelsbrunner. Dynamic data structures for orthogonal intersection queries. Report F59, Inst. Informationsverarb., Tech. Univ. Graz, Graz, Austria, 1980. 18

H. Edelsbrunner and H. A. Maurer. On the intersection of orthogonal objects. *Inform. Process. Lett.*, 13:177–181, 1981. 23

H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990. 196, 199, 201

H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15:223–241, 1996. 90

H. Edelsbrunner and R. Waupotitsch. Computing a ham-sandwich cut in two dimensions. *J. Symbolic Comput.*, 2:171–178, 1986. 201

H. Edelsbrunner, Leonidas J. Guibas, and J. Stolfi. Optimal point location in monotone subdivisions. Technical Report 2, DEC/SRC, 1984. 98

Herbert Edelsbrunner, Günter Rote, and Emo Welzl. Testing the necklace condition for shortest tours and optimal factors in the plane. *Theoretical Computer Science*, 66(2):157–180, 1989. 121

Stephan A. Ehmann and Ming C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *Computer Graphics Forum*, volume 20, pages 500–510, 2001. ISSN 1067-7055. 61

Atef A. Elassal and Vincent M. Caruso. USGS digital cartographic data standards - Digital Elevation Models. Technical Report Geological Survey Circular 895-B, US Geological Survey, 1984. 3

I. Emiris and J. Canny. An efficient approach to removing geometric degeneracies. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 74–82, 1992. 195, 196, 197, 201

I. Z. Emiris, J. F. Canny, and R. Seidel. Efficient perturbations for handling geometric degeneracies. *Algorithmica*, 19(1–2):219–242, September 1997. 196, 197, 201

A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL a computational geometry algorithms library. *Softw. – Pract. Exp.*, 30(11):1167–1202, 2000. 210

D. A. Field. Implementing Watson's algorithm in three dimensions. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 246–259, 1986. 88

S. Fortune. Numerical stability of algorithms for 2-d Delaunay triangulations and Voronoi diagrams. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 83–92, 1992a. 204, 205

S. Fortune. Robustness issues in geometric algorithms. In M. C. Lin and D. Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes Comput. Sci.*, pages 9–14. Springer-Verlag, 1996. 149, 155

S. Fortune. Stable maintenance of point set triangulations in two dimensions. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 494–505, October 1989. 150, 155, 156, 204, 205

S. Fortune. Voronoi diagrams and Delaunay triangulations. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pages 193–233. World Scientific, Singapore, 1st edition, 1992b. 75

S. Fortune and V. Milenkovic. Numerical stability of algorithms for line arrangements. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 334–341, June 1991. URL `http://www.cs.miami.edu/~vjm/papers.html`. 204

S. Fortune and C. J. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 163–172, May 1993. 150, 179

S. J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987. 93

Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 249–254. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. URL `http://visinfo.zib.de/EVlib/Show?EVL-2000-61`. 69, 72

H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14, pages 124–133, July 1980. 41, 47

Donald Fussell and K. R. Subramanian. Fast ray tracing using K-D trees. Technical Report TR-88-07, U. of Texas, Austin, Dept. Of Computer Science, March 1988. 56

Natasha Gelfand, Michael T. Goodrich, and Roberto Tamassia. Teaching data structure design patterns. In *Proc. ACM Symp. Computer Science Education*, 1998. 211

A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, Cambridge, 1985. 77

Andrew S. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press, 1989. 7

D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991. 149, 157, 161, 183, 186

Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987. 56

Francisco Gomez, Suneeta Ramaswami, and Godfried T. Toussaint. On removing non-degeneracy assumptions in computational geometry. In *CIAC '97: Proceedings of the Third Italian Conference on Algorithms and Complexity*, pages 86–99, London, UK, 1997. Springer-Verlag. ISBN 3-540-62592-5. 203

Francisco Gomez, Ferran Hurtado, Toni Sellares, and Godfried Toussaint. On degeneracies removable by perspective projections. *International Journal of Mathematical Algorithms*, 2:227–248, 2001. 203

Michael T. Goodrich and Roberto Tamassia. *Data Structures and Algorithms in Java*. John Wiley & Sons, New York, NY, 1998. 211

Stefan Gottschalk, Ming Lin, and Dinesh Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 171–180. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. 52, 53, 59, 61

I. S. Gradshteyn and I. M. Ryzhik. *Tables of Integrals, Series, and Products*. Academic Press, San Diego,CA, sixth edition, 2000. 165

Torbjörn Granlund. *GMP, The GNU Multiple Precision Arithmetic Library*, 2.0.2 edition, 1996. URL `http://www.swox.com/gmp/`. http://www.swox.com/gmp/. 210

L. Guibas, J. Pach, and M. Sharir. Generalized sphere-of-influence graphs in higher dimensions. In *Manuscript*, 1992a. 121

L. J. Guibas. Kinetic data structures — a state of the art report. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Proc. Workshop Algorithmic Found. Robot.*, pages 191–209. A. K. Peters, Wellesley, MA, 1998. 139

Leonidas J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4(2):74–123, April 1985. 80, 150

Leonidas J. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: building robust algorithms from imprecise computations. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 208–217, 1989. 149

Leonidas J. Guibas, D. E. Knuth, and Micha Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992b. 80, 83, 90

Leonidas J. Guibas, D. Salesin, and J. Stolfi. Constructing strongly convex approximate hulls with inaccurate primitives. *Algorithmica*, 9:534–560, 1993. 204

Paul E. Haeberli. Paint by numbers: Abstract image representations. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 207–214, August 1990. 71, 108

Horst W. Hamacher. *Mathematische Lösungsverfahren für planare Standortprobleme.* Verlag Vieweg, Wiesbaden, 1995. 103

W. Härdle. *Applied nonparametric regression*, volume 19 of *Econometric Society Monograph.* Cambridge University Press, New York, 1990. 128

Alejo Hausner. Simulating decorative mosaics. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 573–578, 2001. URL `http://visinfo.zib.de/EVlib/Show?EVL-2001-152`. 106, 107, 108, 109

Nicholas J. Higham. Analysis of the Cholesky decomposition of a semi-definite matrix. In M. G. Cox and S. J. Hammarling, editors, *Reliable Numerical Computation*, pages 161–185. Oxford University Press, 1990. 132

Hisamoto Hiyoshi and Kokichi Sugihara. Voronoi-based interpolation with higher continuity. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 242–250, 2000. 112

Kenneth E. Hoff III, Tim Culver, John Keyser, Ming Lin, and Dinesh Manocha. Proc. of the conference on computer graphics (siggraph99). pages 277–286, LA, California, August8–13 1999. URL `http://www.cs.unc.edu/~geom/voronoi/`. 71, 108, 253

Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 71–78, July 1992. 133

Jian Huang, Yan Li, Roger Crawfis, Shao Chiung Lu, and Shuh Yuan Liou. A complete distance field representation. In *Proceedings of the conference on Visualization 2001*, pages 247–254. IEEE Press, 2001. ISBN 0-7803-7200-X. 68

Philip M. Hubbard. Real-time collision detection and time-critical computing. In *SIVE 95, The First Worjshop on Simulation and Interaction in Virtual Environments*, number 1, pages 92–96, Iowa City, Iowa, July 1995. University of Iowa, informal proceedings. 61

Ferran Hurtado, Rolf Klein, Elmar Langetepe, and Vera Sacristán. The weighted farthest color voronoi diagram on trees and graphs. *Computational Geometry: Theory and Applications*, 27: 13–26, 2004. 93

F. K. Hwang. An $O(n \log n)$ algorithm for rectilinear minimal spanning tree. *J. ACM*, 26:177–182, 1979. 93

IEEE754. *IEEE Standard for binary floating point arithmetic, ANSI/IEEE Std* 754 − 1985. New York, NY, 1985. Reprinted in SIGPLAN Notices, 22(2):9–25, 1987. 157

H. Inagaki, K. Sugihara, and N. Sugie. Numerically robust incremental algorithm for constructing three-dimensional Voronoi diagrams. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 334–339, 1992. 88

J. W. Jaromczyk and Godfried T. Toussaint. Relative neighborhood graphs and their relatives. *Proc. of the IEEE*, 80(9):1502–1571, 1992. 114, 116

B. Joe. 3-dimensional triangulations from local transformations. *SIAM J. Sci. Statist. Comput.*, 10 (4):718–741, 1989.

B. Joe. Construction of three-dimensional Delaunay triangulations using local transformations. *Comput. Aided Geom. Design*, 8(2):123–142, May 1991a. 80

B. Joe. Geompack. A software package for the generation of meshes using geometric algorithms. *Advances in Engineering Software and Workstations*, 13(5–6):325–331, September 1991b. 90

M. W. Jones and R. A. Satherley. Shape representation using space filled sub-voxel distance fields. In Bob Werner, editor, *Proc. of the Int'l Conf. on Shape Modeling and Applications (SMI)*, pages 316–325, Genova, Italy, May 7–11 2001. IEEE. 67, 71

Lili Ju, Qiang Du, and Max Gunzburger. Probabilistic methods for centroidal Voronoi tessellations and their parallel implementations. *Parallel Computing*, 28(10):1477–1500, October 2002. ISSN 0167-8191. URL `http://www.elsevier.com/gej-ng/10/35/21/60/60/31/abstract.html`. 106

Erich Kaltofen and Gilles Villard. Computing the sign or the value of the determinant of an integer matrix, a complexity survey. *J. Comput. Appl. Math.*, 162(1):133–146, 2004. 183, 194

Tom Kamphans and Elmar Langetepe. Optimal competitive online ray search with an error-prone robot. In *Proc. of the 4th International Workshop on Efficient and Experimental Algorithms*, 2005. URL `http://web.informatik.uni-bonn.de/I/publications/kl-ocolr-05.ps`.

Tom Kamphans and Elmar Langetepe. The pledge algorithm reconsidered under errors in sensors and motion. In *Proc. of the 1th Workshop on Approximation and Online Algorithms*, volume 2909 of *Lecture Notes Comput. Sci.*, pages 165–178, Berlin, 2003. Springer.

T. C. Kao and D. M. Mount. Incremental construction and dynamic maintenance of constrained Delaunay triangulations. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 170–175, 1992. 92

Vijay Karamcheti, Chen Li, Igor Pechtchanski, and Chee Yap. *The CORE Library Project*, 1.2 edition, 1999a. URL `http://www.cs.nyu.edu/exact/core/`. http://www.cs.nyu.edu/exact/core/. 210

Vijay Karamcheti, Chen Li, Igor Pechtchanski, and Chee Yap. A core library for robust numeric an geometric computation. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 351–359, 1999b. 183, 210

M. Karasick, D. Lieber, and L. R. Nackman. Efficient Delaunay triangulations using rational arithmetic. Report RC 14455, IBM T. J. Watson Res. Center, Yorktown Heights, NY, March 1989. 167

Norio Katayama and Shin'ichi Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proc. ACM SIGMOD Conf. on Management of Data*, pages 369–380, 1997. 52

Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 269–278, August 1986. 52

J. M. Keil and C. A. Gutwin. The Delaunay triangulation closely approximates the complete Euclidean graph. In *Proc. 1st Workshop Algorithms Data Struct.*, volume 382 of *Lecture Notes Comput. Sci.*, pages 47–56. Springer-Verlag, 1989. 79

W. Keller-Gehrig. Fast algorithms for the characteristic polynomial. *Theoret. Comput. Sci.*, 36: 309–317, 1985. 200

Ron Kimmel, Nahum Kiryati, and Alfred M. Bruckstein. Multi-valued distance maps for motion planning on surfaces with moving obstacles. *IEEE Transactions on Robotics and Automation*, 14 (3):427–436, June 1998. 68, 72

Jan Klein and Gabriel Zachmann. Adb-trees: Controlling the error of time-critical collision detection. In *8th International Fall Workshop Vision, Modeling, and Visualization (VMV)*, pages 37–45, University München, Germany, November 19–21 2003. ISBN 1-58603-393-X. URL `http://www.gabrielzachmann.org/`. 62, 65

Jan Klein and Gabriel Zachmann. Point cloud surfaces using geometric proximity graphs. *Computers & Graphics*, 28(6):839–850, December 2004a. ISSN 0097-8493. URL `http://dx.doi.org/10.1016/j.cag.2004.08.012`. 117, 127, 132

Jan Klein and Gabriel Zachmann. Point cloud collision detection. *Computer Graphics forum (Proc. EUROGRAPHICS)*, 23(3):567–576, Aug 30 – Sep 3 2004b. ISSN 0167-7055. URL `http://www.gabrielzachmann.org/`. 127, 128, 129, 132

Jan Klein and Gabriel Zachmann. Interpolation search for point cloud intersection. In Vaclav Skala, editor, *Proc. of WSCG 2005*, pages 163–170, University of West Bohemia, Plzeň, Czech Republic, January 31 – February 7 2005. ISBN 80-903100-7-9. URL `http://www.gabrielzachmann.org/`. 132, 134, 137

Reinahrd Klein, Andreas Schilling, and Wolfgang Straßer. Reconstruction and simplification of surfaces from contours. In *Pacific Graphics*, pages 198–207. IEEE Computer Society, 1999. URL `http://cg.cs.uni-bonn.de/publications/publication.asp?id=31`. 72

Rolf Klein. *Algorithmische Geometrie*. Springer, Bonn, 2005. 28, 202, 214, 230

Rolf Klein, Kurt Mehlhorn, and Stefan Meiser. Randomized incremental construction of abstract Voronoi diagrams. *Comput. Geom. Theory Appl.*, 3(3):157–184, 1993. 93

James T. Klosowski, Martin Held, Jospeh S. B. Mitchell, Henry Sowrizal, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, January 1998. 52, 61

Thomas Larsson and Tomas Akenine-Möller. Efficient collision detection for models deformed by morphing. *The Visual Computer*, 19(2):164–174, May 2003. 59

Thomas Larsson and Tomas Akenine-Möller. Collision detection for continuously deforming bodies. In *Eurographics*, pages 325–333, 2001. short presentation. 60, 61

J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991. 72

C. L. Lawson. Software for C^1 surface interpolation. In J. R. Rice, editor, *Math. Software III*, pages 161–194. Academic Press, New York, NY, 1977. 80

D. T. Lee. Two-dimensional Voronoi diagrams in the L_p-metric. *J. ACM*, 27(4):604–618, 1980. 93

D. T. Lee and A. K. Lin. Generalized Delaunay triangulation for planar graphs. *Discrete Comput. Geom.*, 1:201–217, 1986. 91, 92

D. T. Lee and C. K. Wong. Voronoi diagrams in L_1 (L_∞) metrics with 2-dimensional storage applications. *SIAM J. Comput.*, 9(1):200–211, 1980. 93

I.-K. Lee. Curve reconstruction from unorganized points. *Computer Aided Geometric Design*, 17(2): 161–177, 2000. 128

S. Leutenegger, J. Edgington, and M. Lopez. STR : A simple and efficient algorithm for R-tree packing. In *Proceedings of the 13th International Conference on Data Engineering (ICDE'97)*, pages 497–507, Washington - Brussels - Tokyo, April 1997. IEEE. ISBN 0-8186-7807-0. 54

David Levin. Mesh-independent surface interpolation. In Hamann Brunnett and Mueller, editors, *Geometric Modeling for Scientific Visualization*, pages 37–49. Springer, 2003. 129

C. Li and C. Yap. A new constructive root bound for algebraic expressions. In *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, pages 496–505, 2001a. 182

Chen Li. *Exact Geometric Computation: Theory and Application.* Ph.D. dissertation, Department of Computer Science, New York University, New York, 2001. 182, 183, 210

Chen Li and Chee Yap. Recent progress in exact geometric computation. Technical Report, Dept. Comput. Sci., New York University, 2001b. 183

Tsai-Yen Li and Jin-Shin Chen. Incremental 3D collision detection with hierarchical data structures. In *Proc. VRST '98*, pages 139–144, Taipei, Taiwan, November 1998. ACM. 61

Peter Lindstrom and Valerio Pascucci. Visualization of large terrains made easy. In *Proc. IEEE Visualization*, San Diego, 2001. 5, 6

Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hughes, Nick Faust, and Gregory Turner. Real-Time, continuous level of detail rendering of height fields. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 109–118. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. 3, 6

Giuseppe Liotta, Franco P. Preparata, and Roberto Tamassia. Robust proximity queries in implicit Voronoi diagrams. Technical Report CS-96-16, Center for Geometric Computing, Comput. Sci. Dept., Brown Univ., Providence, RI, 1996. URL `ftp://ftp.cs.brown.edu/pub/techreports/96/cs96-16.ps.Z`.

William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987. 7

H. A. Maurer and T. A. Ottmann. Dynamic solutions of decomposable searching problems. In U. Pape, editor, *Discrete Structures and Algorithms*, pages 17–24. Carl Hanser Verlag, München, Germany, 1979. 213, 216

E. M. McCreight. Efficient algorithms for enumerating intersecting intervals and rectangles. Report CSL-80-9, Xerox Palo Alto Res. Center, Palo Alto, CA, 1980. 18

K. Mehlhorn and M. H. Overmars. Optimal dynamization of decomposable searching problems. *Inform. Process. Lett.*, 12:93–98, 1981. 213

Kurt Mehlhorn and Stefan Näher. The implementation of geometric algorithms. In *Proc. 13th World Computer Congress IFIP94*, volume 1, pages 223–231, 1994. 149, 176

Kurt Mehlhorn and Stefan Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000. 179, 181, 183, 210

T. S. Michael and Thomas Quint. Sphere of influence graphs and the l_∞-metric. *Discrete Applied Mathematics*, 127(3):447 – 460, 2003. 116

D. Michelucci. Arithmetic isuues in geometric computations. In *Proc. 2nd Real Numbers and Computer Conf.*, pages 43–69, 1996. 168

D. Michelucci. The robustness issue, 1997. URL `citeseer.ist.psu.edu/361993.html`. 149, 168

D. Michelucci and J.-M. Moreau. Lazy arithmetic. *IEEE Transactions on Computers*, 46(9):961–975, 1997. 210

Gordon Müller, Stephan Schäfer, and W. Dieter Fellner. Automatic creation of object hierarchies for radiosity clustering. *Computer Graphics Forum*, 19(4), December 2000. ISSN 0167-7055. 56

Bruce Naylor, John Amanatides, and William Thibault. Merging BSP trees yields polyhedral set operations. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 115–124, August 1990. 42

Bruve F. Naylor. A tutorial on binary space partitioning trees. *ACM SIGGRAPH '96 Course Notes 29*, 1996. 40, 42, 46, 47

J. Nievergelt and K. H. Hinrichs. *Algorithms and Data Structures: With Applications to Graphics and Geometry*. Prentice Hall, Englewood Cliffs, NJ, 1993. 153

J. Nievergelt and P. Schorn. Das Rätsel der verzopften Geraden. *Informatik Spektrum*, 11:163–165, 1988. 153

J. Nievergelt, P. Schorn, M. de Lorenzi, C. Ammann, and A. Brüngger. XYZ: Software for geometric computation. Report 163, Institut für Theorische Informatik, ETH, Zürich, Switzerland, July 1991. 211

M. Novotni and R. Klein. A geometric approach to 3d object comparison. In *International Conference on Shape Modeling and Applications*, pages 167–175, May 2001. 72

A. Okabe and A. Suzuki. Locational optimization problems solved through Voronoi diagrams. *European J. Oper. Res.*, 98(3):445–456, May 1997.

Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, UK, 1992. ISBN 0-471-93430-5. 75, 76, 80

R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Matching 3D models with shape distributions. In Bob Werner, editor, *Proceedings of the International Conference on Shape Modeling and Applications (SMI-01)*, pages 154–166, Los Alamitos, CA, May 7–11 2001. IEEE Computer Society. 37

K. Ouchi. Real/expr: Implementation of exact computation, 1997. URL `http://cs.nyu.edu/exact/realexpr`. 210

M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes Comput. Sci.* Springer-Verlag, Heidelberg, West Germany, 1983. 213, 230

M. H. Overmars and J. van Leeuwen. Dynamization of decomposable searching problems yielding good worst-case bounds. In *Proc. 5th GI Conf. Theoret. Comput. Sci.*, volume 104 of *Lecture Notes Comput. Sci.*, pages 224–233. Springer-Verlag, 1981a. 213, 228

M. H. Overmars and J. van Leeuwen. Some principles for dynamizing decomposable searching problems. *Inform. Process. Lett.*, 12:49–54, 1981b.

M. H. Overmars and J. van Leeuwen. Two general methods for dynamizing decomposable searching problems. *Computing*, 26:155–166, 1981c. 213, 217

M. H. Overmars and J. van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problems. *Inform. Process. Lett.*, 12:168–173, 1981d. 213, 228

Mark H. Overmars. Designing the Computational Geometry Algorithms Library CGAL. In *Proc. 1st ACM Workshop on Appl. Comput. Geom.*, volume 1148 of *Lecture Notes Comput. Sci.*, pages 53–58. Springer-Verlag, May 1996. URL `http://www.cs.ruu.nl/CGAL/Papers/panel.ps.gz`. 182, 214

David W. Paglieroni. Distance transforms: Properties and machine vision applications. *CVGIP: Graphical Models and Image Processing*, 54(1):56–74, January 1992. 68

I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 14(2):105–116, June 1995. ISSN 0167-7055. 61

M. S. Paterson and F. F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom.*, 5:485–503, 1990. 39, 45

Mark Pauly, Markus H. Gross, and Leif Kobbelt. Efficient simplification of point-sampled surfaces. In *IEEE Visualization 2002*, pages 163–170, 2002. 130

Mark Pauly, Richard Keiser, Leif P. Kobbelt, and Markus Gross. Shape modeling with point-sampled geometry. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):641–650, 2003. 132

Bradley A. Payne and Arthur W. Toga. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications*, 12(1):65–71, January 1992. 72

Hanspeter Pfister, Jeroen van Baar, Matthias Zwicker, and Markus Gross. Surfels: Surface elements as rendering primitives. *ACM Transactions on Graphics (SIGGRAPH 2000)*, 19(3):335–342, 2000. 127, 133

B. Piper. Properties of local coordinates based on Dirichlet tesselations. In G. Farin, H. Hagen, H. Noltemeier, and W. Knödel, editors, *Geometric modelling*, volume 8 of *Computing. Supplementum*, pages 227–240, Wien / New York, 1993. Springer. ISBN 0-387-82399-9, 3-211-82399-9. Based on lectures given at the 1st Dagstuhl-seminar, Dagstuhl, Germany. 111

F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 3rd edition, October 1990. ISBN 3-540-96131-3. 75, 157

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.).* Cambridge University Press, Cambridge, 1992. ISBN 0-521-43108-5. 132

V. T. Rajan. Optimality of the Delaunay triangulation in R^d. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 357–363, 1991. 80, 90, 91

J. Revelles, C. Urena, and M. Lastra. An efficient parametric algorithm for octree traversal. In *WSCG 2000 Conference Proceedings*, University of West Bohemia, Plzen, Czech Republic, 2000. URL `http://visinfo.zib.de/EVlib/Show?EVL-2000-307`. 9

C.A. Rogers. Covering a sphere with spheres. *Mathematika*, 10:157–164, 1963. 137

Roman mosaic 1. Mosic from a roman bath in bath, uk. URL `http://www2.sjsu.edu/depts/jwss/bath2004/baths.html`. 104

Roman mosaic 2. Mosaic floor from a late roman bath, with main theme two peacocks enframing a vessel of "kantharos" shape. URL `http://www.culture.gr/2/21/211/21110m/e211jm04.html`. 104

Nick Roussopoulos and Daniel Leifker. Direct spatial search on pictorial databases using packed R-trees. In Sham Navathe, editor, *Proceedings of ACM-SIGMOD 1985 International Conference on Management of Data, May 28–31, 1985, LaMansion Hotel, Austin, Texas*, pages 17–31, New York, NY 10036, USA, 1985. ACM Press. ISBN 0-89791-160-1. URL `http://www.acm.org/pubs/articles/proceedings/mod/318898/p17-roussopoulos/p17-roussopoulos.pdf;http://www.acm.org/pubs/citations/proceedings/mod/318898/p17-roussopoulos/`. 54

Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. *ACM Transactions on Graphics (SIGGRAPH 2000)*, 19(3):343–352, 2000. 127, 133

J.S. Sanchez, F. Pla, and F.J. Ferri. Prototype selection for the nearest neighbour rule through proximity graphs. *Pattern Recognition Letters*, 18(6):507–513, June 1997. 125

Francisco J. Santisteve. Robust geometric computation (rgc), state of the art, 1999. URL `citeseer.ist.psu.edu/santisteve99robust.html`. 149, 168

N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29 (7):669–679, July 1986. 97

Junaed Sattar. The spheres-of-influence graph. Web Page. URL `http://www.cs.mcgill.ca/~jsatta/pr507/about.html`. 113

J. B. Saxe and J. L. Bentley. Transforming static data structures to dynamic structures. In *Proc. 20th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 148–168, 1979. 213

Stefan Schirra. Robustness and precision issues in geometric computation. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, chapter 14, pages 597–632. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000. 149

P. Schorn. An object-oriented workbench for experimental geometric computation. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 172–175, 1990. 211

P. Schorn. *Robust Algorithms in a Program Library for Geometric Computation*, volume 32 of *Informatik-Dissertationen ETH Zürich*. Verlag der Fachvereine, Zürich, 1991.

H. R. Schwarz. *Numerical Analysis: A Comprehensive Introduction*. B. G. Teubner, Stuttgart, 1989. 166, 193

Robert Sedgewick. *Algorithms*. Addison-Wesley, Reading, 2 edition, 1989. 136

R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. Technical Report 260, IIG-TU Graz, Austria, 1988. 92

B. Serpette, J. Vuillemin, and J. C. Herv'e. BigNum: a portable and efficient package for arbitrary-precision arithmetic. Technical report, INRIA, May 1989. 210

J. A. Sethian. *An Analysis of Flame Propagation*. PhD thesis, Dept. of Mathematics, University of California, Berkeley, 1982. URL `http://visinfo.zib.de/EVlib/Show?EVL-1982-2`. 68

J. A. Sethian. *Level Set Methods*. Cambridge University Press, 1996. URL `http://visinfo.zib.de/EVlib/Show?EVL-1996-149`. 71

J. A. Sethian. *Level set methods and fast marching methods*. Cambridge University Press, 1999. 68

M. I. Shamos. *Computational Geometry*. Ph.D. thesis, Dept. Comput. Sci., Yale Univ., New Haven, CT, 1978. 79

Micha Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995. 96

J. Shewchuk. Lecture notes on geometric robustness. Technical report, University of California at Berkeley, Berkeley, CA, 1999. 183, 189

Jonathan R. Shewchuk. Robust adaptive floating-point geometric predicates. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 141–150, 1996. 174

Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete Comput. Geom.*, 18(3):305–363, 1997. 149, 169, 171, 173, 174

R. Sibson. A brief description of natural neighbour interpolation. In Vic Barnet, editor, *Interpreting Multivariate Data*, pages 21–36. John Wiley & Sons, Chichester, 1981. 110

R. Sibson. A vector identity for the Dirichlet tesselation. *Math. Proc. Camb. Phil. Soc.*, 87:151–155, 1980. 110

K. Sugihara and M. Iri. Two design principles of geometric algorithms in finite-precision arithmetic. *Appl. Math. Lett.*, 2(2):203–206, 1989. 150

K. Sugihara, M. Iri, H. Inagaki, and T. Imai. Topology-oriented implementation - an approach to robust geometric algorithms. *Algorithmica*, 27(1):5–20, 2000. 150

Kokichi Sugihara. How to make geometric algorithms robust. *IEICE Transactions on Information and Systems*, E83-D(3):447–454, 2000. 149, 165

Kokichi Sugihara. Robust gift-wrapping for the three-dimensional convex hull. *J. Comput. Syst. Sci.*, ??, 1993. to appear. 155

R. Tamassia and J. S. Vitter. Optimal cooperative search in fractional cascaded data structures. *Algorithmica*, 15(2), 1996. 98

Roberto Tamassia, Luca Vismara, and James E. Baker. A case study in algorithm engineering for geometric computing. In *Proc. Workshop on Algorithm Engineering*, pages 136–145, September 1997. URL `http://www.dsi.unive.it/~wae97/proceedings/ONLY_PAPERS/pap14.ps.gz`. 211

M. Tanemura, T. Ogawa, and W. Ogita. A new algorithm for three-dimensional Voronoi tesselation. *J. Comput. Phys.*, 51:191–207, 1983. 88

Enric Torres. Optimization of the binary space partition algorithm (BSP) for the visualization of dynamic scenes. In C. E. Vandoni and D. A. Duce, editors, *Eurographics '90*, pages 507–518. North-Holland, September 1990. 46

Godfried T. Toussaint. A graph-theoretical primal sketch. In Godfried T. Toussaint, editor, *Computational Morphology*, pages 229–260, 1988. 121

S. Uno and M. Slater. The sensitivity of presence to collision response. In *Proc. of IEEE Virtual Reality Annual International Symposium (VRAIS)*, page 95, Albuquerque, New Mexico, March01–05 1997. 62

T. Lewis V. Barnett. *Outliers in Statistical Data*. John Wiley and Sons, New York, 1994. 117

V. K. Vaishnavi and D. Wood. Data structures for the rectangle containment and enclosure problems. *Comput. Graph. Image Process.*, 13:372–384, 1980.

V. K. Vaishnavi and D. Wood. Rectilinear line segment intersection, layered segment trees and dynamization. *J. Algorithms*, 3:160–176, 1982. 23

V. K. Vaishnavi, H. P. Kriegel, and D. Wood. Space and time optimal algorithms for a class of rectangle intersection problems. *Inform. Sci.*, 21:59–67, 1980.

Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997. 61

M. J. van Kreveld. *New Results on Data Structures in Computational Geometry*. Ph.D. dissertation, Dept. Comput. Sci., Utrecht Univ., Utrecht, Netherlands, 1992. 214

J. van Leeuwen and H. A. Maurer. Dynamic systems of static data-structures. Report F42, Inst. Informationsverarb., Tech. Univ. Graz, Graz, Austria, 1980. 213, 217

J. van Leeuwen and M. H. Overmars. The art of dynamizing. In *Proc. 10th Internat. Sympos. Math. Found. Comput. Sci.*, volume 118 of *Lecture Notes Comput. Sci.*, pages 121–131. Springer-Verlag, 1981. 213

J. van Leeuwen and D. Wood. Dynamization of decomposable searching problems. *Inform. Process. Lett.*, 10:51–56, 1980. 213

Ming Wan, Frank Dachille, and Arie Kaufman. Distance-field based skeletons for virtual navigation. In *Proceedings of the conference on Visualization 2001*, pages 239–246. IEEE Press, 2001. ISBN 0-7803-7200-X. 72

C. A. Wang. Efficiently updating the constrained Delaunay triangulations. *BIT*, 33:238–252, 1993. 92

C. A. Wang and L. Schubert. An optimal algorithm for constructing the Delaunay triangulation of a set of line segments. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 223–232, 1987. 92

D. F. Watson. Computing the n-dimensional Delaunay tesselation with applications to Voronoi polytopes. *Comput. J.*, 24(2):167–172, 1981. 88

Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, January 1984. 52

Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 479–488. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. URL `http://visinfo.zib.de/EVlib/Show?EVL-2000-87`. 36, 37

Holger Wendland. Piecewise polynomial, positive definite and compactly supported radial basis functions of minimal degree. *Advances in Computational Mathematics*, 4:389–396, 1995. 128

Jane Wilhelms and Allen Van Gelder. Octrees for faster isosurface generation extended abstract. In *Computer Graphics (San Diego Workshop on Volume Visualization)*, volume 24, pages 57–62, November 1990. 7

C. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7(1):3–23, 1997a. 150, 157, 182

C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete Comput. Geom.*, 2:365–393, 1987a. 93

C. K. Yap. Robust geometric computation. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 35, pages 653–668. CRC Press LLC, Boca Raton, FL, 1997b. 149

C. K. Yap. Symbolic treatment of geometric degeneracies. In *Proc. 13th IFIP Conf. System Modelling and Optimization*, pages 348–358, 1987b. 196

C. K. Yap. Towards exact geometric computation. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 405–419, 1993. 194, 209

C. K. Yap and T. Dubé. The exact computation paradigm. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 452–492. World Scientific, Singapore, 2nd edition, 1995. URL `http://cs.nyu.edu/cs/faculty/yap/papers/paradigm.ps`. 209

Christine Youngblut, Rob E. Johnson, Sarah H. Nash, Ruth A. Wienclaw, and Craig A. Will. Different applications of two-dimensional potential fields for volume modeling. Technical Report UCAM-CL-TR-541, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, August 2002. URL `http://www.cl.cam.ac.uk/users/nad/pubs/`. 72

Jiaxun Yu. Exact arithmetic solid modeling. Technical Report CSD-TR-92-037, Comput. Sci. Dept., Purdue University, June 1992. 167

Gabriel Zachmann. Minimal hierarchical collision detection. In *Proc. ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 121–128, Hong Kong, China, November 11–13 2002. URL `http://www.gabrielzachmann.org/`. 57, 59

Gabriel Zachmann. Real-time and exact collision detection for interactive virtual prototyping. In *Proc. of the 1997 ASME Design Engineering Technical Conferences*, Sacramento, California, September 1997. Paper no. CIE-4306. 61

Gabriel Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS '98*, pages 90–97, Atlanta, Georgia, March 1998. 52, 61

B. Zhu and A. Mirzaian. Sorting does not always help in computational geometry. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 239–242, August 1991. 79

Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. EWA splatting. *IEEE Trans. on Visualization and Computer Graphics*, 8(3):223–238, 2002. ISSN 1077-2626. 127

Copyright Notices