Realistic Input Models for Geometric Algorithms

Mark de Berg* Matthew J. Katz† A. Frank van der Stappen*
Jules Vleugels*

Abstract

Many algorithms developed in computational geometry are needlessly complicated and slow because they have to be prepared for very complicated, hypothetical inputs. To avoid this, realistic models are needed that describe the properties that realistic inputs have, so that algorithms can de designed that take advantage of these properties. This can lead to algorithms that are provably efficient in realistic situations.

We obtain some fundamental results in this research direction. In particular, we have the following results.

- We show the relations between various models that have been proposed in the literature.
- For several of these models, we give algorithms to compute the model parameter(s) for a given scene; these algorithms can be used to verify whether a model is appropriate for typical scenes in some application area.
- As a case study, we give some experimental results on the appropriateness of some of the models for one particular type of scenes often encountered in geographic information systems, namely certain triangulated irregular networks.

1 Introduction

Many of the problems studied in computational geometry come from application areas such as computer graphics, robotics, and geographic information systems (GIS). These problems are studied in an abstract setting. There are two dangers in the abstraction process. The first, well-known danger is that the abstract version is a too-much-simplified version of the real problem, so that the obtained solution is not applicable in practice. Most researchers are well aware of this danger, and they try to get rid of unrealistic assumptions as much as possible. The second danger is that the abstract version of the problem is *too* general, so that useful properties are lost. As a result, the algorithms have to be prepared for very complicated, hypothetical inputs. Actually, the situation is even worse: because one is usually interested in the *worst-case* performance, the algorithms are geared to these unrealistic inputs. Thus they become needlessly complicated and slow.

Consider as an example the construction of a binary space partition (BSP) tree for a set of disjoint triangles in 3-space. Paterson and Yao [15] showed that it is always possible to construct a BSP of size $O(n^2)$. Moreover, there are sets of triangles for which any BSP must have quadratic size, so the $O(n^2)$ upper bound is the best we can hope for *in the worst case*. The lower bound example, however, is very artificial: it consists of n lines arranged in a very special way, namely in a grid-like fashion on a ruled surface. Hence, it does not say anything about what can be achieved in practice; perhaps realistic scenes have some special property that makes it possible to construct a BSP of linear size. Indeed, de Berg [5] showed that so-called *uncluttered scenes*—and it is claimed that in practical applications most scenes are uncluttered—admit a linear size BSP. (See Section 2 for a definition of "uncluttered".)

*Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands. Supported by the Dutch Organisation for Scientific Research (N.W.O.) and by the ESPRIT IV LTR Project No. 21957 (CGAL).

†Department of Mathematics & Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel. Supported by the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities.

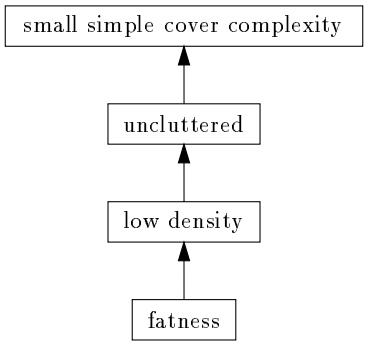

Figure 1: Relations between the models.

Another example comes from ray tracing. Agarwal and Matoušek [2] developed a data structure for 3-dimensional objects with roughly $O(n^{3/4})$ query time, which uses roughly linear storage. It is believed that this is theoretically close to optimal. But octrees, for instance, have been observed to behave well in many practical situations. Apparently, ray-shooting queries are not as difficult in practice as they are in theory. For this reason, Mitchell et al. [13] developed a data structure for 2-dimensional objects based on quad trees, and they analyzed its performance in terms of the so-called *simple-cover complexity*. They showed that the storage of the data structure is small if the simple-cover complexity of the scene is small, and that the query time is small if the simple-cover complexity of the query ray is small. (See Section 2 for a definition of "simple-cover complexity".) Again, it is expected that this will usually be the case in practice.

These two examples point in a promising research direction, where problems are studied under realistic assumptions, and algorithms analyzed using different, more practical complexity measures. This type of research can lead to algorithms and data structures that are *provably efficient* in *realistic situations*. Hence, the research is interesting from a theoretical as well as from a practical point of view. The crucial factor in this approach lies in the development of realistic models for the input scenes. The models should be general enough so that they are realistic, while still precluding complicated hypothetical inputs.

In this paper we take a first step in this direction, by studying the relations between various models that have been proposed in the literature for sets of non-intersecting objects. In particular, we study the following models: fatness, low density, clutteredness, and simple-cover complexity. (Definitions of the models are given in Section 2.) In Section 3 we show that these models are related as follows: fatness implies low density, which in turn implies unclutteredness, which implies small simple-cover complexity. Hence, the models form a hierarchy as depicted in Figure 1.

The relations between fatness and low density, and low density and unclutteredness, were already known [5, 18]. One of the contributions of our paper is that we prove that any uncluttered scene—and hence any low-density scene, or any scene consisting of fat objects—has small simple-cover complexity.

Each of the models assigns one (or two) parameter(s) to a scene. If a model is realistic in a certain setting, then its parameter should have a favourable value—usually a small constant—for a typical scene in that setting. This has to be verified, of course. Therefore we develop, in Section 4, algorithms to compute the model parameters. This is also useful because some algorithms—for instance, the range searching data structure for fat objects developed by Overmars and van der Stappen [14]—may require the value of the model parameter as input in order to work correctly. Finally, analyzing an input (for instance by computing its model parameters) can be useful for selecting the algorithm best tailored to that specific input.

What constitutes a good model depends on the application domain: architectural models have different properties than molecular models, for instance. For each application domain, it should be verified experimentally which models are appropriate. As a case study, we report in Section 5 some

initial experimental results indicating which models are realistic for certain triangulated irregular networks (TINs), which are encountered frequently in GIS. More precisely, we have taken digital-elevation model (DEM) files giving the elevations of points in certain areas in the US. The data points are arranged in a grid. We selected the most important points using the so-called VIP method [8], and computed the 2D Delaunay triangulation of the selected points, thus obtaining a TIN. We then computed the model parameters for the resulting set of 2D triangles, which indicate whether or not the models are realistic.

2 The models

Below we define the models we study in this paper. Each model assigns one (or two) parameter(s) to a scene or to the objects in it. Algorithms can then be analyzed not only in terms of the number of objects in the scene, but also in terms of these model parameters. The hope is that, for realistic inputs, the model parameters have favorable values (not-too-large, or for some models: not-too-small, constants), so that the performance of the algorithm is good in practice.

Before we define the models we need to introduce some terminology and notation.

We say that two objects are *non-intersecting* if they have disjoint interiors. A *d-dimensional scene* is defined to be any collection of non-intersecting, constant-complexity objects in $\mathbb{E}^d$.

We denote the volume of an object $\mathcal{P}$ by $\mathrm{vol}(\mathcal{P})$, and the Euclidean distance between two points p and q by $\mathrm{dist}(p,q)$. For sets A and B, we define $\mathrm{dist}(A,B) := \inf_{a\in A, b\in B} \mathrm{dist}(a,b)$. The minimal enclosing ball of an object $\mathcal{P}$ is denoted by $\mathrm{meb}(\mathcal{P})$, and the radius of the minimal enclosing ball is denoted by $\rho_{\mathrm{meb}}(\mathcal{P})$. We define ω_d to be the volume of the d-dimensional unit ball. (For even dimensions, $\omega_d = \pi^{d/2}/(d/2)!$, and for odd dimensions, $\omega_d = 2(2\pi)^{\lfloor d/2\rfloor}/d!!$.)

Finally, we shall sometimes work with bounding boxes of the objects. These are always axis-parallel. Thus the *bounding box* $\mathrm{bb}(\mathcal{P})$ of an object $\mathcal{P}$ is the smallest axis-parallel hyperrectangle that contains the object.

Fatness. The first, and best known, model we consider is fatness. Intuitively, an object is called fat if it contains no long and skinny parts. There are many different definitions of fatness [1, 3, 12, 10], which are all more or less equivalent—at least for convex objects. We shall use a definition similar to the one used in van der Stappen's thesis [18], where the motion planning problem for scenes consisting of fat objects is studied.

Definition 2.1 *Let $\mathcal{P} \subseteq \mathbb{E}^d$ be an object and let β be a constant with $0 \leqslant \beta \leqslant 1$. Define $U(\mathcal{P})$ as the set of all balls centered inside $\mathcal{P}$ whose boundary intersects $\mathcal{P}$. We say that the object $\mathcal{P}$ is* β-fat *if for all balls $B \in U(\mathcal{P})$, $\mathrm{vol}(\mathcal{P} \cap B) \geqslant \beta \cdot \mathrm{vol}(B)$. The* fatness *of $\mathcal{P}$ is defined as the maximal β for which $\mathcal{P}$ is β-fat.*

(Van der Stappen's original definition is slightly different: what he calls k-fat for some $k \geqslant 1$, is $(1/k)$-fat in our definition.) In the rest of the paper, we shall sometimes omit the fatness-parameter β, and call an object *fat* if it is β-fat for some not too small constant β. In addition, the fatness of a scene of objects is defined as the maximal β for which every individual object is β-fat.

Low density. The model of low density was introduced by van der Stappen [18] (see also [19]) and refined by Schwarzkopf and Vleugels [17]. It forbids any ball B to be intersected by many objects whose minimal-enclosing-ball radius is at least as large as the radius of B.

Definition 2.2 *Let $\mathcal{S} := \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ be a d-dimensional scene, and let $\lambda \geqslant 1$ be a parameter. We call $\mathcal{S}$ a* λ-low-density scene *if for any ball B, the number of objects $\mathcal{P}_i \in \mathcal{S}$ with $\rho_{\mathrm{meb}}(\mathcal{P}_i) \geqslant \mathrm{radius}(B)$ that intersect B is at most λ. The* density *of $\mathcal{S}$ is defined to be the smallest λ for which $\mathcal{S}$ is a λ-low-density scene.*

We say that a scene has *low density* if its density is a small constant.

Clutteredness. This model was introduced by de Berg [5].[1] It is defined as follows.

Definition 2.3 *Let $\mathcal{S}$ be a d-dimensional scene, and let $\kappa \geqslant 1$ be a parameter. We call $\mathcal{S}$ a* κ-cluttered *scene if any hypercube whose interior does not contain a vertex of one of the bounding boxes of the objects in $\mathcal{S}$ is intersected by at most κ objects in $\mathcal{S}$. The* clutter factor *of a scene is the smallest κ for which it is κ-cluttered.*

We sometimes call a scene whose clutter factor is a small constant *uncluttered*. Of the models we consider, this is the only one not invariant under rotations, since it uses bounding boxes and axis-parallel hypercubes. Although invariance under rotations is a desirable property from a mathematical point of view, it is not appropriate in all applications; architectural models, for instance, are not invariant under rotations.

Simple-cover complexity. The last model we consider is a variant of the simple-cover complexity as defined by Mitchell et al. [13]. Given a scene $\mathcal{S}$, we call a ball *δ-simple* if it intersects at most δ objects in $\mathcal{S}$.

Definition 2.4 *Let $\mathcal{S}$ be a d-dimensional scene, and let $\delta > 0$ be a parameter. A* δ-simple cover *for $\mathcal{S}$ is a collection of δ-simple balls whose union covers* $\mathrm{bb}(\mathcal{S})$. *We say that $\mathcal{S}$ has* (σ, δ)-simple-cover complexity *if there is a δ-simple cover for $\mathcal{S}$ of cardinality σn. The* δ-simple-cover complexity *of $\mathcal{S}$ is the smallest σ for which $\mathcal{S}$ has (σ, δ)-simple-cover complexity.*

(In the original definition, there was an additional, somewhat unnatural, parameter ϵ. Furthermore, the balls only had to cover the complement of the objects within the bounding box, not the entire bounding box.) We will say that a scene has *small simple-cover complexity* if there are small constants σ and δ such that it has (σ, δ)-simple-cover complexity. Contrary to the other three models, simple-cover complexity involves two parameters. It is evident that the value of σ heavily depends on the choice of the parameter δ. In general, the parameter δ should be at least as large as the maximum number of objects touching in a single point, otherwise no δ-simple cover exists.

3 Relations between the models

We say that one model *implies* another model if, for any given constant model parameter(s) for the first model, there exist constant model parameters for the second model that only depend on the constants for the first model, such that any input satisfying the first model for the given constants also satisfies the second model for the derived constants. Clearly, this relation is transitive: if model A implies model B, and model B implies model C, then model A implies model C. In this section we prove that the models defined in the previous section form a hierarchy as depicted in Figure 1, and as made precise in Theorems 3.1, 3.2, and 3.3 below. The hierarchy is strict, that is, the relations stated in the theorem and the ones implied by transitivity are the only relations between the models. For instance, an uncluttered scene does not necessarily have low density.

There are two obvious reasons why establising the relations between the models is useful. First, in this initial stage of the development of a model-based theory of geometric algorithms, it is important that we understand the properties and relations between the models. Second, if we know that model A implies model B, and we have an efficient data structure or algorithm for model B, then we can safely use that structure or algorithm in application domains where model A is appropriate. For example, if we have an application domain where the objects are fat, then algorithms based on the octree-like data structure of Mitchell et al. [13] will be efficient.

Theorem 3.1 *(i) Any d-dimensional scene consisting of β-fat objects has density at most $2^d/\beta$.*

[1] In an earlier version [4] of this paper the term 'bounding-box-fitness' was used.

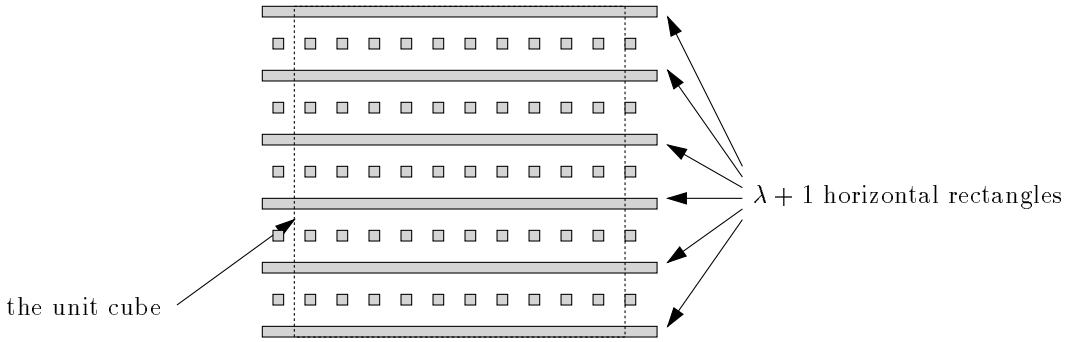

Figure 2: A 1-cluttered scene that is not a λ-low-density scene.

(ii) For any parameters λ, β with $\lambda \geqslant 1$ and $\beta > 0$, there is a 2-dimensional λ-low-density scene that is not β-fat.

Proof. (i) This was proved by van der Stappen [18]. His proof roughly works as follows. For any region R, he defines a region R' that results from growing R by an amount that is proportional to the size of R. Each object $\mathcal{P}$ that intersects R and is larger than R contains at least some constant fraction of the volume of R'. The combination of the volume of R' and a lower bound on the volume of $\mathcal{P} \cap R'$ results in the claimed bound on the number of objects that can intersect R.

(ii) A scene consisting of a single line segment (or a scene consisting of unit line segments such that the distance between any two segments is more than one) has density 1 but fatness 0. □

Theorem 3.2 *(i) Any d-dimensional λ-low-density scene has clutter factor at most $\lceil\sqrt{d}\rceil^d \lambda$.*

(ii) For any parameters κ, λ with $\kappa \geqslant 1$ and $\lambda \geqslant 1$, there is a 2-dimensional κ-cluttered scene that is not a λ-low-density scene.

Proof. (i) This was proved by de Berg [5]. He observes that any cube intersecting an object without containing a bounding-box vertex of that object in its interior must be relatively small with respect to the diameter of the object. From this the result easily follows.

(ii) Let κ, λ be given. We can assume without loss of generality that $\kappa = 1$. Let C be the unit square. Let L be a set of $\lambda + 1$ thin horizontal rectangles intersecting C; the rectangles are equally spaced, the distance between them being roughly $1/(\lambda - 1)$. This scene is not a λ-low-density scene, but it is not 1-cluttered either. However, we can make it 1-cluttered by adding a row of small squares in between each pair of horizontal rectangles, as in Figure 2; the horizontal distance between two consecutive squares should just be smaller than the distance between consecutive horizontal rectangles. □

Theorem 3.3 *(i) There are constants σ and c such that any d-dimensional κ-cluttered scene has $(\sigma, c\kappa)$-simple-cover complexity.*

(ii) For any parameters σ, δ, κ with $\sigma \geqslant 2$, $\delta \geqslant 1$, and $\kappa \geqslant 1$, there are scenes with (σ, δ)-simple-cover complexity that are not κ-cluttered.

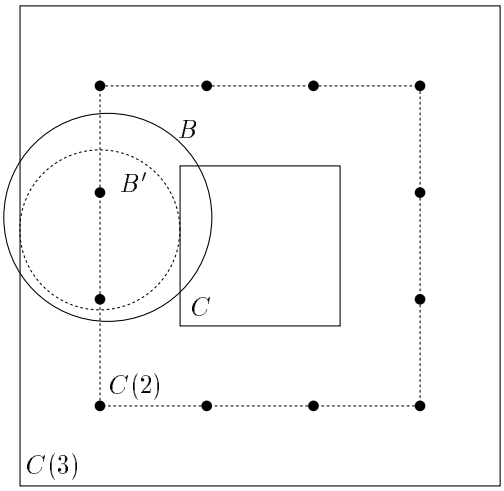

Figure 3: The points added to surround a cube C.

Proof. (i) Let $\mathcal{S}$ be a d-dimensional κ-cluttered scene consisting of n objects. De Berg [4] showed that $\mathrm{bb}(\mathcal{S})$ can be covered by a collection $\mathcal{C}$ of $2^{2d}n = O(n)$ cubes with the following properties:

(P1) The cubes in $\mathcal{C}$ contain no bounding-box vertices in their interior, so they intersect at most κ objects.

(P2) A cube in $\mathcal{C}$ intersects at most $O(2^d)$ other cubes in $\mathcal{C}$.

From the set $\mathcal{C}$ we shall construct a point set P of size $O(n)$ with the following property: any ball that does not contain any point from P in its interior is δ-simple (that is, intersects at most δ objects) for a suitable constant δ. This is done as follows.

For a cube C, denote by $C(\alpha)$ the cube with the same center as C but scaled by a factor α. Let $C \in \mathcal{C}$. We put a $k \times k \times \cdots \times k$ grid on each facet of $C(2)$ with the following property: for any point on the boundary of $C(2)$, there is a grid point at distance less than $x/2$, where x is the side length of C. This is satisfied if we take $k = \lceil 2\sqrt{d-1} + 1/2 \rceil$. Figure 3 illustrates the construction in the plane. Let P_C be the resulting collection of grid points. We claim that now the following holds: any ball intersecting the interior of C and not containing a point from P_C in its interior must be fully contained in $C(3)$. Indeed, let B be a ball that intersects C and is not fully contained in $C(3)$. Then we can define a ball $B' \subset B$ whose center lies on the boundary of $C(2)$ and that touches both the boundary of C and the boundary of $C(3)$. But then the radius of B' is $x/2$ and so B', and hence also B, contains a grid point from P_C.

Now consider the set $P := \bigcup_{C \in \mathcal{C}} P_C$. Note that P contains roughly $2d \cdot \lceil 2\sqrt{d-1} \rceil^{d-1} n$ points. Let B be a ball not containing any of these points. We know that $B \subset C(3)$ for any cube $C \in \mathcal{C}$ that intersects B. Since the cubes in $\mathcal{C}$ have only limited overlap (property (P2)), this means that B cannot intersect more than $O(1)$ cubes from $\mathcal{C}$. (An easy argument based on volumes can be used to prove this.) Hence, such a ball B is δ-simple for some $\delta = O(\kappa)$.

Using P we can find a δ-simple cover of size $O(n)$ as follows: use the method of Bern et al. [6] to add to P some additional points such that the number of simplices of the Delaunay triangulation is $O(|P|) = O(n)$, and use the Delaunay balls (the circumscribing balls of the simplices in the Delaunay triangulation) for the cover. Since the Delaunay balls do not contain points from P in their interior, this gives us a δ-simple cover.

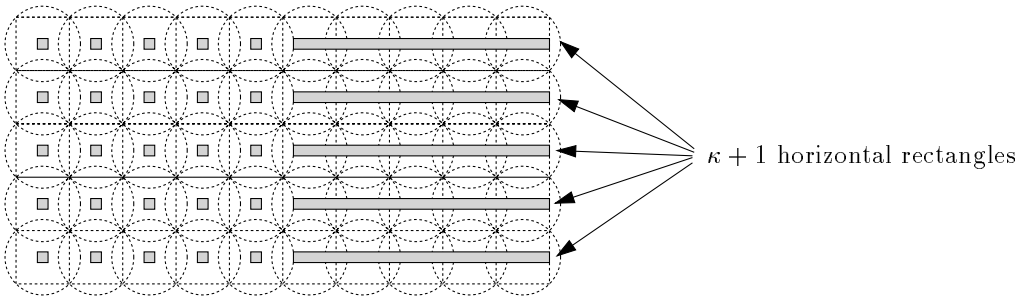

Figure 4: A scene with small simple-cover complexity that is not uncluttered.

(ii) Consider a regular grid with $2(\kappa+1) \times (\kappa+1)$ cells. Place a small square inside each grid cell in the left half of the grid, and place long and thin horizontal rectangles in the middle of each of the rows in the right half of the grid—see Figure 4. The total number of objects we have placed is $(\kappa+1)^2 + (\kappa+1)$. It is easy to do this in such a way that the circumscribing discs of the grid cells are 1-simple. Hence, there is a 1-simple cover of size $2(\kappa+1)^2 \leqslant 2n$. Clearly, the scene is not κ-cluttered: there is a square without bounding-box vertices in its interior that intersects each of the $\kappa+1$ horizontal rectangles.

□

4 Computing the model parameters

To check whether a given model is realistic in a certain application, one needs to determine the model parameter for typical scenes in that application. Computing model parameters is also useful for other reasons. Some algorithms may need to know the value of the parameter in order to work correctly. (The range searching data structure of Overmars and van der Stappen [14] is an example.) In addition, the ability to compute the model parameters for a given scene can be used to establish which model best fits the scene, allowing us to choose "the right algorithm for the job". For example, if the scene consists of relatively fat objects, a simple algorithm that is very efficient for fat objects might be preferred to a more general algorithm for uncluttered scenes. (Such an algorithm-selection strategy is commonplace in commercial LP applications.) In this section we develop algorithms to compute the model parameters for the models defined in Section 2. We only consider the planar case, and we assume that the objects are simple polygons.

Fatness. The fatness of a convex object $\mathcal{P}$ can easily be computed: it is $\mathrm{vol}(\mathcal{P})/(\omega_d \cdot \mathrm{diam}(\mathcal{P})^2)$, where $\mathrm{diam}(\mathcal{P})$ denotes the diameter of $\mathcal{P}$. Computing the fatness of non-convex objects is much more difficult—see Vleugels's thesis [20] for some results. Since in our setting the individual objects in a scene have constant complexity, we do not go into the computation of the fatness here.

Low density. To compute the density of a planar scene $\mathcal{S}$, we have to determine the smallest λ such that any disc D is intersected by at most λ objects $\mathcal{P}_i \in \mathcal{S}$ with $\rho_i \geqslant \mathrm{radius}(D)$, where $\rho_i := \rho_{\mathrm{meb}}(\mathcal{P}_i)$. It is easy to see that we can restrict our attention to discs with radius ρ_i for some $1 \leqslant i \leqslant n$.

To simplify the description, we assume from now on that all radii ρ_i are distinct. We first order the objects according to the radii of their minimal enclosing discs, such that $\rho_1 > \ldots > \rho_n$. Define $\mathcal{S}_i := \{\mathcal{P}_1, \ldots, \mathcal{P}_i\}$. Let $\mathcal{M}_j(\rho)$ denote the Minkowski sum of $\mathcal{P}_j$ and the disc of radius ρ centered at the origin. Our algorithm is based on the following observation. A disc D of radius ρ_i intersects λ objects $\mathcal{P}_j$ with $j \leqslant i$ if and only if the center of D is covered by λ of the Minkowski sums $\mathcal{M}_j(\rho_i)$ with $j \leqslant i$.

Our algorithm globally works as follows. We consider the ρ_i in decreasing order. By the observation above, we can compute the maximum number of objects in $\mathcal{S}_i$ intersected by any disc of radius ρ_i as follows: compute the Minkowski sums $\mathcal{M}_j(\rho_i)$ for $j \leqslant i$, and find the cell in the arrangement defined by the Minkowski sums that is contained in the maximum number of Minkowski sums. Implemented in this straightforward manner, the algorithm is not very efficient. We can make it more efficient by using the fact that we treat the radii in order of decreasing value, as follows. Suppose that the maximum number of objects intersected by any disc of radius ρ_j for any $j < i$ is λ. (Of course, we only consider the objects in $\mathcal{S}_j$ when we consider discs of radius ρ_j). Suppose furthermore that there is a disc of radius ρ_i intersecting more than λ objects from $\mathcal{S}_i$, so that we have to increase the current value of the density. Because $\rho_j > \rho_i$ for all $j < i$, this disc must intersect $\lambda + 1$ objects, one of which is the object $\mathcal{P}_i$ that now for the first time comes into consideration. Hence, we can restrict our attention to the part of the arrangement defined by the Minkowski sums $\mathcal{M}_j(\rho_i)$ lying inside $\mathcal{M}_i(\rho_i)$. To determine the objects whose Minkowski sums intersect $\mathcal{M}_i(\rho_i)$, we do a range query with $\mathcal{M}_i(2\rho_i)$ in $\mathcal{S}_{i-1}$. We thus obtain the following algorithm.

1. $\lambda_1 := 1$.
2. **for** $i := 2$ to n
3. **do** Determine the set S of objects in $\mathcal{S}_{i-1}$ intersecting $\mathcal{M}_i(2\rho_i)$. Compute $\mathcal{M}_j(\rho_i)$ for each $\mathcal{P}_j \in S$, and compute the arrangement defined by these Minkowski sums inside $\mathcal{M}_i(\rho_i)$. Check whether there is a cell in this arrangement covered by λ_{i-1} of these Minkowski sums. If so, set $\lambda_i := \lambda_{i-1} + 1$, otherwise set $\lambda_i := \lambda_{i-1}$.
4. **return** λ_n

The range searching query we have to perform in Step 3 to find the objects in $\mathcal{S}_{i-1}$ intersecting $\mathcal{M}_i(2\rho_i)$ is done using a generalized version of the data structure of Schwarzkopf and Vleugels [17], as described by Vleugels [20]. The preprocessing time of the data structure for a λ-low-density scene is $O(n \log^3 n + \lambda n \log^2 n)$, and the time to perform a query is $O(\log^2 n + \lambda)$. This structure allows queries with all possible radii, so it has to be built only once at the start of the algorithm.

To compute the Minkowski sums and their arrangement, we spend quadratic time in the number of objects reported by the range query. Because the range with which we query has size $O(\rho_i)$, we know that the number of reported objects is $O(\lambda_{i-1})$. Hence, the time spent in stage i is $O(\log^2 n + \lambda_i^2)$, leading to a total running time of

$$O(n \log^3 n + \lambda n \log^2 n) + \sum_{i=2}^{n} O(\log^2 n + \lambda_i^2),$$

which is $O(n \log^3 n + \lambda n \log^2 n + \lambda^2 n)$. Note that the running time is $O(n \log^3 n)$ if λ is a constant.

Theorem 4.1 *The density of a planar scene $\mathcal{S}$ consisting of n polygonal objects can be computed in $O(n \log^3 n + \lambda n \log^2 n + \lambda^2 n)$ time, where λ is the density.*

Clutteredness. Let $\mathcal{S}$ be a planar scene consisting of n objects, and let V be the set of vertices of the bounding boxes of the objects in $\mathcal{S}$. We call a square *empty* if its interior does not contain any vertex in V. Recall that the clutter factor of $\mathcal{S}$ is the maximum number of objects in $\mathcal{S}$ intersected by any empty square. To compute the clutter factor we can restrict our attention to empty squares that are maximal in the sense that they are not contained inside some other empty square. Any maximal empty square has either three points from V on its boundary, or it has two points on opposite sides. Hence, the center of such a square must either lie on a vertex of the L_∞-Voronoi diagram of V, or it must lie on the vertical or horizontal part of a L_∞-bisector that occurs in the Voronoi diagram—see Figure 5.

Our algorithm to compute the clutter factor is thus as follows.

1. Compute the L_∞-Voronoi diagram of the set of bounding-box vertices of the objects.

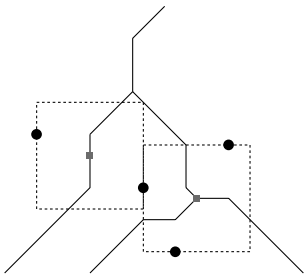

Figure 5: Maximal empty cubes are centered on the L_∞-Voronoi diagram.

2. For each vertex of the Voronoi diagram, compute the number of objects intersecting the maximal empty square centered at that vertex.

3. For each vertical or horizontal part of a L_∞-bisector that occurs in the Voronoi diagram, sweep a maximal empty square along that part to compute the maximum number of objects intersected by any maximal empty square centered on that part.

Step 1 can be done in $O(n \log n)$ time [11].

For Step 2 we store the vertices of the objects in a range tree, and the edges in a segment tree [16]. We apply fractional cascading [7] to these data structures, so that we can report the number of objects intersecting any square in $O(\log n + A)$ time, where A is the number of answers.

To implement Step 3, we first determine the objects intersected by the sweep area of the square along the L_∞-bisector part we are dealing with, using the range searching data structure. If there are A such objects, this takes $O(\log n + A)$ time. Because the sweep we have to perform on the objects intersecting the sweep area is basically one-dimensional, it can be performed in $O(A \log A)$ time.

To analyze the final time complexity of the algorithm it remains to observe that for a κ-cluttered scene, the number of objects reported for each query in Step 2 of the algorithm is obviously bounded by κ. Moreover, the number of objects reported for each query in Step 3 is bounded by 2κ, because each sweep area can be covered by two empty squares. This leads to the following result.

Theorem 4.2 *The clutter factor of a planar scene $\mathcal{S}$ consisting of n polygonal objects can be computed in $O(n \log n + n\kappa \log \kappa)$ time, where κ is the clutter factor.*

Note that the running time is $O(n \log n)$ if the clutter factor is a constant.

Simple-cover complexity. For this model we would like to compute for a given parameter δ and a given scene $\mathcal{S}$, the δ-simple-cover complexity of $\mathcal{S}$. Currently we have no efficient algorithm to compute for this; the best algorithm we have runs in exponential time. We also do not have a proof that computing the δ-simple-cover complexity is NP-complete, although the seemingly-simpler DISC-PACK problem—which consists of covering a set of n planar points by a given set of discs—is NP-hard [9], and a restricted version, in which the discs are identical, is NP-complete [9]. We leave the determination of the computational complexity as an open problem.

5 Experimental verification of the models: a case study

Below we give some experimental results on the applicability of the models for one particular type of scenes: polyhedral terrains, as encountered frequently in GIS.

Because many algorithms working on terrains in fact work with 2D projections of these terrains, we shall do the same. Thus our test scenes are collections of triangles in the plane. The test scenes

test scene	n	fatness		density		clutter factor		scc
		β	avg	λ	avg	κ	avg	σ
Death Valley	100	$2.44 \cdot 10^{-4}$	$3.63 \cdot 10^{-2}$	20	10.5	22	6.52	1.01
	200	$2.20 \cdot 10^{-5}$	$3.96 \cdot 10^{-2}$	16	9.87	22	5.63	0.622
	500	$1.60 \cdot 10^{-4}$	$4.45 \cdot 10^{-2}$	19	9.24	17	4.75	0.471
	1000	$1.42 \cdot 10^{-5}$	$4.96 \cdot 10^{-2}$	21	9.05	17	4.57	0.366
	2000	$6.34 \cdot 10^{-6}$	$5.37 \cdot 10^{-2}$	20	8.82	21	4.49	0.335
	5000	$4.53 \cdot 10^{-6}$	$5.73 \cdot 10^{-2}$	27	8.82	22	4.35	0.315
San Bernardino	100	$2.65 \cdot 10^{-4}$	$4.65 \cdot 10^{-2}$	13	8.32	11	4.32	0.332
	200	$2.65 \cdot 10^{-4}$	$4.66 \cdot 10^{-2}$	16	8.83	14	4.49	0.264
	500	$2.91 \cdot 10^{-5}$	$5.37 \cdot 10^{-2}$	15	8.75	15	4.26	0.270
	1000	$1.81 \cdot 10^{-4}$	$5.61 \cdot 10^{-2}$	17	8.61	17	4.26	0.252
	2000	$1.67 \cdot 10^{-6}$	$5.59 \cdot 10^{-2}$	23	8.80	21	4.33	0.265
	5000	$2.03 \cdot 10^{-6}$	$5.80 \cdot 10^{-2}$	25	8.76	19	4.30	0.275

Table 1: Experimental results for two of the test scenes.

were generated as follows. We used so-called DEM files, describing certain areas in the US. Such a DEM file specifies the elevation of a set of sample points in the area at hand; the sample points form a regular grid. Using the so-called VIP method [8] the n most important points were extracted for the following values of n: 100, 200, 500, 1000, 2000, and 5000. The test scenes were then generated by computing the Delaunay triangulation of the extracted sample points. Below we describe the outcome of our tests on test scenes taken from Death Valley and from San Bernardino; the results we got for other areas were similar. Table 1 lists the numerical data that we acquired from our experiments on these test scenes; Figure 6 shows the actual scenes we got for Death Valley and for San Bernardino, both with $n = 1000$.

Fatness. The fatness of a scene is determined by the least fat object in that scene. The presence of a single outlier in the scene—that is, an object that is considerably less fat than the other objects—can have a dramatic impact on the performance of algorithms with a running time depending on the fatness of the scene. The results in Table 1—particularly those for the Death Valley scenes—confirm that this is indeed a potential danger: the fatness of the scene is sometimes a factor 1,000 smaller than the average fatness of its triangles. Over 95 percent of the triangles in the Death Valley scenes, and over 99.5 percent of the triangles in the San Bernardino scenes, have a fatness that is within a factor ten of the average fatness.

To illustrate the impact of outliers on the performance of algorithms, we consider the range searching algorithm by Overmars and van der Stappen [14]. The planar version of this algorithm has a quadratic dependence on the fatness. In the Death Valley scenes, exclusion of the least fat five percent of the triangles results in an overall fatness that is approximately one tenth of the average fatness of the complete scene; the fatness of the left-out five percent is a factor 100 smaller than this. Therefore, including these five percent least-fat triangles would increase the running time of the algorithm by a factor up to 10,000.

The experiments suggest that—at least for this type of TINs—algorithms with a running time depending on the minimal fatness of the scene will perform badly, whereas algorithms whose running time depends on the average fatness may perform well. Furthermore, it is interesting to see that the average fatness is fairly constant over different test scenes generated from the same area.

Low density. In the algorithm for computing the density, we add the objects one by one, keeping track of the density of the objects added so far. This is done by repeatedly computing the intersection of the Minkowski sum of the newly added object with the Minkowski sums of the priorly added objects (see Section 4 for further details).

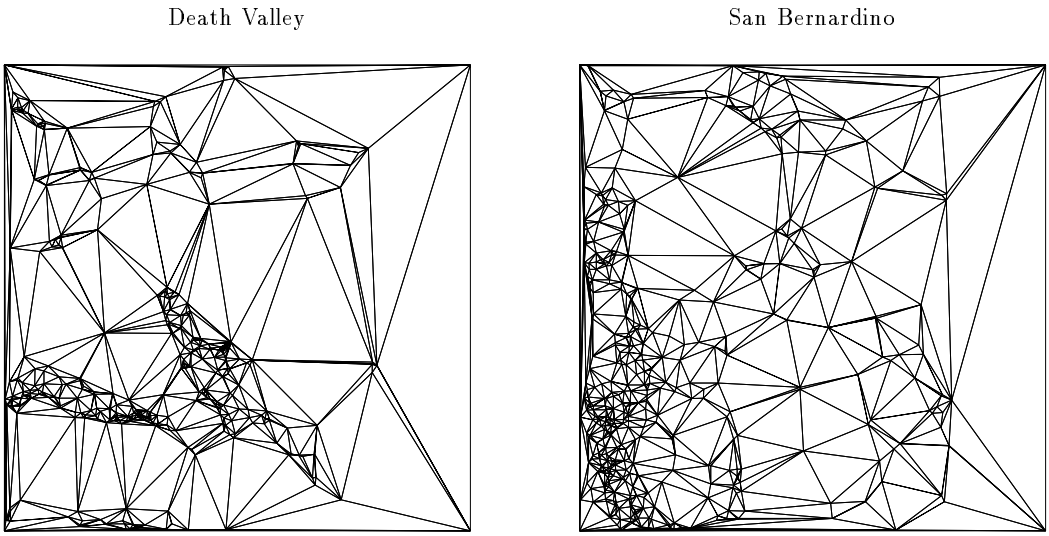

Figure 6: The Death Valley and San Bernardino test scenes for $n = 1000$.

For our test scenes, the density typically varies between approximately fifteen and twenty-five (see Table 1). We compared the density to the average number of (larger) objects intersecting a disc from the collection of discs with radii from ρ_i $(1 \leqslant i \leqslant n)$ and touching two objects from $\mathcal{S}$. This average number does not differ much from the density (see also Table 1). This suggests that, unlike with fatness, the density of a scene supplies a good estimation of the average number of (larger) objects intersecting a region.

The density of our test scenes appears to increase somewhat with a growing number of points. This is hardly surprising because the density of a scene is determined by the single disc intersected by the largest number of objects. In contrast, the average number seems to either decrease with a growing number of points—as in the Death Valley scenes—or remain relatively constant, which is the case for San Bernardino. This suggests that regions with high density are introduced by the need to approximate the terrain with only few data points, especially if the terrain contains steep ridges. The approximation provided by the triangulation grows more accurate as the number of data points increases.

Clutteredness. The planar clutter factor of a scene is determined by the square without bounding-box vertices in its interior that intersects the largest number of objects. In the scenes we consider, the clutter factor typically lies between eleven and twenty-two (see Table 1); although the exact clutter factor varies somewhat with the number of points, it does not seem to increase significantly with a growing number of points.

We compared the clutter factor to the average number of objects intersected by a square from the collection of all squares touching three bounding-box vertices (but without bounding-box vertices in their interiors). Although the clutter factor lies in the same range as the density, this average number is in all cases lower than the average number of larger objects intersecting a disc (see again Table 1). In our test scenes, the average number of objects intersecting a square is typically less than five, and varies little for different scenes. This suggests that only few squares are intersected by relatively many objects. Indeed, our experiments show that only approximately five percent of the squares intersect nine or more objects, while typically less than 0.5 percent intersect twelve or more objects. Like the density, the average number either decreases or remains relatively constant for larger scenes.

Simple-cover complexity. Currently we have no algorithm to compute the simple-cover complexity of a scene. Instead we build smoothed quadtrees for the scenes, the leaves of which are (square) cells that intersect at most δ objects, where δ is clearly bounded from below by the maximum number of objects touching in a single point. The trees are smoothed in the sense that any two neigboring cells differ at most a factor two in side length. The resulting squares form a δ-simple cover of the scene. In our experiments we chose $\delta = 23$, which is just above the maximum number of objects touching in a point in any of our test scenes. We believe the size of the quadtree to be closely related to the simple-cover complexity, as we do not expect situations in which a small crowded corner of an otherwise empty cell would lead to many recursive splits of that cell without actually reducing the maximum number of objects in its subcells. The fact that the tree is smoothed makes it possible to cover the (δ-simple) squares by a similar number of δ'-simple discs, where δ' lies within a constant factor of δ. Table 1 shows that the ratio of the number of quadtree leaves and the number of objects is indeed a small constant.

6 Conclusions

We have studied the relations between various scene models that have been proposed in the computational-geometry literature, developed algorithms to compute the model parameters, and experimentally investigated the appropriateness of some of the models for certain types of TINs. We believe that our paper points into an important research direction, which can help to bridge the gap between theory (computational geometry) and practice (its application areas). Future work in this direction should include:

- For a number of important application domains, there should be models that are on one hand general enough to include (almost all) scenes in the domain, and on the other hand restrictive enough so that complicated, hypothetical input scenes are precluded. Ideally, these models should be widely accepted as being realistic for their respective application areas. To achieve this, experimental research in the spirit of Section 5 is needed.
- New algorithms should be developed that solve a problem from a given application domain efficiently within a model appropriate for that domain. Here the focus should be on problems for which no efficient solution is available for general, unrestricted scenes.

We do not claim that the models we have studied in this paper are the best or most intuitive ones. Nevertheless, we believe that scenes in many areas will be uncluttered and have small simple-cover complexity. Solutions based on octree-like structures as described by de Berg [5] and Mitchell et al. [13] will perform well in that case. For many application domains, there may well be models that are more intuitive or more appropriate than these models. If this is the case, one would of course like to work with these models. We want to emphasize, however, that it is important to stick to a few, generally accepted models, in order to be able to develop a coherent model-based theory.

References

[1] P. K. Agarwal, M. J. Katz, and M. Sharir. Computing depth orders for fat objects and related problems. *Comput. Geom. Theory Appl.*, 5:187–206, 1995.

[2] P. K. Agarwal and J. Matoušek. On range searching with semialgebraic sets. *Discrete Comput. Geom.*, 11:393–418, 1994.

[3] H. Alt, R. Fleischer, M. Kaufmann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig. Approximate motion planning and the complexity of the boundary of the union of simple geometric figures. *Algorithmica*, 8:391–406, 1992.

[4] M. de Berg. Linear size binary space partitions for fat objects. In *Proc. 3rd Annu. European Sympos. Algorithms (ESA '95)*, volume 979 of *Lecture Notes in Computer Science*, pages 252–263, 1995.

[5] M. de Berg. Linear size binary space partitions for uncluttered scenes. Technical Report UU-CS-1998-12, Dept. Comput. Sci., Utrecht Univ., 1998.

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

[7] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.

[8] Z. Chen and J. A. Guevara. System selection of very important points (VIP) from digital terrain models for constructing triangular irregular networks. In *Proc. 8th Internat. Sympos. Comput.-Assist. Cartog. (Auto-Carto)*, pages 50–56, 1988.

[9] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, 12(3):133–137, 1981.

[10] M. van Kreveld. On fat partitioning, fat covering, and the union size of polygons. *Comput. Geom. Theory Appl.*, 9:197–210, 1998.

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

[12] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. *SIAM J. Comput.*, 23:154–169, 1994.

[13] J. S. B. Mitchell, D. M. Mount, and S. Suri. Query-sensitive ray shooting. *Internat. J. Comput. Geom. Appl.*, 7:317–347, 1997.

[14] M. H. Overmars and A. F. van der Stappen. Range searching and point location among fat objects. *J. Algorithms*, 21:629–656, 1996.

[15] 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.

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

[17] O. Schwarzkopf and J. Vleugels. Range searching in low-density environments. *Inform. Process. Lett.*, 60:121–127, 1996.

[18] A. F. van der Stappen. *Motion Planning amidst Fat Obstacles.* Ph.D. thesis, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, October 1994.

[19] A. F. van der Stappen, M. H. Overmars, M. de Berg, J. Vleugels. Motion planning in environments with low obstacle density. Technical Report UU-CS-1997-33, Dept. Comput. Sci., Utrecht Univ., 1997, to appear in *Discrete Comput. Geom.*.

[20] J. Vleugels. *On Fatness and Fitness—Realistic Input Models for Geometric Algorithms.* Ph.D. thesis, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, March 1997.