Models of 2D-Scene Complexity: A Look at the Intrinsic Complexity of Scenes

by

Mark R. Sauer

B.Sc. (Computer Science and Statistics)

University of Toronto, 1997

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

we accept this thesis as conforming
to the required standard

The University of British Columbia

August 1999

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date 10-Sept-1999

DE-6 (2/88)

Abstract

The worst-case time and space complexity of many algorithms in computational geometry is high when expressed as a function of the input size. This can make algorithms seem inefficient in general. Usually, however, the algorithm will not run with identical times on every input of a given size. Intuitively, inputs of higher-complexity cause higher run-time and/or space costs in an algorithm. The size of an input is only one factor contributing to the complexity of an input. As such, input size is not always an adequate predictor of the behaviour of an algorithm. This thesis looks at models capturing the intrinsic complexity of a scene of objects. The models identify quantifiable properties which can be used as additional parameters in the time and space complexity of computational geometry algorithms. In particular properties of two-dimensional scenes will be explored. One problem is then looked at and analyzed with respect to the new models: the Binary Space Partition problem. Under the assumption that an input scene is considered to be simple by our most general model, the analysis reveals that such a scene will have a linear sized binary space partition.

Contents

Acknowledgements

First and foremost I would like to thank my supervisor, David Kirkpatrick, for his unending patience, advice and encouragement while I was writing this thesis. He and I had many hours of discussions about the material in this thesis. I learned a great deal from these discussions. They ultimately created this thesis. I would also like to thank my girlfriend, Lauri Gale, for her constant love, encouragement and support. Additionally, my friend Jan van der Valk and my parents deserve many thanks for their love and support.

MARK R. SAUER

The University of British Columbia

August 1999

Chapter 1

Introduction

One very important aspect of computer science is the study of the computational complexity of various problems. In doing this, one usually begins with a statement describing the input to the problem and the output expected. Different methods by which the problem would be solved by a particular model of a computer are analyzed, leading to a number of upper and lower bounds for the time and space complexity of the problem. In doing this analysis, the worst possible, most complicated input is often looked at for determining the output from the input. Focusing on the worst case scenario ensures that regardless of which input is supplied to the algorithm, it will not be worse than the worst case input. Functions based on the input size are determined that give upper and lower bounds on how much time and space the algorithm will need if presented with the worst case input of a certain size to compute the output.

There are many problems for which a worst case analysis leads to a very accurate description of how hard the problem is to solve, regardless of the input presented of a given size. As an example of such a problem, suppose you are given

a list of points in the plane, and you want to compute their bounding box, which is the smallest axis-aligned box that contains all of the points in the list. Assuming that no properties of the input list are known, it is impossible to determine the bounding box without reading the entire input. This requires a linear worst case lower bound on the running time of any algorithm solving this problem. This will be true regardless of the input list supplied to the problem solver (the algorithm).

However, there are also problems where the bounds determined by a worst case analysis are not very accurate for a large class of size n inputs. That is to say that the algorithm will perform faster or use less memory than predicted by the worst case lower bound on time and/or space complexity for a potentially large number of size n inputs. An example of this is the quicksort algorithm for sorting an array of n objects. A worst case analysis of this algorithm shows that it requires $\Theta(n^2)$ time to sort the array - seemingly slower than a $\Theta(n \log n)$ sort like heapsort or mergesort. However, the quicksort algorithm is one of the most popular sorting algorithms in practice because it sorts faster than other comparison-based sorting algorithms. To understand why this is the case, we have to look more closely at the analysis of the quicksort algorithm. When doing this, we discover that the algorithm gives bad behaviour on a very specific type of input, one which causes a very unbalanced pivot point on most iterations (we assume the reader is familiar with the quicksort algorithm, and refer the reader to Cormen et al. [5] or any other good introductory algorithms text for more details on the algorithm). It can be shown that for any constant $k > 0$, all but $O(\frac{1}{n^k})$ of the $n!$ possible input permutations yield an $O(n \log n)$ running time. The majority of all possible length n inputs to the quicksort algorithm cause it to give $O(n \log n)$ behaviour, where the hidden constants are quite low. The worst case analysis for this algorithm does not reveal

much about how efficient the algorithm is on most input arrays.

Suppose you have an algorithm and one input to that algorithm, and you are interested in obtaining a lower bound on how long the algorithm will take on that input. In this case, your best bet is to look at the worst case lower bound on the running time for the algorithm. This will give a guarantee (provided that the input size is large enough), that the algorithm will do no worse than what the lower bound indicates. If, however, you are planning on running a long stream of inputs of a given size through an algorithm, the worst case bound may be an overly pessimistic measure of the run time needed to process the entire stream of inputs. There are three popular algorithm analysis techniques, based on the input size, that give a more accurate picture of the behaviour of such an algorithm on a long stream of inputs. The techniques are average case analysis, expected case analysis and amortized analysis.

Average case analysis is a special case of expected case analysis. In average case analysis, every input to an algorithm of size n is considered to be equally likely. This could be because of the nature of the input process creating the inputs, or because the algorithm itself chooses to look at the data in a random order as in a randomized algorithm. The average case running time for an algorithm is computed as an average over the entire set of size n inputs of the running time needed by the algorithm on each input.

Expected case analysis generalizes average case analysis. The difference is that whereas average case analysis places a uniform distribution across the size n inputs, expected case analysis allows an arbitrary distribution, known as the *input model*, to be placed on the inputs. The input model is a statistical model that assigns probabilities to each input in the set of possible size n inputs. These probabilities

tell us how often each input will occur over an infinitely long stream of inputs to the algorithm. Once an input model has been selected, the expected running time for the algorithm is computed by first determining the running time of the algorithm for all inputs, and then by taking the statistical expectation of all of these running times (weighted by the input model).

Average and expected case analysis give an idea of what to expect when an algorithm is run repeatedly over a long sequence of inputs. Any one input may have bad performance, but over a large number of inputs to the algorithm, the measured average performance will tend to the expected running time for the algorithm, assuming that an accurate input model has been chosen. That is to say the expected running time of the algorithm gives a probabilistic bound on the amount of time needed to run the algorithm on a series of inputs of a given size. As the size of the input series increases, the probability of the bound holding increases (this follows from the law of probability).

In amortized analysis, the complexity of a series of n operations is looked at (for say maintaining a certain data structure). Amortized analysis is useful for algorithms whose computational complexity may vary depending on the current state of the data structures the algorithm is maintaining. Many data structure maintenance algorithms exhibit the behaviour that they have a small running time for a sequence of operations, then after a number of these inexpensive operations, a more expensive operation with a longer running time occurs. Amortized analysis looks at the ratio between the cheaper operations and the expensive operations to see if the extra cost of the expensive operation can somehow be explained or paid for by the number of cheaper operations performed before an expensive operation. A way of thinking about this is that each operation performed costs a certain amount

of money to perform. The amount of money needed should correlate to the running time of the operation. Then for each operation to occur, a certain amount of money is supplied to perform each operation. Now, if the cheap operations cost less than the amount of money given to the algorithm to perform its work, the algorithm saves some money. This money can then be used to pay for future operations. That is, if at some point an expensive operation comes up that needs more money to run than is being supplied to the algorithm, the algorithm can use its savings to pay for its operation. In amortized analysis, one tries to determine the smallest amount of money that needs to be paid for each operation so that the system always has enough money to pay for any action performed by the algorithms. This amount of money is the amortized complexity per operation of the algorithm.

An example may help at this point. Suppose you want to maintain a list of elements, and you will be adding elements to the list from time to time. You do not want to wait a very long time to add elements into the list, and you also do not want to waste a lot of memory having empty slots available for future insertions. You are happy if there are never more open slots available than items in the list. A list with these properties can be maintained by starting with one empty slot, and then whenever the list fills up, doubling the size of the list, copying over the old list to the new memory. Each insertion to the list will be cheap if the memory is already there, say it costs one dollar. The expensive operation is when the old list must be copied over to the new list, say it costs n dollars (where n is the current number of items in the list). This doubling of the list size will always occur after $n/2$ elements have been added into the list from the previous list doubling operation. Thus if we give 3 dollars per operation, each cheap operation will be able to save 2 dollars. When the list doubling needs to take place, $n/2 * 2 = n$ dollars will be saved - enough to

pay for the expensive operation. Thus the amortized complexity of this algorithm is 3. In this case we would say that the algorithm has constant amortized complexity. It is noteworthy that the worst case complexity of this algorithm is n, since the worst case cost of a single step would be one in which the list copy operation was performed. Looking only at the worst case complexity of an operation would lead us to conclude that the algorithm would require $O(n^2)$ steps for a sequence of n insertions to the list. However, the amortized analysis shows that only $O(n)$ steps are needed for a sequence of n insertions to the list.

All of the analysis techniques discussed so far are based on input size. In every case a function is computed based on the input size that places a bound on how much time and/or space an algorithm will need to complete its task. While these bounds are accurate, better bounds can be achieved by taking a look at additional properties of the input. In many problems, the complexity of dealing with a certain input depends on some property of the input. An example of this is so-called Jarvis march algorithm for computing the convex hull of a set of points in the plane. This algorithm starts with a point known to be on the hull (the left-most point in the set), and then for each point on the hull, it scans the remaining unclassified points to find the next point on the hull. The exact details of the algorithm and the convex hull problem are beyond the scope of this thesis, and refer the reader to Cormen et al. [5] for more details, but the point is that the analysis of the algorithm indicates that it runs in time $O(nh)$ where h is the number of vertices on the convex hull. This is so because $O(n)$ vertices are scanned at each convex hull vertex. Analyzing the Jarvis march algorithm only as a function of the input size would lead to a $O(n^2)$ run time bound. It is well known that for a uniform distribution of n points in the plane, the average number of points on the convex hull of the points is $O(logn)$. Because

of this fact it is concluded that the Jarvis march algorithm will on average perform with complexity $O(n \log n)$; this shows a significant improvement over worst-case analysis. This is the sort of improvement that is possible by taking into account properties of the input to an algorithm. In this thesis, we look at properties which attempt to capture the intrinsic complexity of two dimensional scenes of objects. The intent of these properties is that they should improve the analysis of geometric algorithms, giving more accurate bounds on the time complexity of algorithms.

The properties should represent natural characteristics of a scene. They should be quantifiable so that the time complexity of algorithms can be related to the property. When discussing the properties, we will talk about models of scene complexity. This is just a more formal way of talking about the technique of measuring the complexity of scenes with respect to a particular property. The goal of the models is to explain the differences in the behaviour of algorithms on scenes of the same size.

1.1 Overview of the Properties

We will now give an overview of the properties that we look at in the remainder of the thesis. In this thesis we will mainly be dealing with results about planar scenes. It is our hope to give the reader a feeling for the intuition behind the properties without getting into the formal specifications of the properties. The formal definitions can be found in the discussion in chapters 2 and 3.

1.1.1 Fatness

What are properties of a scene that make it simple? This seems like a simple enough question. To answer it, we first look to the literature. A number of authors have

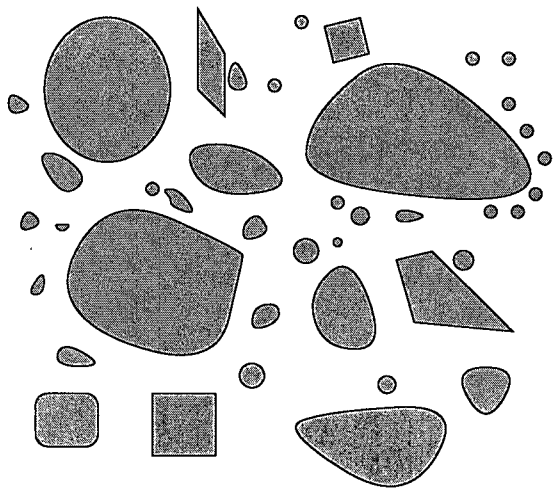


Figure 1.1: A scene consisting of fat objects, with no long and skinny parts. Note that small objects can be placed close to large objects.

talked about the property of *fatness* of objects, e.g. [1, 2, 13, 19, 9, 17, 20]. The basic idea is that a scene is "simple" if it is composed of disjoint *fat objects*, which are objects that lack long and skinny parts. Very roughly, the fatness of an object tells you how close the object is to a circle. The fatness of a scene is the fatness of the least fat object in the scene, as this object is the limiting factor in the model. Because each object is fat in a scene of fat objects, the nature of the interactions between a given fat object with other close objects is limited and rather simple. See figure 1.1 for a picture of a scene of fat objects. Note that the area between adjacent objects is simple and not too complicated to describe. This model does not forbid scale differences between the sizes of adjacent objects either; large objects can be adjacent to small objects.

Putting these observations together gives us some insight into the real underlying characteristic of a scene that a scene of fat objects gives you. Looking at such scenes carefully reveals that the number of objects of comparable size in the

"neighbourhood" of any object in the scene is limited. That is to say that although a given object may have a large number of small neighbours, it cannot have more than a constant number of large neighbours. Suppose you are looking at the scene through a disc shaped window. Wherever you place this window over a scene of fat objects, you will discover that there are never more than a constant number of objects of size comparable to the radius of the window through which you are looking. It is this property of uniformity at any scale that makes a scene of fat objects simple.

1.1.2 Density

The density model takes the property observed for scenes of fat objects and makes it the defining characteristic. That is to say that the model considers a scene to be simple if each object has a small number of comparable size neighbours. More precisely, the density model requires that wherever you place your disc window over the scene, it will intersect no more than a constant number of objects of size comparable with the radius of the window. The density of a scene is the maximum number of objects that can be intersected by a disc window counting only those objects that have radius larger than or equal to the radius of the window. The density model states that a scene is not complicated if its density is constant.

The power that this model has over the fatness model is that now the shape of the objects is no longer restricted. Rather than being only a function of the shape of the objects, it is now a function of the shape and placement of the objects. See figure 1.2 for a comparison of a scene of fat objects, and a similar scene that has low density. Because no restriction is placed on the shapes of objects, it is easy to see that this model is more general than fatness. For some applications, this

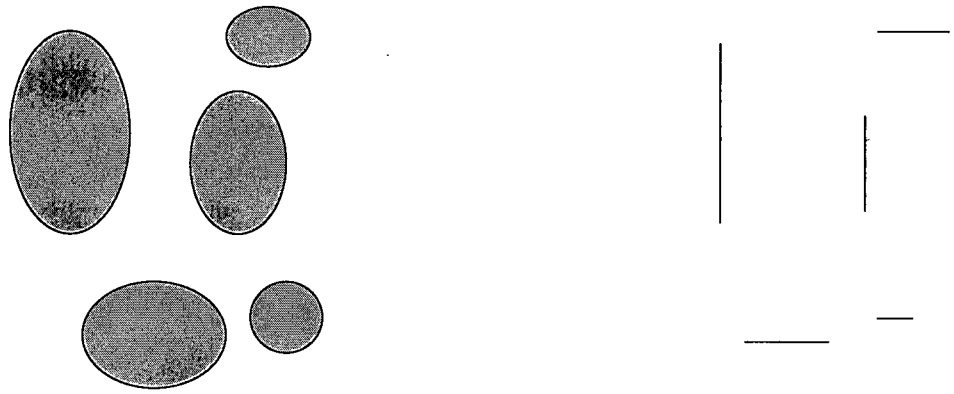


Figure 1.2: On the left is a scene of fat objects. On the right is a similar scene that has low density.

uniformity of the density across the scene is quite useful. For other applications, it can actually be a drawback in the following sense. Consider a scene that is rather sparse throughout most of the scene, but has a single region where the object density is high. The density of the scene would then be the density of this single region. This parameter would overestimate the actual density of most of the scene. Accordingly, some algorithms will operate efficiently when supplied an input containing a small number of dense regions (up to a certain extent); in such a case the density of the scene fails to accurately predict the algorithmic behaviour. An example exhibiting this behaviour is the Binary Space Partition problem, which we look at in more detail in section 3.5 of chapter 3.

1.1.3 Guarding Sets

The next model states that the complexity of a scene of objects relates to the number of points needed to provide a good approximation of the scene. A set of points provides a good approximation of the scene if wherever a box intersects more

than a constant number of objects from the scene, the box also intersects at least one point. Thus a sufficient guarding set is a set of points that provides a good approximation to the scene. The points in the set are the *guards*. This model appeared in a paper by de Berg et al. in [10], and is more general than the previous two models.

The idea of determining the minimum size guarding set for a scene differs intuitively from the previous notions in that it is a more abstract way of measuring the complexity of a scene. Whereas before we were measuring fatness of objects, or counting objects in windows, we now have to consider different numbers of and locations of points to see if they will provide a sufficient sized guarding set. Finding a minimum sized guarding set for a scene is probably hard to do. However, most of the time, we do not need to know the minimum sized guarding set, we just want to know, say, if a linear sized guarding set exists for the scene. This can be done for certain classes of scenes. We will see one such method in our next model, clutteredness.

1.1.4 Clutteredness

A special case of the guarding set model is the clutteredness model. This model asks whether the bounding-box corners of the objects in the scene provide a sufficient guarding set. The clutter factor of the scene is the maximum number of objects that is permitted to intersect a square that does not contain any bounding-box corners. This model was actually introduced before the guarding set model in a paper by de Berg. [6]. It is less general than the guarding set model, but fairly straightforward to compute. Even though a number of problems have been analyzed with respect to this model, the model has some unsatisfying characteristics. One drawback is

that a scene with a certain clutter factor can have its clutter factor lowered with the addition of new objects - this would happen if the new objects introduced to the scene caused bounding box corners to appear in a region with a high clutter factor. This is not as bad as it seems at first glance, since if we are relating the computational complexity of a problem to the clutter factor of the input, having the clutter go down with a new object has the effect of improving our analysis of the problem on the larger scene. Nonetheless, it seems more natural to have a model that is monotonic in this sense - i.e. that is a scene's complexity can not be made to go down with the addition of objects. If a scene's complexity can go down with the addition of objects, then it seems that somehow the original scene was not as complex as was measured by the property. It is this aspect that we find unnatural about this model.

1.1.5 Simple-Cover-Complexity

The final model from the literature, looked at in this thesis, is known as simple-cover complexity. This model first appeared in a paper by Mitchell et al. [14] and was modified by de Berg et al. [9] to a form close to the definition used here. The idea in this model is that a scene is simple if it can be covered by an arrangement consisting of a small number of simple hyperdiscs. A hyperdisc is considered to be simple if it intersects a constant number of objects from the scene. It turns out that this model is identical to the guarding set model in the plane [10]. The thinking behind this model is that the number of regions in the scene, in some decomposition, that do not contain more than a constant number of objects can be related to the amount of work necessary to perform certain algorithms.

As an example, consider the ray shooting problem. In this problem, you are

given a scene of disjoint objects in two or three dimensions, the location of the start of a ray, and the direction that the ray is pointing. You want to know which object face the ray first intersects in the scene. One way of doing this would be to first do some preprocessing on the scene that would partition the scene into a number of regions that, individually, do not intersect too many objects. Once this is done, the ray shooting query would be performed in two steps. First the region containing the origin point of the ray would have to be located. To determine which object the ray intersects, the constant number of objects intersected by the region would have to be looked at. If none of the objects in the first region intersect the ray, then the next region will be selected by determining which region the ray enters. All of the objects intersecting that region would then have to be checked. In this way, eventually the object face first intersected by the ray will be detected. If it does not hit any objects, the algorithm will terminate because it will detect when the ray goes beyond the boundary of the partitioning of the of scene. The performance of this algorithm is dependent on how long it would take to locate the region containing the origin of the ray, and the number of regions that the ray passes through before the ray hits an object. Since only a constant number of objects must be checked in each region, the performance of the ray shooting query algorithm is dependent in a linear fashion on the number of regions the ray passes through (which is the simple-cover-complexity of the ray) plus the time needed to find the origin of the ray, which, using the algorithm of Mitchell et. al. [14], can be done in $O(\log R/r)$ time (where R is the diameter of the scene of objects and r is the radius of the largest simple hyperdisc intersecting the origin of the ray). The amount of storage needed for this solution is proportional to the simple-cover-complexity of the scene. Previous algorithms for this problem required $O(\sqrt{n}\log n)$ query time to have a linear sized

data structure [4, 12], and to get $O(\log n)$ query time all solutions require $\Theta(n^2)$ space [14].

1.1.6 Weighted Cover Complexity

While the aforementioned models do represent properties of complexity in scenes, they do not go far enough. Intuitively, it seems that a scene should not be seen as complicated because there is one region of high complexity within the scene. Yet, most of the aforementioned models possess this property. One highly complex region in the scene will have the effect of blowing up the apparent complexity of the whole scene for most of these models; this remains true even if the complicated region represents a very small fraction of the area covered by the scene. Although guarding sets and simple-cover-complexity do not directly suffer from this problem, they do tend to provide a high estimation of the complexity of a scene consisting of set of n parallel lines. These observations have led us to find some new measures of scene complexity. From this point on, we look at a scene of objects as if it were composed of line segments. In this way, it makes sense to talk about the endpoint of an object, as this is just the endpoints of the line segments composing the scene. These endpoints will also be referred to as *representatives* of the objects in the scene.

Our new models are similar to the simple-cover-complexity model in that they are based on a cover of discs or quadtree cells. However, our models differ in their restrictions on the size of the covering objects, and in the way each covering disc or quadtree cell contributes to the complexity. If an object intersects one of the covering cells, then it must either pass totally through the region, or end in that region. If the object passes entirely through a region, it adds to the complexity of the scene by partitioning the region into two sections. If, on the other hand, the

object ends in a region, it adds to the complexity of the scene by causing complicated interactions in the region. As a result, in our models we will allow any number of objects to pass entirely through a covering region, but we will bound the number of objects that end within a given region. Since the objects that pass entirely through a covering region have the effect of dividing the region into two pieces, we want to count the number of such objects throughout the scene; we define the weight of a covering region to be the number of objects passing entirely through a region. Adding up the weights of each covering region gives the weighted cover complexity of the scene.

These weighted cover complexity models better predict the behaviour of some algorithms on scenes with a small number of regions of high complexity. Figure 1.3 is an example of such a scene that has low complexity under one of our weighted models, but possesses 8 parallel lines, a structure which would cause the scene to have high complexity under the previous models. We look at a problem in section 3.5 of chapter 3 that can be analyzed with respect to our weighted cover complexity models. Doing so improves its analysis.

Weighted models differ in the shape of the covering object used to cover the scene. They differ in how complex they perceive various scenes to be. In this thesis, we look at disc covers and quadtree covers.

The disc cover is the natural extension of the simple-cover-complexity model into the weighted framework. An unfortunate problem that this model shares with the simple-cover-complexity model is that it seems to be hard to compute what the disc complexity is for a scene, as the weighted disc cover leading to the minimum complexity must be found.

To deal with the computability problems of the weighted disc cover, another

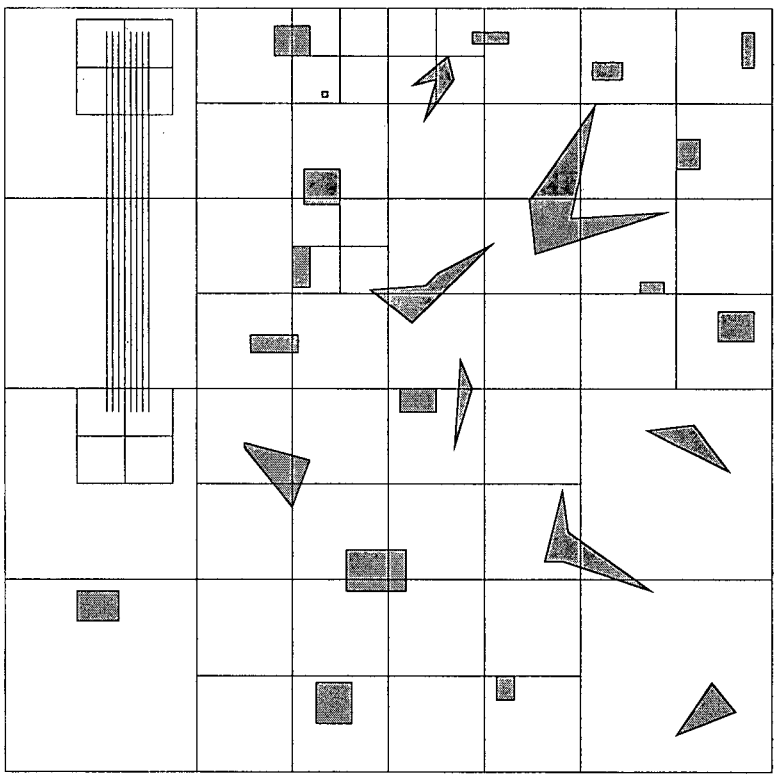


Figure 1.3: The following is a scene that would have high simple cover complexity, but for which it has low quadtree complexity. Each cell in this decomposition is permitted to have 4 corners in it.

cover type is used - the adaptive quadtree decomposition of the scene. In this cover, a quadtree is formed so that each cell has no more than a constant number of representatives. Quadtree cover complexity deals well with scenes that have clusters of object representatives in a given area. As an example of such a scene, see figure 1.3. In the figure, the quadtree algorithm quickly shrinks the cell size to the level of the endpoints. This prevents too many cells from intersecting the densely packed parallel lines. Even though the scene has relatively small quadtree cover complexity, the weighted disc complexity of the scene is arbitrarily high. We discuss the reason why this is the case in section 3.4.

1.1.7 Note on Generalizations

It should not be taken that the ultimate goal of this research is to find a model that considers the least scenes to be complicated without any proof that scenes deemed complicated are indeed hard to deal with by some algorithm. We are studying the relationships between various models. One relationship is the notion of generality which is defined as follows: model A is said to be more general than model B if A assigns a lower complexity to more scenes than B where each low complexity scene in B retains its low complexity in A. Focusing on more general models is interesting only if the new models provide better insight into the characteristics of scenes that tend to be easier to deal with by algorithms. In this way, a more general model should also be a better predictor of the actual behaviour of some existing algorithms.

The models mentioned form a hierarchy where the models get progressively more and more general. It is noted here that this does not imply that all problems that run well when given a low complexity scene according to one model, say fatness,

will necessarily run well when given a low complexity input according to another, more general model, say unclutteredness. In fact the opposite is sometimes true.

Take the example of the motion planning problem for bounded-reach robots. Keeping things simple, a bounded-reach robot is one which is bounded by how far it can reach from a reference point on the robot. The robot can be either the free-flying type, a robotic arm, or any other type, so long as it satisfies the bounded reach criterion. The motion planing problem asks how easy it is to determine a path from point A of the robot to point B (in the configuration space), such that the robot does not crash into any of the obstacles. The computational complexity of the free space of a scene of objects determines how difficult the motion planning problem is to solve. The worst case situation for a robot with f degrees of freedom is a complexity for the free space of $\Theta(n^f)$. However, if the scene consists of fat obstacles or has low density, the complexity of the free space is $\Theta(n)$ [18], which is an improvement for this class of scenes. De Berg et al. [10] have shown that in two dimensions, the complexity of the free space of an uncluttered scene is $\Theta(n^{\frac{f}{2}})$, which is an improvement over the worst case, but not quite as small as is the case for low density scenes. The analysis of the free space complexity for a bounded-reach robot shows that generalizing a complexity model sometimes causes algorithms to operate less efficiently under the more general model.

This problem reveals one more point about the usefulness of a model. New algorithms can exploit models to improve their performance on scenes that are measured to be uncomplicated by those models. It is productive to use the most general model possible that does not underestimate the complexity of the scene relative to the problem being solved.

The next two chapters of this thesis will look at the models described in

much more detail. The specific results to back up the above claims will be referred to, or shown in full detail. The final chapter will take analyze an application from computational geometry using this the above models. We will see that we can better predict the complexity of the problems mentioned by using the quadtree complexity of the input scene as a parameter to the analysis.

Chapter 2

Models of Scene Complexity

In this chapter we look at and expand upon a number of models of scene complexity mentioned in the introduction. For each model, we give its formal definition and show its relationships to previous models that have been described. For some models, we look at some problems that it has helped to analyze. The models we look at are fatness, density, guarding sets, clutteredness, and simple cover complexity. Before starting, we need to give some definitions.

In this chapter we will view a scene $\mathcal{S}$ as a set of n disjoint bounded objects. We are mainly dealing with properties about scenes that are contained in two dimensional euclidean space in this thesis, although we will quote results that have been proven to hold in higher dimensions in their stronger form. Sometimes we will view an object as a polygonal shape, and at other times as a collection of line segments - the context of the discussion should make this distinction clear. The bounding box of an object $\mathcal{O}$, denoted $bb(\mathcal{O})$ is the smallest axis aligned box containing $\mathcal{O}$. The minimally enclosing hyperdisc around an object $\mathcal{O}$, denoted $meb(\mathcal{O})$ is the smallest hyperdisc containing $\mathcal{O}$. The *size* of an object is the radius of the

minimally enclosing hyperdisc around the object. The *radius* of an object is the same as its size. A *set of representatives* of an object $\mathcal{O}$ is a finite set of points that will represent $\mathcal{O}$; these representatives could be the object's bounding box corners, endpoints or any other set of related points and would depend on the model used.

2.1 Fatness and Density

The first model of scene complexity that we will look at is known as fatness. It is a parameterized model that measures the minimum fatness of all objects in the scene. The fatness of an object represents, informally, how close the object is to being a disc. Intuitively, a fat object is one that does not contain any long and skinny parts. The fatness of the scene is the fatness of the least fat object. A scene that consists of objects of a certain minimum fatness satisfies a nice property that limits the number of neighbours of similar or larger size any object in the scene can have. The precise definition of the fatness of an object $\mathcal{O}$ is given as follows[1]:

Definition 1 (van der Stappen [17]) *Let $\mathcal{O} \subseteq \mathbf{E}^d$ be an object and let β be a constant with $0 \leq \beta \leq 1$. Define $U(\mathcal{O})$ as the set of all hyperdiscs centered inside $\mathcal{O}$ whose boundary intersects $\mathcal{O}$. We say that the object $\mathcal{O}$ is β-fat if for all hyperdiscs $B \in U(\mathcal{O})$, $vol(\mathcal{O} \cap B) \geq \beta \cdot vol(B)$. The fatness of an object $\mathcal{O}$ is defined as the maximal β for which $\mathcal{O}$ is β-fat.*

The maximal β occurs as the minimum over all hyperdiscs in $U(\mathcal{O})$ of the fraction of the hyperdisc covered by $\mathcal{O}$. As an example, a half plane in two dimensions would be $\frac{1}{2}$-fat because the smallest fraction occurs for any hyperdisc centered

[1]The original definition of fatness due to van der Stappen calls an object $1/\beta$-fat when we call it β-fat.

on the boundary of the half plane. A line segment is 0-fat since the area of intersection between a line segment and any hyperdisc is zero. A disc D is $\frac{1}{4}$-fat since the hyperdisc centered at a point on the boundary of D with radius equal to the diameter of D is four times the area of D. The *fatness of a scene* of objects is defined to be the minimum fatness over all of the objects in the scene. We will often talk informally about a scene consisting of fat objects. To be precise, we should talk about scenes consisting of β-fat objects for some constant β. However, we will allow ourselves, as is often done in papers about fatness, the ability to talk about scenes of fat objects meaning that the scene consists of β-fat objects for some not-too-small constant β.

It is noted that the complexity analysis of a number of problems has been improved by assuming the input is a fat scene. Some problems that have been looked at are the Binary Space partition problem and the Motion Planning Problem, the first of which we discuss further in section 3.5 of chapter 3. We refer the interested reader to Vleugels' thesis [20] and to van der Stappen's thesis [17] for a discussion of these and some other applications.

A scene of fat objects possesses the property that each object in the scene does not have too many neighbours of similar or larger size. To see why this is the case, consider some arbitrary point in the scene. It must be shown that there cannot be too many objects close to the point. Van der Stappen, in his thesis [17], proves this by showing that each such object takes up a constant fraction of the space near the point. He does this by considering arbitrarily placed hyperdiscs that are no larger than the diameter of the smallest object in the scene. He uses a simple packing argument to show that the maximum number of objects that can intersect the hyperdisc is bounded by the number of objects that can fit in a small ring

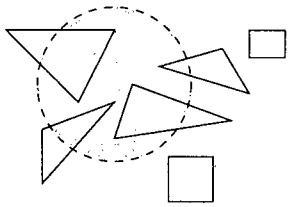


Figure 2.1: A disc covering part of a scene.

wrapped around the hyperdisc. The following theorem results:

Theorem 1 (van der Stappen [17]) *Let $\mathcal{S}$ be a scene of non-intersecting β-fat objects in $\mathbf{E}^d$ where the size of the minimal enclosing hyperdisc is at least σ. For a given constant $c \geq 0$, the number of objects intersecting any region with minimal enclosing hyperdisc size at most $c\sigma$ is bounded by $(c+1)^d/\beta$.*

From theorem 1, it follows that a fat scene (consisting only of fat objects) has a minimum object density across the whole scene. In fact, it is easy to see that by focusing on a subset of the objects in a scene that have radius larger than any constant, ρ, will also possess a similar minimum density of the objects. This can be seen by applying theorem 1 to the subset of the scene with the objects smaller than ρ removed. This observation leads us to another model of scene complexity - the *density* model. The density model considers all locations and sizes of a disc window looking at the scene and measures the maximum number of objects of size larger than or equal to the disc that can be intersected by disc window (see figure 2.1).

The formal definition for a scene $\mathcal{S}$ to have low-density is given as follows:

Definition 2 (Van der Stappen [17]) *Let $\mathcal{S}$ be a d-dimensional scene, and $\lambda \geq 0$ a parameter. We call $\mathcal{S}$ a* λ-low-density scene *if for any hyperdisc B, the number of objects $\mathcal{O}_i \in \mathcal{S}$ with $\rho_{meb}(\mathcal{O}) \geq radius(B)$ 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.*

Informally, we will say that a scene has *low-density* if it is a λ-low-density scene and λ is a small constant.

We will see, formally, in the next subsection that this model is more general than the fatness model; by this we mean that a scene consisting of fat objects will have low density, and that there are low density scenes which consist of non-fat objects. The main way in which this model is more general is that it now permits objects in the scene to have long and skinny parts. Because of the fact that many applications of fatness only use the density limiting property of fatness, many of these same applications will also run well on low density scenes. In Vleugels' thesis [20], for instance, he shows that the point-location and range-searching result of van der Stappen [17] can be generalized to apply to a low density scene. As a result, many of the applications of fatness are also applications of low density; these applications run well on a larger class of input scenes than previously known.

2.1.1 A Scene of Fat Objects has Low Density

We now state the results that prove that a low density scene is more general than a scene of fat objects. We first show that fatness is a sufficient condition for low density, and then we describe a scene that has low density but consists of 0-fat objects.

Theorem 2 (van der Stappen [17]) *Any d-dimensional scene consisting of* β*-fat objects has density at most* $\frac{2^d}{\beta}$.

The proof of this looks at some disc window overlooking a scene of fat objects. Sufficiently large objects that intersect the disc window are isolated. A constant

lower bound is then determined for the area of intersection between an object that intersects the disc window and the neighbourhood of the disc window (which is a larger disc with the same centre). Because of this constant lower bound on area of intersection, there are only a constant number of such objects that can pack into the neighbourhood of the disc window. This constant number is shown to be at most $\frac{2^d}{\beta}$ in van der Stappen's proof.

Figure 1.2 from the introductory chapter shows an example (on the right) of a scene that has low density but consists of line segments which are 0-fat. In general, any scene of β-fat objects can be converted into a scene with density at most $\frac{2^d}{\beta}$ by replacing every fat object with a line segment that is contained in the interior of the object. The fatness of a scene created in this way is 0, yet, the density of the scene will not increase by reducing its constituent objects to line segments. This proves the following theorem.

Theorem 3 *There is family of low density scenes such that each scene consists of 0-fat objects.*

2.2 Approximation by Sets of Points

The previous two models have looked at the objects themselves, looking either at the shape of the objects or measuring the number of objects of a certain size intersecting a given region of the scene. If the objects are too skinny or there are too many large objects intersecting the given region, the scene is deemed complicated. The next two models look at a seemingly different property of the scene: how easily can a scene be approximated by a set of points. Easily here refers to the number of points that are needed to approximate the scene. If a linear number of points are enough, then

the scene is considered simple, otherwise it is considered complicated. An *empty covering shape* is a shape (square, circle, etc.) that intersects the scene without containing any approximating points in its interior. Using squares, for instance, a set of points provides a good approximation to the scene if wherever an empty square is placed over the scene, it does not intersect more than a constant number of objects. The first model discussed, the *guarding set model*, makes the above idea precise. The second model, the *clutteredness model*, is actually a special case of the guarding set model, which focuses its attention on whether the bounding box corners of objects form a small guarding set against squares.

2.2.1 Guarding Sets

This notion was described by de Berg et al. in [10]. The points used to approximate the scene are called *guards* in this model, as they guard against covering shapes. The covering shapes are called *ranges*. In the model, a set of guards are essentially scattered throughout the scene. A good approximation to the scene is provided by the guards if each range that intersects more than a constant number of objects from the scene also contains at least one guard.

The precise definition of a κ-guarding set is given below:

Definition 3 (De Berg et al. [10]) *Let $\mathcal{S}$ be a d-dimensional scene. Let $\mathcal{R}$ be a family of subsets of $\mathbf{R}^d$ called* ranges. *Let κ be a positive integer. A set G of points is called a* κ-guarding set *for $\mathcal{S}$ against $\mathcal{R}$ if any range from $\mathcal{R}$ not containing a point from G intersects at most κ objects from $\mathcal{S}$.*

Ranges that do not contain any guards will be referred to as *empty ranges*. A *crowded range* is one that intersects more than κ objects. Thus, if we have a

sufficient set of guards, any crowded range can not be empty as an empty range would intersect no more than κ objects from the scene. In this model, we will say that a scene admits a small guarding set against a particular family of ranges if there is a κ-guarding set G of linear size (in the number of objects).

The model allows for different ranges to be used, which can have the effect of changing the effectiveness of the approximations to the scene. The ranges can be anything from cubes to discs to fat objects to line segments. However, de Berg et al. in [10] have argued that approximating against non fat ranges does not adequately measure the complexity of a scene. In this spirit we only consider families of ranges that are fat. De Berg et al. have also shown that if a scene admits a linear size guarding set against any family of fat objects then it will also admit a linear size guarding set against the family of hypercubes. It is therefore sufficient to prove properties about guarding sets against hypercubes, as these result will automatically apply to guarding sets against any fat shape. As a result, hypercubes will be our default family of ranges in the subsequent discussion.

Finally, we note that this model is strictly more general than the density and the uncluttered ness models (to be discussed in the next subsection).

Application: Motion Planning Problem for Robots

One application of the guarding set model studies the computational complexity of the free space for the motion planning problem for robots. We introduced the motion planning problem in the introduction. The motion planning problem usually involves taking a scene of obstacles, a starting location and a final location and planning a path through the obstacles so that a robot (which will likely have a shape to it, but could also be a point) can travel from the starting to the final location if such motion

is possible. As part of the task of planning the motion, we need to have a model of the free space that the robot can maneuver in. The computational complexity of the free space represents how hard it is to compute a model of the free space. Once the free space has been computed, it must be determined if there is a path from the start to the finish in the free space. There are a number of algorithms for determining and computing such a path. It is beyond the scope of this thesis to present the details of these. We are concerned with the computational complexity of the free space, called the *free space complexity*, however, as this is a limiting factor in an exact solution to the motion planning problem. The results presented apply for a robot that has *bounded reach* with f degrees of freedom. A bounded reach robot is one where the maximum distance that any part of the robot can be from a fixed reference point on the robot is no larger than twice the size of the smallest object. That is, let $\mathcal{R}$ be a robot with f degrees of freedom moving in a two-dimensional scene $\mathcal{S}$ containing n obstacles. Let $p_{\mathcal{R}}$ be an arbitrary reference point inside $\mathcal{R}$. The *reach* of $\mathcal{R}$, denoted reach($\mathcal{R}$), is defined as the maximum over all configurations of $\mathcal{R}$ of the distance from $p_{\mathcal{R}}$ to any point on $\mathcal{R}$. A *bounded reach* robot $\mathcal{R}$ is defined such that the reach of $\mathcal{R}$ is bounded as $\text{reach}(\mathcal{R}) \leq c \cdot \min_{\mathcal{C} \in \mathcal{S}}\{\text{size}(\mathcal{C})\}$, where c is a (small) constant. If we assume that the scene of obstacles forms a low density scene then it has been shown that the free space complexity for this scene is $\Theta(n)$, regardless of the dimensionality of the scene [18]. In [10], de Berg et al. show that if a scene has a κ-guarding set against squares, then the free space complexity is $O(\kappa^f n^{\frac{f}{2}})$. This is an improvement of the worst case bound of $\Theta(n^f)$ for the free space complexity. For more details on the motion planning problem, we refer the reader to Vleugels' thesis [20].

2.2.2 Clutteredness

The clutteredness model is a special case of the guarding set model. This model looks at a scene to see if the bounding box corners of the objects in the scene form a linear-sized guarding set against squares. Why might one be interested in a more restrictive model? The reason is that it is hard to compute, for a scene, what the minimum number of guards needed is to provide a good approximation of the scene. In the clutteredness model, we just have to form one particular set of guards, and check to see if they provide a good approximation of the scene. This is quite straightforward to accomplish.

By definition, the clutteredness model is more restrictive than the guarding set model, and therefore considers more scenes to be complicated. It can be shown, however, that clutteredness is still more general that the density and fatness models.

The precise definition for a scene of objects to be κ-cluttered is as follows:

Definition 4 (De Berg [7]) *Let $\mathcal{S}$ be a d-dimensional scene. 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.*

Informally, we will say that a scene is *uncluttered* if the clutter factor for the scene is a small constant.

Applications: Motion Planning and Linear Sized Binary Space Partitions

The motion planning problem has been analyzed with respect to the clutteredness model. In [10], de Berg et al. give a lower bound for the free space complexity of a κ-cluttered scene with n objects where the robot has f degrees of freedom.

The lower bound is $\Omega(\kappa^f n^{\frac{f}{2}})$. Since this model is less general than the guarding set model, the upper bound on the free space complexity for a scene with a small guarding set applies here. We get that the free space complexity for a bounded-reach robot is $\Theta(\kappa^f n^{\frac{f}{2}})$, for both uncluttered scenes and scenes with small guarding sets. This result shows that although the unclutteredness model is more general than the density model (as will be seen in theorems 4 and 5), the generality does not always come without a cost in terms of the efficiency of solving problems with respect to the model.

Another problem that we look at in more detail in section 3.5 is the binary space partition problem. The binary space partition problem takes a scene of disjoint objects as input and recursively produces a set of dividing hyperplanes so that each region induced by the hyperplanes will end up containing no more than one object from the scene. See figure 2.2 for a picture of a binary space partition. A useful property of a binary space partition is its size, or the number of regions it partitions the space into. Paterson and Yao [15] have shown that in the plane, a binary space partition of size $O(n \log n)$ always exists for a set of n polygons. In three-dimensions, they have given an example of a scene that requires a size $\Omega(n^2)$ binary space partition. The sizes of BSP's for scenes of fat objects, low density scenes, uncluttered scenes and scenes with linear-sized guarding sets have been studied. In every case, a BSP of linear-size is guaranteed [6, 7]. For this problem, it is interesting to see if even more general models can be related to the size of a BSP to get an even better understanding about the types of scenes that lead to small BSP's; we return to this problem in section 3.5.

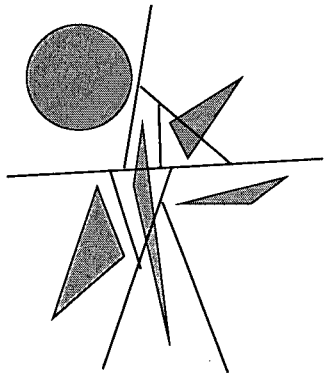


Figure 2.2: A binary space partition of a scene.

2.2.3 A Scene with Low Density is Uncluttered

We now show that the clutteredness model is more general than the density model (for appropriately chosen constants). We first show that a low density scene has a constant clutter factor. Then we describe a scene that is uncluttered but has high density.

Theorem 4 (De Berg [7]) *Any d-dimensional λ-low-density scene has clutter factor at most $\lfloor\sqrt{d}\rfloor^d\lambda$.*

Proof: (Summary of proof by de Berg) Let $\mathcal{S}$ a λ-low-density scene of d-dimensions. We know that any disc window of radius r will intersect no more than λ objects with radius greater than or equal to r. Now looking at the bounding box corners of objects in the scene, we want to determine the largest number of objects that can intersect a hypercube without leaving a bounding box vertex in the hypercube. Let $\mathcal{O}$ be a hypercube such that no objects leave bounding box corners inside. For an object to intersect $\mathcal{O}$ the side-length of its bounding box must be at least l, where l is the side-length of $\mathcal{O}$. This tells us that such an object must have radius at least

$\frac{l}{2}$. How many discs of radius $\frac{l}{2}$ are needed to cover $\mathcal{O}$? To determine this, we cover $\mathcal{O}$ with a grid of smaller hypercubes such that the diameter of each sub-cube is l. If $\mathcal{O}'$ is one of the sub-cubes, and $\delta(\mathcal{O}')$ represents its diameter, then its side length, $l(\mathcal{O}') = \frac{\delta(\mathcal{O}')}{\sqrt{d}} = \frac{l}{\sqrt{d}}$. Thus to cover $\mathcal{O}$, we need $\lfloor\sqrt{d}\rfloor^d$ sub-cubes. Now each of these cubes can be wrapped in its minimally enclosing disc, which by the λ-low-density of the scene must not intersect more than λ objects larger than or equal to $\frac{l}{2}$ in radius. We also know that there are no objects with radius smaller than $\frac{l}{2}$ since $\mathcal{O}$ was chosen not to contain any bounding box corners. $\mathcal{O}$ therefore does not intersect more than $\lfloor\sqrt{d}\rfloor^d\lambda$ objects in total. This proves that $\mathcal{S}$ is a $\lfloor\sqrt{d}\rfloor^d\lambda$-uncluttered scene. □

We now give an example of a scene that has high density but is uncluttered. To construct such a scene, we start with a scene that has high density because one region in the scene is intersected by a large number of objects. Now it may be that these objects would leave enough bounding box corners to make the region uncluttered already. But we will assume many of the objects pass entirely through the region without leaving a bounding box corner inside. We can modify this scene to one that is uncluttered by adding extra line segments. These line segments will have the effect of projecting bounding box corners into the dense region to prevent large boxes from intersecting the region (recall that boxes cannot contain bounding box corners in the unclutteredness model). See figure 2.3 for an example showing how line segments project bounding box corners into the dense region. In order that the added line segments do not themselves form a cluttered region, we must ensure that there is sufficient spacing between adjacent line segments in the periphery of the scene; to accomplish this, each new line segment should be drawn outside of the bounding box of the previous line segment; in this way, no square can intersect

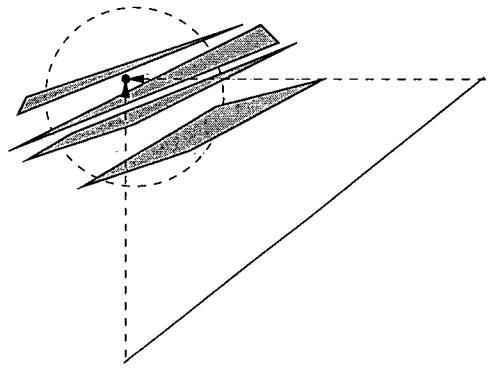


Figure 2.3: A dense scene with an added line segment that projects a bounding box corner inside the dense region.

more than one of the added line segments without also intersecting a bounding box corner. The number of line segments we have to add is proportional to the radius of the disc region that contains the long objects, the spacing between the objects in the region and the constant factor we want to achieve for the clutter factor; this is roughly proportional to $\frac{r}{\phi\kappa}$ where r is the radius of the disc, ϕ is the smallest distance between two adjacent objects and κ is the desired (small) clutter factor. Looking at the density of the modified scene, we see that it has high density because the line segments added are ignored by the disc window. Also, the scene has become κ-cluttered, and is therefore uncluttered. This proves that we can create a family of such scenes starting with any high density scene, as the following result indicates:

Theorem 5 (De Berg et al. [9]) *There is a family of $\Omega(n)$-dense scenes such that each scene is κ-cluttered for a small constant κ. That is, each scene in the family has high density, but is uncluttered.*

2.2.4 On to Linear-Sized Guarding Sets

We now look at how guarding sets fit in with other models. By definition, the guarding set model is more general than the clutteredness model. If a scene $\mathcal{S}$ of dimension d is uncluttered with clutter factor κ then the bounding box vertices of the objects will form a κ-guarding set of size $2^d n$. This implies that the model is more general than the density or fatness models.

It is a little more difficult to come up with a scene that has a high clutter factor and yet has a linear-sized guarding set. The difference would be in the location of the associated guards for the two models. Because the next model we look at turns out to be the same as the guarding set model (to within a constant factor), we save the presentation of this scene until the next section.

2.3 Simple Cover Complexity

We now come to another type of complexity measure. The model of simple cover complexity is a *cover complexity measure.* Such a measure consists of two parts. First the scene is covered by a number of shapes (cubes, discs, etc.), where the size of each shape is somehow restricted. A scene is *covered* by a set of shapes if the union of the shapes contains the bounding box of the scene. Second the complexity, or amount of interaction between objects inside each of the covering shapes is measured and added up. This total becomes the "cover complexity" of the scene. One cover complexity measure differs from another one by the shape chosen to cover the scene, the method used to restrict the size of the covering shapes and by the method used to determine the complexity associated with the shape. This model attempts to measure the amount of interaction in the scene. The inherent idea is different than

trying to approximate a scene by a set of points, although they are related. The rest of the models discussed in this thesis are cover complexity measures.

In the *simple-cover-complexity* model, discs are used to cover the scene. Let ϵ be a positive small constant. To restrict the size of a disc of radius r, the ϵ-radial expansion of the disc (i.e. the disc with radius $(1+\epsilon)r$ with the same centre as the disc) must not intersect more than a constant number of object facets from the scene. An object facet is a $(d-1)$-dimensional face on the object; in 2-dimensions, for instance, a facet of a polygon would be one of the line segments on its boundary. A disc that is restricted in this way will be called a *simple disc*. Each disc in the simple cover is assigned a value of 1 for its complexity. This parameter indicates that a constant amount of "complexity" is contained inside the disc; in other words, there is only a constant amount of interaction between the objects inside any disc in the cover. Thus the simple cover complexity of the scene is a count of the number of simple discs needed to cover the scene.

Our definition for a scene $\mathcal{S}$ of disjoint objects to have (σ, δ)-simple-cover complexity is given below. In the definition, the *ϵ-expansion* of a hyperdisc with radius r is the same-centered hyperdisc of radius $(1+\epsilon)r$.

Definition 5 *Let $\mathcal{S}$ be a d-dimensional scene, $\epsilon > 0$ a small constant and δ a parameter. A* δ-simple hyperdisc *is a hyperdisc whose ϵ-expansion intersects at most δ object facets in $\mathcal{S}$. A* δ-simple cover for $\mathcal{S}$ *is a collection of δ-simple hyperdiscs whose union covers $bb(\mathcal{S})$. 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 (note this comes from the arrangement of covering hyperdiscs with the lowest cardinality).*

Informally, we will say that a scene has *small simple-cover-complexity* if there

are small constants δ and σ such that it has (σ, δ)-simple-cover complexity.

Originally the simple-cover-complexity model appeared in a paper by Mitchell et al. [14]. There the simple-cover-complexity was defined in terms of a region of space that needed to be covered. This region could be the bounding box of the scene, a line segment within the scene, or any other compact region. The complexity of a line segment relates to how complicated the scene is near the line segment, that is, how many object facets are there close to the line segment. Taking the complexity of a bounding box, however, gives an idea of what the average complexity is over the whole scene. Mitchell et al. defined the simple-cover-complexity in terms of ϵ-simple hyperdiscs as we do. When de Berg et al. in [9] talked about simple-cover-complexity, they dropped the idea of using ϵ-expansions. To justify this, they note that the ratio between the simple-cover-complexity of a scene computed for two different values of ϵ is proportional to the ratio between the two values of ϵ. They also note that dropping ϵ-expansions from the definition does not significantly change the measurement of the simple-cover-complexity of most scenes in practice (although it can cause very large changes in theory). De Berg et al. also focused their discussion of simple-cover-complexity as a complexity measure of the whole scene. Our definition of simple-cover-complexity returns to using ϵ-expansions, but keeps de Berg et al.'s notion of simple-cover-complexity looking at the whole scene.

2.3.1 Relationships to Previous Models

We show in this section that the simple-cover-complexity model is more general than the clutteredness model and that the model is identical to the guarding set model (to within a constant factor). That is, a scene in the plane has a linear sized guarding set if and only if it has small simple-cover-complexity.

Theorem 6 (De Berg et al. [9]) *Any κ-cluttered scene has (σ, δ)-simple-cover complexity where $\sigma = O(d2^{5d+3})$ and $\delta = O(6^d\kappa)$.*

Proof: We give the outline the proof, and refer the interested reader to de Berg et al. [9] for the details.

Suppose $\mathcal{S}$ is a κ-cluttered d-dimensional scene. We will also assume that κ bounds the number of object facets passing through a cell, not just the number of objects. The basic idea of the proof is to first show that $\mathcal{S}$ can be covered by a set of hypercubes that each do not intersect any of the bounding box corners of the objects in $\mathcal{S}$. This cover must be of the nature that each hypercube in the cover is not adjacent to more than a constant number of other hypercubes in the cover. Once such a cover has been constructed, a simple-cover can be constructed by wrapping each hypercube in the cover with the minimally enclosing hyperdisc.

De Berg et al. [9] have described how to cover the $\mathcal{S}$ with $O(d2^{5d+3})$ hypercubes with the property that each cube in the cover will be adjacent to at most $O(6^d)$ other hypercubes in the cover. Now since $\mathcal{S}$ is κ-cluttered, each cell in the cover intersects no more than κ object facets from $\mathcal{S}$. Any minimally enclosing hyperdisc wrapped around a hypercube in the cover will intersect at most $O(6^d)$ other cubes, and thus at most $O(6^d\kappa)$ other objects from $\mathcal{S}$. From this it is concluded that $\mathcal{S}$ has (σ, δ)-simple-cover complexity for $\sigma = O(d2^{5d+3})$ and $\delta = O(6^d\kappa)$. □

Now we look at the converse to the above relationship between simple-cover-complexity and clutteredness. We note that there are scenes that have small simple-cover-complexity, yet are cluttered. One family of such a scene consists of a section of n parallel lines of length l with separation between leftmost and rightmost lines δ. The simple-cover-complexity of the lines themselves is at least $\frac{ln}{\delta\lambda}$ as we shall see in theorem 9 from the next chapter (where δ is the maximum number of object facets

permitted inside a cell and λ is the distance between the leftmost and rightmost lines). We can add objects to this to give the resultant scene small simple-cover-complexity; this is done by adding an equivalent number of point objects (to the actual simple-cover-complexity of the parallel lines) just outside of the bounding box of the parallel lines. This would have the effect of averaging out the simple cover complexity of the scene over the now large number of objects, at least $n + \frac{ln}{\delta\lambda}$, in the scene so that the resultant simple-cover-complexity would be a small constant. This shows that the scene has small simple-cover-complexity. Yet, the scene is cluttered on account of the parallel lines (assuming the clutter factor is less than n). A square can be placed to intersect all of the parallel lines without containing an endpoint of the scene.

Theorem 7 (De Berg et al. [9]) *There is a family of cluttered scenes which have small simple-cover-complexity and linear sized guarding sets against hypercubes.*

It is also interesting to look at the relation between linear size guarding sets and simple cover complexity. De Berg et al. have shown [10] that if a scene admits a linear-sized guarding set then it has small simple cover complexity. In the same paper, they also show that a planar scene with small simple cover complexity has a linear size guarding set. The exact theorem proved is shown below:

Theorem 8 *Let $\mathcal{S}$ be a d-dimensional scene consisting of n objects.*

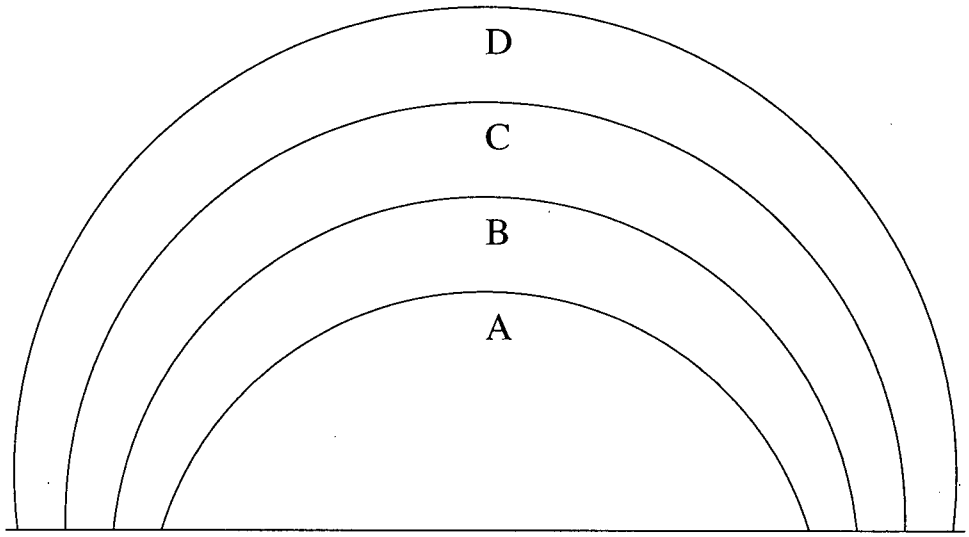
(i) If $\mathcal{S}$ has a κ-guarding set of size m against hypercubes, then it has (s, δ)-simple-cover complexity for $s = 2^{5d+3}d(m/n)$ and $\delta = 6^d\kappa$.

(ii) For $d = 2$, if $\mathcal{S}$ has (s, δ)-simple-cover complexity, then it has a 8δ-guarding set against squares of size $O(sn)$.

If we have an algorithm that is not sensitive to small regions consisting of parallel lines in the scene, then the simple-cover-complexity does a better job at estimating the complexity of this configuration when compared to the clutteredness model. The clutteredness model may be more appropriate for algorithms that require a guarantee that the whole scene is of a uniform complexity. But the above has demonstrated that the simple-cover-complexity model is a more general model in the sense that it considers more scenes to be simple. Finally, we note that since the guarding set model is equivalent to the simple-cover-complexity model in the plane, that the above example shows that the guarding set model is also strictly more general than the clutteredness model.

2.4 Picture of Relationships Between Models

Below is a diagram representing the classes of scenes that each model considers to have low complexity (with appropriate constants set for each model so that the models can all be compared by theorems from this chapter). Note how the models form a nice hierarchy. This is an interesting feature of the models that will be extended in the so-called weighted complexity models discussed in the next chapter. Once again we reiterate that the usefulness of the models are in how well they can be used to predict the behaviour of algorithms when input simple scenes. A more general model is only useful if there are algorithms whose behaviour is predicted accurately by the model.



A - Scenes of Fat Objects
B - Low Density Scenes
C - Uncluttered Scenes
D - Scenes with Small Simple-Cover-Complexity or
a Linear-sized Guarding Set

Figure 2.4: Hierarchy of Models

Chapter 3

Weighted Complexity Models

The goal of this chapter is to describe some new models of scene complexity that may be better predictors of the time or space complexity of certain types of algorithms. Before describing what we mean by a weighted complexity model, we will explain where the models of the previous chapter fall short in their goal as predictors for these algorithms. Certainly they do a fine job of predicting the behaviour of many algorithms. However, the ultimate goal is to have a model that classifies scenes as uncomplicated whenever a particular algorithm actually finds the scene to be simple to deal with. The problem is that we want to be able to determine this for a scene in less time than it takes to run the algorithm itself on the scene. Thus the models attempt to efficiently measure certain properties of scenes that make certain algorithms behave better. The more scenes that can be deemed simple with respect to a particular algorithm the better. It is in this light that we present new models.

Suppose you are processing a scene that is for the most part sparse (say it has low density over most of the area of the scene), yet it contains one small region consisting of a large number of relatively long and thin objects each passing through

a common area (see figure 3.1). Any of the models discussed up to this point would consider this scene to have high complexity, and would therefore put high bounds on the complexity estimates for dealing with this scene. There are algorithms that do not find such a scene to complicated to deal with, however, like the solution to the binary space partition problem presented in section 3.5. The previous models in the hierarchy up to and including the clutteredness model deem a scene to be complicated if it contains one complicated region. Such a region could consist of non-fat objects, a set of large objects that are closely packed in a region or a set of objects that pass through a region without leaving a bounding box corner inside. These models look for a certain uniformity of complexity throughout the scene to deem a scene simple. A small complicated portion of the input scene will deem a scene complicated. As such, these models do a nice job at predicting the behaviour of algorithms whose worst case behaviour is determined by the complexity inherent in only a small portion of the input scene. If a nonuniform scene, like the one in figure 3.1, is presented to an algorithm that does not require uniformity throughout its input, like the binary space partition problem, then these models will overestimate the complexity of the scene. That is, the models will put an overly large upper bound on the time or space complexity of any such algorithm with respect to this scene.

The simple-cover-complexity and guarding set models will not necessarily deem a scene complicated merely because the scene contains a complicated region. They focus on the scene as a whole when assigning it a complexity. However, if you consider the relatively simple scene, $\mathcal{S}$, consisting of n parallel lines each of length l with the separation between leftmost and rightmost lines being λ, then the models of simple-cover-complexity and guarding sets do not behave so well. These

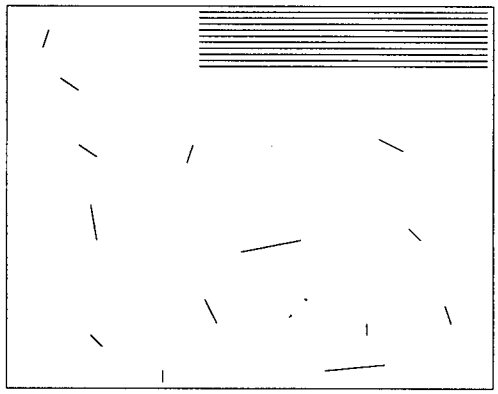


Figure 3.1: A sparse scene with a dense region in the corner.

models could classify a scene containing a region of clutter as simple if there are a large number of additional extraneous objects (outside the region of clutter). A seemingly simple scene, like $\mathcal{S}$, consisting of parallel lines gets a high complexity, because there are not a large number of extraneous objects to average down the complexity to linear levels in the number of objects. The following theorem shows the complexity assignments to $\mathcal{S}$ by all of the models from the last chapter.

Theorem 9 *Let $\mathcal{S}$ be a planar scene consisting of n identical vertical parallel lines each of length l that start and end in alignment. Let $\lambda < l$ be the distance between the leftmost and the rightmost lines.*

- *$\mathcal{S}$ consists of 0-fat objects.*
- *$\mathcal{S}$ is a n-density scene.*
- *$\mathcal{S}$ is a n-cluttered scene.*
- *$\mathcal{S}$ has $(\delta, \Omega(n^2))$-simple-cover-complexity.*
- *$\mathcal{S}$ has an $\Omega(n^2)$-sized κ-guarding set.*

Proof: Clearly, $\mathcal{S}$ consists of 0-fat objects.

The density of $\mathcal{S}$ is n since a circle with diameter larger than λ can intersect all n of the lines, each of which is larger than the size of the circle.

The clutter factor of $\mathcal{S}$ is n since a box with side length λ can be placed to cover all the lines without including any of the bounding box corners of the lines.

The average separation between the parallel lines is $\frac{\lambda}{n}$. Since no more than δ objects can intersect any disc, we will on average be able to use discs of radius $\frac{\delta\lambda}{2n}$ to cover δ lines. How many discs of this size would it take to cover a group of δ lines? It would be at least the number of discs to cover one of the lines, which is

$$\frac{l}{\frac{\delta\lambda}{n}} = \frac{ln}{\delta\lambda}$$

discs. Thus to cover a group of δ lines, we need at least a linear number of discs. To cover all $\lceil \frac{n}{\delta} \rceil$ groups of δ lines, requires at least

$$\frac{ln^2}{\delta^2\lambda}$$

discs. This proves that the δ-simple-cover-complexity of $\mathcal{S}$ is in $\Omega(n^2)$.

The size of a κ-guarding set for $\mathcal{S}$ against squares is superlinear in n. This follows from de Berg et al.'s result [10] from theorem 8 that in the plane the guarding set and simple-cover-complexity models are equivalent. However, we also give a direct argument as follows. Suppose $\mathcal{G}$ is a linear-sized κ-guarding set for $\mathcal{S}$ against squares. The spacing between adjacent lines is $\frac{\lambda}{n-1}$. Thus, imagine placing a rectangular grid of squares over the lines, where each square has side length $(\kappa)\frac{\lambda}{n-1}$, the left column of cells intersects the first column of $\kappa + 1$ lines and where there are

$$\left\lfloor \frac{\lambda(n-1)}{\kappa\lambda} \right\rfloor = \left\lfloor \frac{n-1}{\kappa} \right\rfloor$$

cells across the top, and

$$\left\lfloor \frac{l(n-1)}{\kappa\lambda} \right\rfloor$$

cells down the side. Each cell in this grid intersects $\kappa + 1$ lines, and therefore must contain at least one guard. This would imply that there are at least

$$\left\lfloor \frac{n-1}{\kappa} \right\rfloor \cdot \left\lfloor \frac{l(n-1)}{\kappa\lambda} \right\rfloor \in \Omega(n^2)$$

guards in $\mathcal{G}$. □

This theorem tells us that the models of the previous chapter all find parallel lines to have high complexity. With respect to some algorithms, this assertion is not desirable.

We have shown two shortcomings with the models of the last chapter. That is most are local measures of complexity, and they all consider the scene consisting of n parallel lines to have high complexity. We would like to create a model that does not have these two properties. In fact we would like to find a model that not only considers the exact scene of parallel lines described to have low complexity, but any part of a scene that consists of "almost" parallel lines to have low complexity. To attain this, we now explain what is meant by a weighted model of scene complexity.

In the last chapter, we introduced the notion of a cover complexity measure of a scene of n disjoint objects. Our weighted-complexity models are based on this notion. They are more forgiving, however, meaning that regions of clutter in the scene do not always cause the scene to have a high complexity. We explain how this is done by looking at how the simple-cover-complexity model is extended by these models.

In the simple-cover-complexity model, the bounding box of the scene is covered with a set of discs. A disc is considered too complicated if it intersects a lot of objects (more than whatever constant is being used) from the scene. The measure or indicator of the complexity of the scene is the minimum number of simple discs needed to cover the scene. Restricting the discs to contain a constant number of

intersections with objects forces the discs to be too small which, as a result, gives a structure consisting of a lot of long and thin objects a high complexity. To get around this, we will say that an object may intersect a disc without concern if it passes entirely through a disc. We call such intersections *silent intersections*. Such intersections have the effect of partitioning the region into smaller sub-regions. As a result we will charge for their presence by giving each covering region a weight that is equal to the one plus the number of silent intersections contained in that region.

Objects that do not pass through a region must end inside the region. Long and thin objects that pass through a region often do not present too much trouble for processing, however, objects that end in a region can cause problems. In order that they do not cause too much trouble, the number of objects permitted to terminate in a region is bound by a constant. In this way, we cover the scene with regions that individually do not contain too many objects that end inside, and charge a weight per region that is equal to the number of sub-regions each region is partitioned into by objects that intersect it silently. The complexity of the scene then becomes a count of a number of sub-regions that individually do not contain too many object endings. We make these notions more precise in the next section.

We again note that the above statements about complexity do not generalize to all algorithms, but they are only true for some algorithms that can handle some variation in the object density across the scene without increasing the complexity. One important example of such a problem is the planar binary space partition problem that we look at in section 3.5. In this section, we show that the size of a binary space partition for a scene is at worst proportional to its quadtree complexity.

3.1 Definitions

In this chapter, we deal only with planar scenes. As well, a *planar scene* is assumed to consists of a set of n line segments that do not intersect, except possibly at the endpoints. The *set of representatives* for a planar scene is the set of endpoints of the line segments in the scene.

A *weighted complexity model* is a cover complexity model where the weights of the cells are greater than or equal to one. To review, a cover complexity model looks at a scene of disjoint objects, and attempts to cover the bounding box of the scene by containing it in a union of simple *cells* or *regions* (different models specify exactly what makes a cell or region simple). The *weight of a region* is meant to capture the complexity contained inside the region, and the *complexity of the scene* is the sum of the weights of the constituent regions of the cover. Different complexity models can be constructed by using different shapes to make up the cover, by restricting the size of the shapes in different ways and by assigning weights in different ways.

In our new models, we look at using discs and adaptive quadtree decompositions of the scene to cover the bounding box of the scene. To restrict the size of the regions in the covers, we limit the number of object representatives in any given region. A *silent intersection* with a region in a cover is an intersection between a line segment from the scene and the region such that the line segment intersects, but its associated representatives do not. The weight given to each region is the number of sub-regions that are contained in the region, which is one plus the number of silent intersections with the region (see figure 3.2 for an example showing the calculation of cell weights).

How does this approach improve upon the previous models, and simple-cover-complexity in particular, to provide a better measure of the interaction between

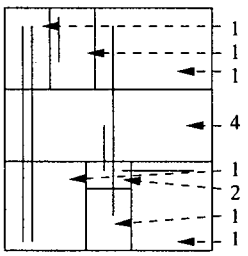


Figure 3.2: A scene covered with rectangles. The weights of each of the covering cells are shown on the right.

objects in the scene? Both models give a count of a number of simple regions. However, in our weighted model we have further relaxed the definition of what it means for a region to be simple. Looking at this more closely, we see that we are actually using the objects themselves to help in the creation of our partition of the space. We first cover the scene with discs or quadtree cells, and then use the objects to partition each of the covering cells into a number of simple sub-regions. This is what we are counting when we count the number of objects that intersect the cell silently. Inside each of the resulting regions, the amount of interaction between objects is bounded. This happens by limiting the number of objects that terminate inside a region. Objects that pass right through a region interact in a very restricted way. In partitioning the scene into a set of interaction limited regions, we are assuming that interaction among objects is important for predicting algorithmic behaviour. This assumption is valid for some applications, like binary space partitions. If a valid cover of the scene partitions it into a linear (in the number of line segments) number of interaction limited regions, then the scene is not inherently as complicated as a cover which partitions it into a superlinear number of interaction limited regions.

We therefore would like to look at different covering shapes to see how pre-

cisely each model captures the amount of interaction between objects. One way of doing this is by looking at how the models deal with a scene of parallel lines. We are assuming that this structure is simple to deal with by some algorithms, and as such want to ensure that our models agree with this assumption. The amount of complexity assigned by different models to this scene will be one determinant of the goodness of a model.

The second criteria of goodness for a model relates to how easy it is to compute what the complexity of the scene is. It is not useful to have a model of complexity that takes longer to compute than the problem you are trying to analyze. Otherwise, you may as well run the algorithm itself on the various inputs to classify inputs as being complicated or not.

We give two disc based models (expanded and unexpanded disc complexity) that do not quite meet all of our criteria. Neither are easy to compute, and only one assigns a reasonably low complexity to a scene consisting of parallel lines. We then give an adaptive quadtree based model that does meet our criteria. For each model, we relate it to previous models and give some other results related to the model. We show that the new models extend the hierarchy of models formed by the models of the last chapter: expanded disc complexity generalizes simple-cover-complexity, and both the quadtree and unexpanded disc complexity models generalize expanded disc complexity. We have not been able to determine the exact relationship between unexpanded disc complexity and quadtree complexity. After the presentation of the models, we look at the complexity assignment by the models to a scene consisting of a series of n parallel lines. As a result, we are able to relate the various models with respect to our first criteria of goodness. Our second criteria is looked at in the discussion of each model, where we look at the time complexity of computing

appropriate covers for each of the models.

3.2 Disc Based Models

Before describing the models, we define some terms and concepts relating to the models. We will talk about two types of discs: unexpanded and expanded discs. An unexpanded disc is a closed disc, and we will usually refer to it simply as a *disc.* An expanded disc consists of an unexpanded disc, called the *base disc* with an open ring around it that has thickness ϵr. Thus, overall, the radius of an expanded disc is $(1+\epsilon)r$, where r is the radius of the corresponding base disc. The ϵ in the definition of an expanded disc is assumed to be a positive fixed global constant. Note that an unexpanded disc has a corresponding expanded disc; we call this corresponding expanded disc the *ϵ-expansion* of the unexpanded disc.

In the following, we assume that δ is a parameter. A *δ-weighted-simple disc* is a disc with radius r whose ϵ-expansion contains no more than δ representatives from the scene. A *silent intersection* with a disc is an intersection of a line segment from the scene with a disc such that the line segment does not begin or terminate in the interior of the disc. The *weight* of a δ-weighted-simple disc is defined to be one plus the number of silent intersections with that disc. A *δ-weighted-simple disc-cover* for a scene is a collection of δ-weighted-simple discs whose union covers the bounding box of the scene. The *weight* of a δ-weighted-simple disc-cover, D, is the sum of the weights of the individual discs in the cover, i.e.

$$w(D) = \sum_{d \in D} w(c)$$

3.2.1 Expanded Disc Complexity

The expanded disc complexity model is a weighted complexity model where the cover consists of discs; it is a δ-weighted-simple disc-cover. In a δ-weighted-simple disc-cover, the size of the discs in the cover are bounded by the number of line segment endpoints that occur in the ϵ-expansion of the discs. For the purposes of assigning a weight to the disc, only the silent intersections with the base disc count. Limiting interaction (by limiting the number of endpoints) in the expansion of a disc, rather than just in the base disc itself serves to make the model more robust against small perturbations in the input scene (meaning that small changes in the location or shape of objects will not cause large changes in the measured complexity). The scene shown later in figure 3.6 is an example of a scene that would not be robust against small perturbations in the scene if only base discs were used to limit interaction.

The definition of the expanded disc complexity of a scene is now given below:

Definition 6 *Let $\mathcal{S}$ be a planar scene. Let δ be a parameter.*

- *We say that $\mathcal{S}$ has (σ, δ)-expanded-disc complexity if there is a δ-weighted-simple cover $\mathcal{C}$ of $\mathcal{S}$ such that the weight of $\mathcal{C}$ is at most σ.*
- *The* δ-expanded-disc complexity *of $\mathcal{S}$ is the* smallest *σ for which $\mathcal{S}$ has (σ, δ)-expanded-disc complexity.*

Informally, we talk about a scene having *low expanded disc complexity* if the δ-expanded-disc complexity of the scene is cn for a fixed small constant $c > 0$.

3.2.2 Relationships

We now show the relationship between expanded disc complexity and simple-cover-complexity. Because both simple-cover-complexity and expanded disc complexity focus on limiting interaction between objects in the ϵ-expansions of the discs, it is easy to relate simple-cover-complexity to expanded disc complexity.

Theorem 10 *If a scene of n objects $\mathcal{S}$ has (σ, δ)-simple-cover-complexity then it has $(\sigma n(1+\delta), 2\delta)$-expanded disc complexity.*

Proof: Suppose $\mathcal{S}$ is a planar scene of n disjoint line segments that has (σ, δ)-simple-cover-complexity. Let $\mathcal{C}$ be a simple-cover that has no more than σn δ-simple-discs. Consider an arbitrary disc d in $\mathcal{C}$. Certainly, d is 2δ-weighted-simple because its expansion does not intersect more than δ line segments from $\mathcal{S}$. The worst possible scenario for d is where each of the segments intersecting d is totally contained in its interior; this would leave 2δ representatives in d, proving that d is at most 2δ-weighted-simple. The weight of d is the number of silent intersections with d. If the maximum possible, δ, intersections with d all turned out to be silent, then the weight of d is at most $(1+\delta)$.

From this we conclude that every disc in $\mathcal{C}$ is 2δ-weighted-simple and has weight no more than $(1+\delta)$. This implies that $\mathcal{C}$ is a $(1+\delta)$-weighted-simple cover. Since there are no more than σn discs in $\mathcal{C}$, it follows that the weight of $\mathcal{C}$ is bounded by $\sigma n(1+\delta)$. $\mathcal{S}$ therefore has $(\sigma n(1+\delta), 2\delta)$-expanded disc complexity. □

This result shows us that every scene that has small simple-cover-complexity also has small expanded-disc complexity (for appropriate constants). It is not the case, however, that expanded disc complexity measures the same thing as simple-cover-complexity. It is relatively simple to construct a family of scenes where the

simple-cover-complexity is more than a constant factor higher than the expanded disc complexity. One simple example is a set of n parallel lines, where the spacing between adjacent lines is the constant of expansion, ϵ, and the length of the lines, l, is larger than or equal to $n\epsilon\delta$, where δ is the parameter limiting the number of intersections in a simple-disc. In this case, the total separation between the lines is $\lambda = n\epsilon$. From theorem 9, we know that we will not be able to cover δ of the lines with less than $\frac{ln}{\delta\lambda}$ simple-discs. In our case, we need at least

$$\frac{n\epsilon\delta n}{\delta n\epsilon} = n$$

simple-discs. Since there are $\lceil \frac{n}{\delta} \rceil$ groups of at least δ lines, we need at least

$$n \cdot \left\lceil \frac{n}{\delta} \right\rceil \geq \left(\frac{n^2}{\delta} \right)$$

simple-discs altogether to cover the parallel lines. Thus the δ-simple-cover-complexity of the scene is at least $\left(\frac{n^2}{\delta}\right)$. As for the expanded disc complexity of this scene, it is quite simple to use $2n + 1$ weighted-simple discs to cover the scene. We can do this because the spacing between the lines is ϵ. You take one disc with radius $\frac{l}{2(1+\epsilon)}$ that intersects all the lines and just misses the endpoints. The remaining endpoints can each be covered by a single small disc each. The resulting δ-expanded-disc complexity of the scene is thus $3n + 1$. We have thus shown that the expanded disc complexity is more than any constant times lower than the simple-cover-complexity of the described scene.

Theorem 11 *There is a family of planar scenes such that for each scene, $\mathcal{S}$, in the family, its δ-simple-cover-complexity divided by its δ-expanded-disc complexity is $\Omega(n)$, where n is the number of objects in $\mathcal{S}$.*

3.2.3 Unexpanded Disc Complexity

We will now look briefly at what we call *unexpanded disc complexity.* The difference between this complexity and expanded disc complexity is that we only concern ourselves with the number of silent intersections with the disc itself. That is, unexpanded disc complexity is expanded disc complexity with ϵ set to 0. Because the buffers created by the expanded discs are removed in this model, it is now possible for small perturbations in the scene to cause a large increase in the complexity of the scene. This being said, an unexpanded disc cover does not magnify the amount of measured interaction in a scene quite as much (that is, it requires fewer discs to cover regions of the scene consisting of almost parallel lines) as expanded disc complexity does, and in this way, better captures the inherent complexity of the scene.

The precise definition is given as follows:

Definition 7 *Let $\mathcal{S}$ be a planar scene. Let δ be a parameter.*

- *We say that $\mathcal{S}$ has (σ, δ)-unexpanded-disc complexity if $\mathcal{S}$ has (σ, δ)-expanded-disc complexity with an expansion factor $\epsilon = 0$.*
- *The δ-unexpanded-disc complexity of $\mathcal{S}$ is the smallest σ for which $\mathcal{S}$ has (σ, δ)-unexpanded-disc complexity.*

Informally, we talk about a scene having *low disc complexity* if the δ-unexpanded-disc complexity of the scene is cn for a fixed small constant $c > 0$.

3.2.4 Relationships

The unexpanded-disc complexity model is more general than the expanded-disc complexity model. This comes from the fact that if you take an expanded-disc cover

$\mathcal{C}$ of a scene, then $\mathcal{C}$ will have the same weight when considered as an unexpanded-disc cover. This observation proves the following result:

Theorem 12 *If $\mathcal{S}$ is a planar scene with $(\sigma n, \delta)$-expanded-disc complexity then it has $(\sigma n, \delta)$-unexpanded disc complexity.*

The unexpanded disc complexity model assigns a lower complexity to parallel lines than does the expanded disc complexity model. We will see this in theorem 20 in section 3.4. There, we will see that the expanded disc complexity of the scene can be made arbitrarily high by reducing the inter-line spacing and/or lengthening the lines. However, the unexpanded disc complexity of the scene is no more than $3n$. The expanded discs in an expanded cover cannot intersect too many endpoints in their ϵ-expansions. This forces the base discs to be too small to cover the parallel lines with a constant number of discs. The expanded disc model overestimates the interaction present in the scene since most of the covering discs do not actually contain any representatives, or non-silent intersections. The more regions in a cover that actually contain some representatives, the more accurately the cover measures the interaction. Regions that do not contain representatives can only contain objects that interact in a very limited manner.

3.2.5 Complexity of Computation

We do not know the exact computational complexity of computing either an expanded or an unexpanded disc cover of a scene. We believe that it is probably NP-hard to compute the minimum weight disc cover in either case. We do not have a direct proof of the NP-hardness of this, but note its similarity to the DISC-COVER problem. In the DISC-COVER problem you have a set of n points in the

plane and you want to cover them with a fixed set of discs (the sizes of the discs are fixed, the locations are not). For this problem, the discs in the set may be replicated as often as needed to cover the points, however, no disc may be used that is not the same radius as one in the set. This problem is NP-hard [11]. To compute an expanded or unexpanded disc cover, we cover a set of line segments in the plane with weighted discs, where the weight of the cover is to be minimized. It seems plausible that a reduction could be made from the DISC-COVER problem to our problem. We leave the exact determination of the computational complexity of computing weighted disc covers as an open problem.

Given that it seems to be hard to compute weighted disc complexities, why did we consider weighted disc covers at all? The reason is that they are closely related to the simple-cover-complexity model. We wanted to remove some of the shortcomings of this model, and a natural logical step was to generalize this model directly. Noting also that it seems hard to compute what the simple-cover-complexity is for a scene [20], we can look to see what problems analyzed with respect to the simple-cover-complexity do to get an approximation of the simple-cover-complexity. Looking at the ray-shooting problem, discussed in section 1.1.5, we note that in the algorithm of Mitchell et al. [14], a quadtree subdivision (although not exactly the same as the quadtree we compute in section 3.3) of the scene is first determined - not an actual disc cover. They then go on to show that the number of cells in the quadtree decomposition is a constant times the number of discs in the minimum disc cover of the scene, thus their algorithm operates a constant times more slowly than it would if it were input an actual disc based cover.

It is this sort of reasoning that leads us to believe that perhaps a decomposition of the scene based on quadtrees will have a complexity that is no worse than

a constant times the complexity of the expanded disc cover. We show in the next section that this is true, even if we compare the minimum disc complexity possible with the quadtree complexity, computed from the worst possible origin point for the quadtree. This guarantees that computing a quadtree decomposition of the scene from an arbitrary starting point will produce a decomposition of the scene with a complexity that is no more than a constant times worse than the complexity of the minimum expanded disc cover. This gives us a useful model that is more general than the expanded disc complexity model at a computation cost that is manageable.

3.3 Quadtree Complexity

Before we can define what we mean by the quadtree complexity of a scene, we must clarify what we mean by an endpoint sensitive adaptive quadtree decomposition of a scene. From now on we will use the term *quadtree decomposition* to refer to an endpoint sensitive adaptive quadtree decomposition. A quadtree decomposition of a scene is a spatial subdivision of the scene determined by the set of line segment endpoints. The quadtree algorithm presented below is similar to the one presented by Mitchell et al. in [14]. In Mitchell et al.'s decomposition, rectangles were used to cover the scene; subdivisions were made by dividing the rectangle in half by splitting parallel to the shorter side of the rectangle. Subdivisions were made whenever a cell in the decomposition intersected too many object facets. They provided a shrinking operation to ensure that the number of cells in the decomposition would be linear in the number of object facets in the scene. In our decomposition, we use squares to cover the scene, and divide cells into four nonoverlapping isomorphic subcells. A split in our decomposition is justified if there are too many line segment endpoints inside the cell, not if there are too many object facets.

The basic algorithm for a quadtree decomposition proceeds as follows. Let δ be a parameter to the algorithm limiting the number of endpoints permitted inside any cell in the final decomposition. Find an axis aligned box that contains all of the objects in the scene. One way to perform this in linear time is to find the minimally enclosing rectangle around the scene by scanning all the endpoints of the scene to find the most extreme ones. Having found this, extend the rectangle into a square by extending the direction that is shorter. Any other square could also be used for this step, regardless of its size and orientation, as long as it contains the whole scene. This box is our initial starting point for the decomposition.

After a cell is created (either initially or as the result of a split), the number of endpoints contained in the cell must be determined; endpoints that lie on the left or bottom border of a square will be counted by that square. This number must be compared with δ. If more than δ are present, the cell must be split. If fewer than δ are present, then the cell is a leaf in the decomposition, and will not be further subdivided.

There are two possible splits that can be performed on a cell of the quadtree: a binary split and a shrinking split. If it is determined that a cell requires a split, then another check must be made to determine whether "progress" will be made by a binary split. A binary split is a split of a cell into 4 identical sub-cubes, each $\frac{1}{4}$ the area of the initial cell, by making two halving splits of a cell parallel to each boundary. Progress is made by a binary split if at least two of the resulting subcells of a split will contain endpoints after the split. If progress will be made by a binary split then it should be performed, and the process described in the previous paragraph performed recursively on each of the resulting subcells. If progress is not made, as would happen if all the endpoints in a cell are clumped into one corner of

Figure 3.3: Quadtree split types.

the cell, a shrinking split must be made. A shrinking split is a split of a cell into 2 subcells: the shrunken cell and the remaining cell. The shrunken cell, is the smallest boundary aligned square that wraps around the points in the cell, where ambiguity of location of the cell is resolved by placing the cell as close to a boundary and/or a corner of the original cell as possible. The remaining cell is just the original cell minus the shrunken cell - i.e. the endpoint-free space left over. The only cell in the shrinking split that needs to be further processed is the shrunken cell, so the algorithm operations recursively on this cell only. See figure 3.3 to see what these two split types look like.

The existence of the shrinking operation in this algorithm is what makes the algorithm different from a standard quadtree algorithm. The shrinking operation prevents too many extra splits from occurring in the situation where all of the endpoints in a cell are clumped into one small region. If regular binary splits were to be performed on this region, there could be a large number of endpoint free cells created. Each of these cells would contribute to the quadtree complexity, blowing up the complexity estimation of the scene. In fact, we will show in the next lemma that the shrinking operation guarantees that the number of cells in the quadtree decomposition is linear in the cardinality of the set of object endpoints.

Lemma 13 *The number of cells in a quadtree built using the adaptive quadtree*

algorithm is linear in the cardinality of the set of endpoints of the scene.

Proof: To establish this, we note that every time a binary split is made, we make progress by causing at least two endpoints to end up in different cells. If progress is not made by a binary split, we make a shrinking split. The shrinking split does not directly make progress, but it does ensure that a binary split of the shrunken cell will make progress. In this way, we will never have more than one shrinking split for a given subset of endpoints in the scene.

Let n be the number of endpoints. We cannot make progress more than n times. If m is the number of cells than contain endpoints, then after making progress n times, m must be n. This can be seen by the well ordering principle (on the quantity $n - m$) quite easily as each binary split increases m by at least one.

After n splits, there will be $\leq 3n + 1$ cells in the quadtree. This can be seen by induction since a binary split creates 4 cells while removing one old cell and a shrinking split creates 2 cells while removing one old cell. The case for 0 splits holds easily, there is just one cell, the initial square. After $n+1$ splits, it must be that the $n+1^{st}$ split created ≤ 4 cells with one cell being removed from the decomposition for a total of at most 3 new cells. This would give at most $\leq (3n+1)+3 = 3(n+1)+1$ cells in the decomposition.

Thus we are guaranteed to have a linear number of cells in the quadtree using the adaptive quadtree algorithm. □

We now give the formal definition of a quadtree cover of a scene:

Definition 8 *Let $\mathcal{S}$ be a planar scene.*

- *A* δ-quadtree cover *of $\mathcal{S}$ is an endpoint sensitive adaptive quadtree decomposition of $\mathcal{S}$, where each cell in the cover is limited to having δ endpoints in its*

interior.

Using this, we give the definition of the quadtree complexity of a scene:

Definition 9 *Let $\mathcal{S}$ be a planar scene and let δ be a parameter.*

- *The* weight *of a cell in a quadtree cover of $\mathcal{S}$ is defined to be one plus the number of silent intersections with that cell.*
- *We say that $\mathcal{S}$ has (σ, δ)-quadtree complexity if there is a δ-quadtree cover $\mathcal{C}$ such that*

$$\sum_{c \in C} w(c) = \sigma$$

where $w(c)$ refers to the weight of the cell c from C.

- *The δ-quadtree complexity of $\mathcal{S}$ is the largest σ for which $\mathcal{S}$ has (σ, δ)-quadtree complexity. Note this arises from the worst of the many possible quadtree covers of $\mathcal{S}$.*

Informally, we say that a scene has *low quadtree complexity* if its δ-quadtree complexity is cn for a small constant $c > 0$.

The quadtree complexity of a scene is defined to be the *largest* σ for which $\mathcal{S}$ has (σ, δ)-weighted-quadtree complexity rather than the smallest, as might have been expected. The reason for this is to prevent our algorithm from making too many optimizations in the cover. Intuitively, to measure the interaction present in a scene, one may feel that its better to use the best possible orientation for the quadtree origin; doing this would provide a better descriptor, but would make computing the quadtree complexity difficult. There are scenes for which the minimum and the maximum quadtree complexities are significantly different (see next section). It is in fact equally difficult to compute the minimum quadtree complexity as it is

to compute the maximum quadtree complexity. However, it is easy to compute a lower bound on the maximum quadtree complexity. This is done by calculating the complexity of a quadtree cover for the scene that started with an arbitrary starting square. Having a lower bound allows us to show give some interesting results. We show, in theorem 18, that the quadtree complexity computed from an arbitrary starting point will be no more than a constant times the expanded disc complexity of the scene. From this it follows that if a scene has low expanded disc complexity, then the scene will also have low quadtree complexity, regardless of the origin square used in the computation of the quadtree complexity.

3.3.1 Quadtree Origin Differences

Different origins in a quadtree decomposition can cause significant differences in the complexity of the associated quadtree covers. Lets look at some ways in which the complexity of a quadtree cover could change. For the purposes of this discussion, we will assume that we have a scene $\mathcal{S}$ with n line segments that is covered by a quadtree cover $\mathcal{C}$, where $\mathcal{C}$ is neither the minimally nor the maximally complex cover of the scene. One way to increase the complexity of $\mathcal{C}$ is to take a cell in the cover and perform an additional quadtree split. This would increase the complexity by at least 3, depending on how many silent intersections the cell had with line segments from the scene. If there were silent intersections in the cell, then more than one of the subcells could silently intersect the segment, causing the segment to contribute more to the complexity of the scene. Even if no segments were silently intersected, the complexity would increase due to the addition of new cells in the scene, which must now be included in the complexity total. If we permitted cells in our quadtree to continue splitting past the point where the splits are useful, we would get a large

increase in the complexity of the scene due to the presence of the many extra cells. The stopping condition is necessary to be able to talk about a maximum complexity. But even within the bounds of the stopping condition, it is possible to change the complexity of some scenes. This occurs by changing the orientation and size of the origin square used as the root cell in the quadtree decomposition. As the origin moves around, the cells in the decomposition also move around, with new cells possibly being created and some cells possibly being removed, as required to satisfy the stopping condition. If the resultant cover, after the adjustment, creates more cells or causes more cells to silently intersect line segments than in the original cover, then the complexity will increase. If the resultant cover reduces the number of cells or causes fewer silent intersections, then the complexity will decrease. We give an example in theorem 14 of a scene where the maximum quadtree complexity is $\Omega(n)$ times the minimum quadtree complexity. The result also indicates that the same scene has unexpanded disc complexity that is $\Omega(n)$ times the minimum quadtree complexity.

Not all scenes have the property that their minimum and maximum quadtree complexity is significantly different. If a scene is such that its minimum and maximum quadtree complexities are both low or both high, then the scene is somehow uniformly simple or complex, regardless of how it is divided up by a valid quadtree decomposition. If the scene does not possess too much interaction then minimum and maximum quadtree complexities are both low and the quadtree leading to the maximum quadtree complexity does not magnify the amount of interaction present too much. If the scene has a great deal of interaction between objects then the minimum and maximum quadtree complexities are both high, and the quadtree leading to the minimum complexity already detects a large amount of interaction.

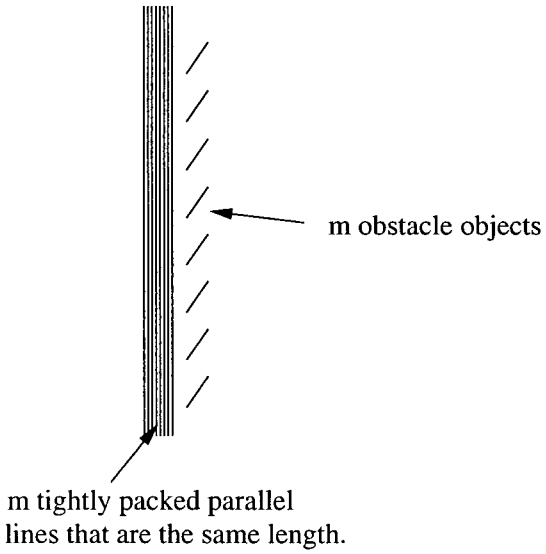


Figure 3.4: A family of scenes whose quadtree complexity and unexpanded disc complexity is $\Omega(n)$ times its minimum quadtree complexity.

If the minimum and maximum quadtree complexities differ by a significant amount, then the amount of interaction detected in the scene depends on the origin and orientation of the initial quadtree cell used in the quadtree decomposition. In this case, we are making a pessimistic choice by defining the quadtree complexity to be the maximum quadtree complexity, but note that in practice, a computed quadtree for such a scene may give a much lower complexity than the maximum quadtree complexity.

Theorem 14 *There is a family of planar scenes consisting of line segments where for each scene, the quadtree complexity and unexpanded disc complexity of the scene is $\Omega(n)$ times its minimum quadtree complexity, where n is the total number of objects in the chosen scene.*

Proof: Consider the scene shown in figure 3.4. This is an example member of a

family of scenes where for each member, its quadtree complexity and unexpanded disc complexity is $\Omega(n)$ times its minimum quadtree complexity. Each member of this family consists of a section of parallel lines and obstacle objects. The long parallel lines on the left are referred to as the *parallel lines*, and the shorter lines on the right as the *obstacle objects*. The number of parallel lines and obstacle objects in each member of the family is the same. Each scene in the family is constructed so that a minimally enclosing ball wrapped around any of the obstacle objects will intersect all of the parallel lines; this is accomplished by ensuring the obstacle objects are sufficiently close to the parallel lines, that they are oriented properly and of sufficient length. Let $\mathcal{S}$ be a particular scene in this family. Let m be the number of parallel lines (and also the number of obstacle objects in $\mathcal{S}$). The total number of objects in the scene is then $n = 2m$. In the following, we assume that δ, the parameter limiting the number of representatives permitted in a covering cell, has the value $\delta = 2$.

To determine an upper bound on the minimum quadtree complexity of $\mathcal{S}$, we first describe the cells needed to cover the line segments in $\mathcal{S}$ so that the complexity will be kept linear in n. The number of additional empty cells created by the splitting process is guaranteed to be linear in the number of representatives by lemma 13; the total number of created cells will be less than $6n + 1$. To minimize the quadtree complexity, we must have the boundary of a large cell falling between the parallel lines and the obstacle objects. To achieve this, the initial square must be such that the vertical centre of the square falls between the parallel lines and the obstacle objects; the horizontal centre lies just λ below the tops of the parallel lines, where $\lambda > 0$ is the horizontal spacing between the parallel lines. See figure 3.5 for a picture of the first few splits in the minimum quadtree decomposition for this scene. The

size of the initial square must be such that the distance from the horizontal centre to the bottom is the length of the parallel lines plus $(m-2)\lambda$. This will cause a single large cell in the lower left quadrant to intersect most of the area of the parallel lines. The tops and bottoms of the parallel lines will each end up being covered with shrunken cells that contain endpoints as well as length λ segments of the tops and bottoms of the parallel lines. The m obstacles will all lie in the lower-right quadrant. The obstacles will cause at most m binary splits to occur inside, creating m cells that cover the objects. Thus the complexity of all cells intersecting the parallel lines is m for the large cell in the lower-left quadrant, m for the cells that contain the top endpoints and m for the cells that contain the bottom endpoints of the parallel lines. The complexity caused by silent intersections is thus $3m$ or $3\frac{n}{2}$. Adding in the complexity caused by the cells of the quadtree we have that the minimum quadtree complexity of $\mathcal{S}$ is less than

$$3\frac{n}{2} + 6n + 1 = \frac{15n+2}{2}$$

Note that increasing δ would lower the minimum quadtree complexity by a proportionate factor. This would come about because more representatives could intersect a given cell, reducing the number of cells in the cover.

The (maximum) quadtree complexity of $\mathcal{S}$ is bounded by placing the initial square so that it is at least a distance $(m-1)\lambda$ to the left of where it is in the minimum quadtree cover. In this case each of the small cells created to contain obstacle objects will also be forced to intersect a section of the parallel lines. Since there are m small cells containing the obstacles and there are m parallel lines, the additional contribution to the complexity from this is $m^2 = \frac{1}{4}n^2$. This puts a superlinear lower bound on the quadtree complexity of $\mathcal{S}$ showing that its quadtree complexity is $\Omega(n)$ times its minimum quadtree complexity.

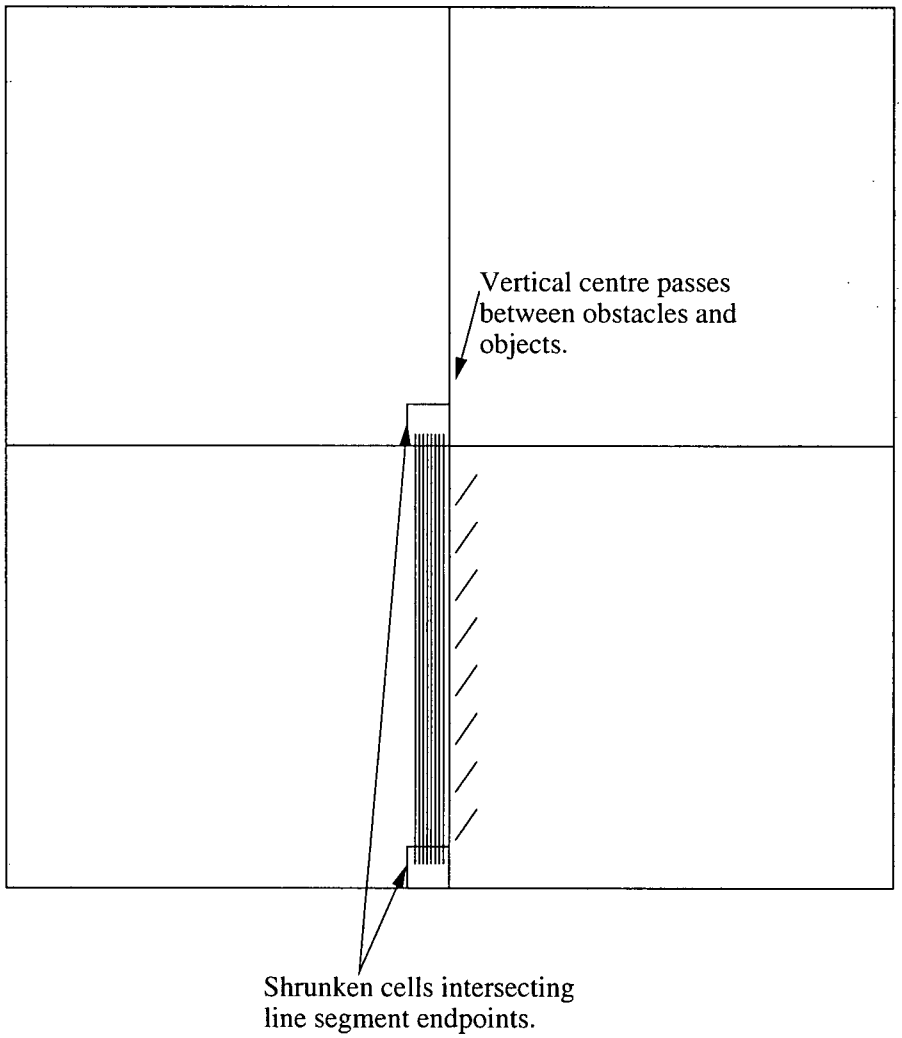


Figure 3.5: First few splits in the minimum quadtree decomposition for the scene.

We now determine the (minimum) unexpanded disc complexity of $\mathcal{S}$. By the construction of $\mathcal{S}$, any attempt to surround the obstacle objects by discs will force the parallel lines to be intersected by $\Omega(m)$ discs. The scene was constructed so that the minimally enclosing ball around any of the obstacle objects would be forced to pass through all of the parallel lines. Enlarging the minimally enclosing balls to allow them to intersect more than one line segment (and hence more than 2 object representatives) will not change this by more than a constant factor; the most obstacles that could be entirely contained in a single disc is $\frac{\delta}{2}$, because each obstacle has two endpoints. At least $\Omega(m)$ discs must intersect the parallel lines, adding $\Omega(n^2)$ to the disc complexity. This puts a superlinear lower bound on the unexpanded disc complexity of $\mathcal{S}$ showing that its unexpanded disc complexity is $\Omega(n)$ times its minimum quadtree complexity. □

3.3.2 Relationships

We now prove that low unexpanded disc complexity is not a sufficient condition for low quadtree complexity. This result is part of the reason that we looked towards expanded disc complexity as a disc complexity measure that would hopefully provide a nice relationship between the disc and quadtree based weighted complexity measures.

Theorem 15 *There is a family of planar scenes consisting of line segments where for each scene, the quadtree complexity of the scene is $\Omega(n)$ times its unexpanded disc complexity, where n is the total number of objects in the chosen scene.*

Proof: Consider the scene shown in figure 3.6. This is an example member of a family of scenes where for each member, its quadtree complexity is $\Omega(n)$ times its

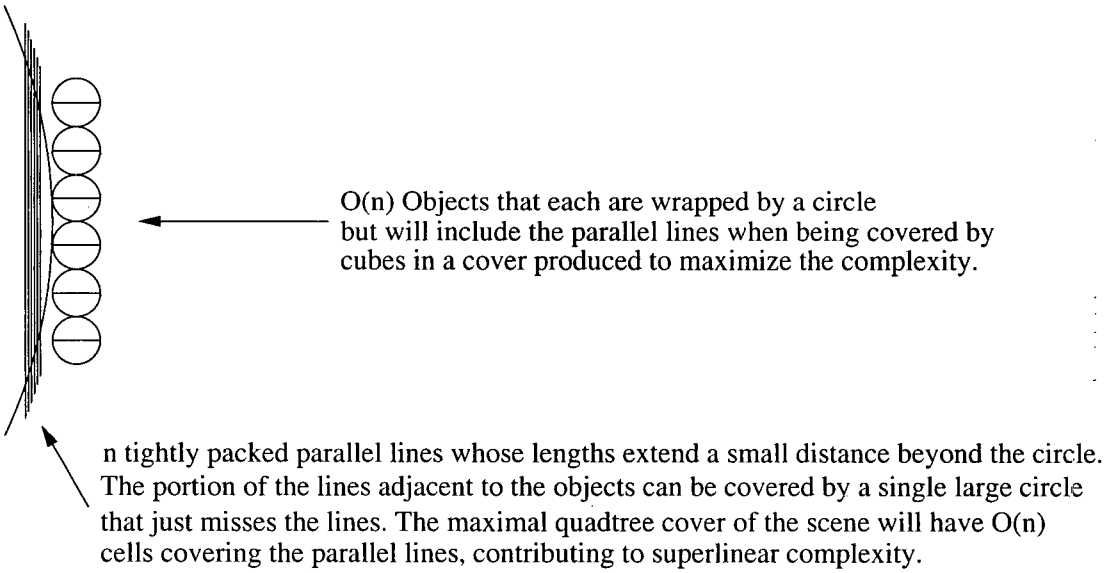


Figure 3.6: A family of scenes whose quadtree complexity is $\Omega(n)$ times its unexpanded disc complexity.

unexpanded disc complexity. Each member of the family consists of a section of tightly packed parallel lines on the left and obstacle objects on the right. Let $\mathcal{S}$ be a particular scene in the family. Let m be the number of parallel lines and also the number of obstacle objects. This gives a total of $n = 2m$ line segments in the scene. Let λ be the interline distance between the parallel lines. The lengths of the parallel lines in $\mathcal{S}$ are set so that each line extends a distance of λ beyond the boundary of a large covering disc that squeezes in between the parallel lines and the obstacle objects (see figure 3.6). Doing this prevents too many discs from being used to cover the parallel lines since a single disc can cover most of the area between the parallel lines. Note, in the figure, that the vertical range of the smallest of the parallel lines includes the vertical range of a bounding box around the obstacle lines; that is, no obstacle is above or below the smallest of the parallel lines. Finally, we

assume for this proof that δ, the bound on the number of permitted representatives in a covering cell, is $\delta = 2$.

Determining the disc complexity of $\mathcal{S}$, we cover the n parallel lines with $2m + 1$ discs: the one large disc, and $2m$ smaller discs to cover the endpoints of the parallel lines. The complexity of this is $m + 1$ for the large disc plus $2m$ for the smaller discs. To cover the remaining m obstacle objects beside the parallel lines, an additional m discs can be used, each disc containing exactly one obstacle object; this adds m to the complexity. The preceding only covers the objects in the scene. To ensure the entire bounding box of the scene is covered, we must cover the complement of the union of the discs inside the bounding box. By looking at the figure, it is clear that there are pieces to be covered between the parallel lines and the obstacle objects, above and below the obstacle objects and on the right side of the obstacle objects between the covering discs. There will be up to $2m$ regions to be covered around the obstacle objects; each of these can be covered by a single disc that does not silently intersect any object in the scene. As well, the regions above and below the obstacle objects can be covered with an additional two discs that do not silently intersect any object. This causes a total increase to the disc complexity of the scene of no more than $2m + 2$. The overall unexpanded disc complexity of the scene is therefore $(m + 1) + (2m) + (m) + (2m + 2) = 6m + 3 = 3n + 3$.

To see why the quadtree complexity of $\mathcal{S}$ is $\Omega(n)$ times its unexpanded disc complexity, we show that there is an origin square that leads to a quadtree decomposition with the needed complexity. To accomplish this, we specify a cover where the parallel lines are covered with more than a constant number of quadtree cells, forcing the parallel lines to contribute a $\Omega(n^2)$ amount to the quadtree complexity of the scene. The origin square of quadtree decomposition can be set so that ev-

ery cell in the quadtree decomposition that contains obstacle objects also intersects every one of the parallel lines. This is done by making the origin square twice as wide as the minimum possible width, and placing it so that its vertical centre line passes to the left of the parallel lines. The first split will cause the entire scene to be contained on the right-hand side of the quadtree. Subsequent splits will continue until there are no more than a constant number of witnesses in each quadtree cell. The splitting caused by the obstacle objects will terminate when each cell contains a constant number of obstacles. By placement, the cells that contain the obstacles will also have the parallel lines passing through them, as they are just to the left of the obstacles. Since there are m obstacles, we will have $\Omega(m)$ quadtree cells covering the obstacles, and as a result the parallel lines. Each of the $\Omega(m)$ quadtree cells will contribute $\Omega(m)$ to the complexity, resulting in a contribution of $\Omega(n^2)$ to the overall complexity by the parallel lines. This proves that the quadtree complexity of $\mathcal{S}$ is $\Omega(n)$ times the unexpanded disc complexity of $\mathcal{S}$. □

The above theorem not only shows us an example where the unexpanded disc complexity is low and the quadtree complexity is high, it also shows us one of the properties of unexpanded discs. It is possible to construct scenes whose unexpanded disc complexity is very sensitive to small perturbations in the locations of the objects in the scene. The unexpanded disc complexity would increase quite a bit if even one of the obstacle objects were to move closer to the parallel lines. In doing this, the large disc would be pressed closer to the parallel lines, and would therefore expose a larger portion of the ends of the parallel lines. To cover these ends would require a large number of small discs, or about $O(n)$ discs that intersect most of the lines.

If we now consider using expanded discs to cover this scene, we will find that we cannot squeeze a large disc between the parallel lines and the obstacle

objects. The reason for this is that the ϵ-expansion of this disc would be quite large - large enough to contain the obstacles. As a result, smaller discs are needed to cover the parallel lines. In fact it is easy to see that this scene has high expanded disc complexity. Using theorem 20 proved in section 3.4, we know that a scene consisting of only n tightly packed parallel lines has high expanded disc complexity. The addition of the n obstacles will not improve this, and will in fact force even more expanded discs to be used to cover the parallel lines.

This observation suggests that a relationship between quadtree complexity and expanded disc complexity may exist. We explore this question, giving a positive result, in the next subsection.

3.3.3 Low Expanded Disc Complexity Implies Low Quadtree Complexity

In this section we will establish that there is a constant $c > 0$ based on ϵ such that for any scene its quadtree complexity will be at most c times its expanded disc complexity. Thus, if a scene has linear expanded disc complexity, then it follows that it will have linear quadtree complexity. We do this by first noting that regardless of the origin square used in the quadtree decomposition of the scene, the number of cells in the resulting decomposition is linear in the number of object faces in the scene. Then we argue that there is a constant $c > 0$ such that for any particular line segment in the scene, the number of quadtree cells needed to cover that object is no more than c times the number of expanded discs needed to cover it in the best expanded disc cover.

By lemma 13, we know that the number of cells needed in a quadtree decomposition of a scene is linear in the cardinality of the set of endpoints. This tells

us that it is not possible to attribute an increase in the quadtree complexity as compared to the expanded disc complexity to there being a superlinear number of cells in the quadtree decomposition.

We now show that every line segment covered by d expanded discs in the minimum expanded disc cover can be covered by cd quadtree cells in the maximum quadtree cover, where $c = \frac{24\sqrt{2}}{\epsilon}$ is a constant. To establish this, we prove the following two lemmata. The first shows that at least a third of the intersections between cells and line segments contain a sizable portion of the line segment. We call an intersection a *healthy* intersection if it contains a piece of the line segment that is proportional to ϵ in length. The second lemma shows there cannot be more than a constant number $(\frac{8\sqrt{2}}{\epsilon})$ of healthy intersections with cells from a quadtree cover in the region of a line segment covered by a particular disc from the expanded disc cover.

To define exactly what we will consider to be a *healthy* and an *unhealthy* intersection, we must define and measure what we call the *minimally intersecting cell.* Consider an arbitrary disc, d, that you might find in an expanded disc cover. By definition of the model, there cannot be more than a constant, δ, object endpoints inside d (or its associated expanded disc). Suppose a line segment, l, passes through d, and that we want to cover this l by a collection of quadtree cells. By the quadtree decomposition algorithm, every cell created must have a reason for its creation. This reason is that there were more than a constant, δ, object endpoints in the neighbourhood of the cell, where the neighbourhood is the union of all possible parent cells that could have created the cell; such a region is a square with side lengths 3 times that of the cell and where the cell appears in the centre. For the neighbourhood of a cell within d to intersect more than δ endpoints, it must extend

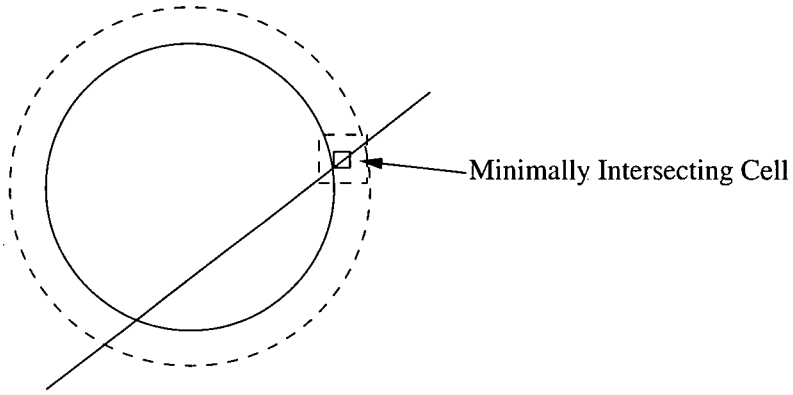


Figure 3.7: The minimally intersecting cell occurs in the ϵ-expansion of the disc.

beyond the ϵ-expansion of d. The *minimally intersecting cell* is defined as the smallest valid cell that could intersect any line segment inside the ϵ-expansion of d. A cell that intersects the line segment in the centre of the ϵ-expansion of d must be larger than a cell that intersects the line segment near the edge of the ϵ-expansion of d. The size of the cell needed to create a particular length of intersection decreases as the distance from the centre of d increases. As a result, the minimally intersecting cell occurs in the ϵ-expansion portion of the expanded disc (see figure 3.7). The size of the minimally intersecting cell acts as a lower bound on the size of any cell that could intersect l inside d. The size of a minimally intersecting cell is bounded from below by a cell, c, occurring in the expansion of d positioned so that a line segment exits d and passes along the diagonal of c, where the length of the c's diagonal is $\frac{\epsilon r}{2}$. Thus, the size of a minimally intersecting cell must be greater than $\frac{\epsilon r}{2\sqrt{2}}$.

Having this, we define an intersection between a cell and a line segment within an expanded disc with radius r to be *healthy* if the length of the intersection is larger than or equal to $\frac{\epsilon r}{4\sqrt{2}}$. The size of a healthy intersection must be larger or equal to one half the size of the minimally intersecting cell. An intersection is

defined to be *unhealthy* if it is not healthy.

For an intersection to be unhealthy, note that the line segment must cut off exactly one corner of the cell. This is true because the cell creating the intersection must be larger than or equal to the size of the minimally intersecting cell. If an unhealthy intersection cut off exactly two corners of the cell, then the length of the intersection would be at least $\frac{\epsilon r}{2\sqrt{2}}$, making the intersection healthy.

Lemma 16 *Suppose c is the intersection of a line segment from the scene and a disc from a minimum expanded disc cover. It is impossible to have more than 2 consecutive unhealthy intersections between cells of a quadtree cover and c. Thus at most $\frac{2}{3}$ of the intersecting cells from a quadtree cover of the scene will form unhealthy intersections with c.*

Proof: For the purposes of this proof, we will permit cells to float around without the restriction that they be formed from a sequence of quadtree splits. We prove that even with this relaxation, the result holds. Using the restriction that cells must come from a valid quadtree decomposition can only increase the fraction of healthy intersections of cells with the line segment.

Without loss of generality, we will assume that c has an upward, or positive slope. We claim that the maximum number of consecutive cells that intersect c in an unhealthy way is 2. There are two cases possible: an *upper-left, lower-right intersection* and a *lower-right, upper-left intersection.* See figure 3.8.

In the lower-right, upper-left intersection case, c passes through the right half of the bottom of the left-most square, and out through the left half of the right-most square. Consider placing a third square, s, above the right-most square. How might it be that s could produce a third unhealthy intersection with c while remaining at

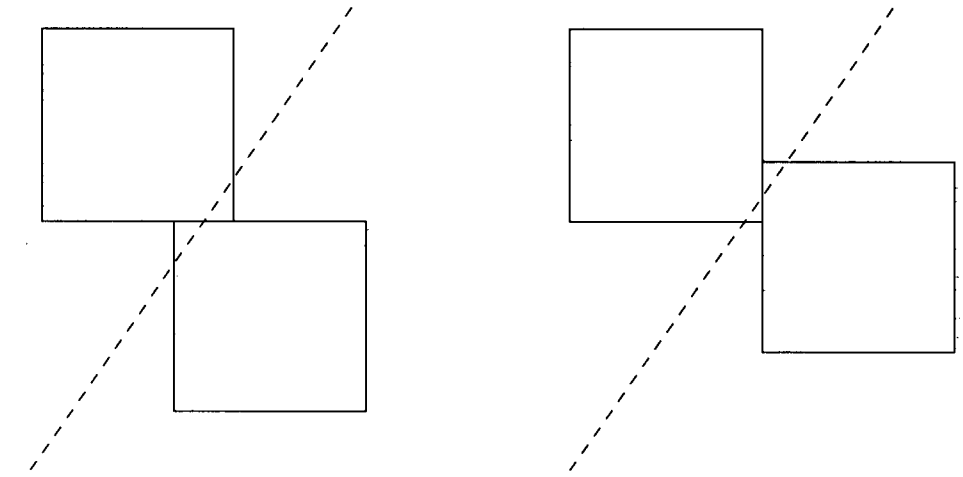


Figure 3.8: A lower-right, upper left and an upper-left, lower-right intersection with a positively sloped line.

least as large as the minimally intersecting cell? Since c exits the right-most square from the top, c will enter the bottom of s and will thus need to exit the right side of s for the intersection to be unhealthy. Now consider how far along the top of the right-most square, c could exit, measured from the left-most square; this distance must be less than $\frac{\epsilon r}{4\sqrt{2}}$, or else the intersection between c and the right-most square would have been healthy. Suppose we try to fit in the minimally intersecting cell as s. The size of the intersection between s and c would be minimized by placing s as far left as possible on top of the right-most square. Doing so would cause c to enter s to the left of the centre-point of the bottom of s. The size of the resulting intersection between c and s is therefore larger than $\frac{\epsilon r}{4\sqrt{2}}$. This implies that c intersects s in a healthy fashion. Using a square larger than the minimally intersecting cell for s would result in a larger intersection. We are forced to conclude that s must intersect c in a healthy fashion regardless of the size of s. By symmetry, placing s below the

left-most square will also result in a healthy intersection.

The argument for the upper-right, lower-left intersection case follows from the previous argument again by symmetry.

If we have had 2 consecutive unhealthy intersections on c, then we know that the next intersection must be healthy. This gives an upper bound of $\frac{2}{3}$, as the fraction of the total number of intersections with c that are unhealthy. □

Now we establish that for any intersection between a line segment from the scene and a disc from a disc cover of the scene that there the number of healthy intersections between the portion of the line segment inside that disc and cells in any quadtree cover is bounded from above by the constant $\frac{24\sqrt{2}}{\epsilon}$.

Lemma 17 *Suppose c is a line segment that is covered by d discs from a minimum expanded disc cover. The maximum number of cells from a quadtree cover that form healthy intersections with c is $\frac{8\sqrt{2}d}{\epsilon}$, where $\epsilon > 0$ is the expansion factor of the expanded disc cover.*

Proof: Consider any of the expanded discs that intersect c in the minimum expanded disc cover, and label it e - the following argument applies equally to any of them. Suppose the radius of e is r. The maximum number of healthy intersections with c that could exist inside e is equal to the diameter of the disc divided by the size of the smallest possible healthy intersection. This is:

$$\frac{2r}{\frac{\epsilon r}{4\sqrt{2}}} = \frac{8\sqrt{2}}{\epsilon}$$

Since there are d discs covering c, there can be at most

$$\frac{8\sqrt{2}d}{\epsilon}$$

cells that form healthy intersections with c. □

We have just shown in lemma 16 that there can be at most two unhealthy intersections with c for every healthy intersection with c. Accounting for both health and unhealthy intersections with c inside e gives us a total of

$$3\frac{8\sqrt{2}}{\epsilon} = \frac{24\sqrt{2}}{\epsilon}$$

intersections.

Putting these two lemmata together establishes the following theorem. Since the number of unhealthy intersections is at most twice the number of healthy intersections, and the number of healthy intersections is bounded by $\frac{8\sqrt{2}}{\epsilon}$, we have that the contribution to the quadtree complexity of the scene of any line segment is no worse than

$$3\frac{8\sqrt{2}}{\epsilon} = \frac{24\sqrt{2}}{\epsilon}$$

times the contribution of that segment in the minimum expanded disc cover. If we suppose that the scene has (σ, δ)-expanded disc complexity, then it follows from the previous discussion that the δ-quadtree complexity will be bounded by $\frac{24\sqrt{2}}{\epsilon}\sigma$, which proves:

Theorem 18 *If a scene has (σ, δ)-expanded disc complexity then it has $(\frac{24\sqrt{2}}{\epsilon}\sigma, \delta)$-quadtree complexity.*

3.3.4 Complexity of Computation

We now look at how hard it is to determine the quadtree complexity of a scene. In our description of the algorithm to this point, we have implicitly assumed that each cell in the quadtree decomposition would know which endpoints it contains. Then for the computation of the cell weights, each cell needs to know which objects it silently intersects. Thus, the algorithm proceeds in two steps, the first uses the set

of endpoints to drive the quadtree decomposition. During each split, the endpoints, and the intersecting line segments are reassigned to the subcells. The second step calculates the complexity of the quadtree decomposition using the line segment intersection information stored in each cell.

More specifically, we assume that each cell in the decomposition stores a number of binary search trees and lists. One list will keep track of the endpoints contained in the cell sorted by one coordinate of the endpoints (with ties broken by the other coordinate). Four binary search trees (preferably AVL or Red-Black trees), will also be maintained to keep track of the line segments that enter each cell along each of the four boundaries of the cell. Each of these trees will store intersections between line segments and one of the boundaries in clockwise order. A final list, which we call the list of free line segments, will be needed to store the line segments that are entirely contained in a cell. This final list does not need to be ordered in any particular way.

We must efficiently maintain the list of endpoints in the subcells created by splitting operations. The problem with a naive approach is that unbalanced splitting by binary splits can lead to $\Omega(n^2)$ complexity. Vaidya [16] and Callahan and Kosaraju [3] both describe techniques for efficiently maintaining lists of points owned by subcells while building a quadtree-like structure. They actually divide cells using a single splitting-plane, but their approach can easily be extended to quadtree style splits. Their techniques result in a construction time of $O(n \log n)$ for building the quadtree decomposition.

In addition to maintaining the endpoint lists, we need to maintain the four search trees and the list of free line segments in each subcell of a particular binary or shrinking split. We must use time proportional to $ni(\mathcal{O}) + \log(n)$ to accomplish

this for each split, where $ni(\mathcal{O})$ stands for the number of intersections with $\mathcal{O}$. If we can do this, we attain an overall complexity of $O(n \log n + QTC(\mathcal{S}))$ for maintaining the object intersection data in the subcells.

We will first deal with binary splits. The first step in performing the update for a binary split is to partition the top and bottom trees into those line segments that enter to the left and right of the vertical split line, and to partition the left and right trees into those line segments that enter above and below the horizontal split line. The split locations can be found and partitions performed in $O(\log n)$ time. The second thing to note is that a line segment crosses a split line if it enters the cell on one side of the split line and exits on the other side. As well, since the line segments in a scene are disjoint, the order in which the line segments intersect the border corresponds to the order in which they cross the split line (if they do in fact cross the split line). To determine all the silent intersections between line segments and the vertical split line it is enough to proceed in a clockwise fashion from the bottom of the vertical split line along the left border to the top of the vertical split line. As each intersection is found, a simple check is made to see if it intersects the vertical split line. If so, it is added to the set of intersections with the vertical split line. If it does not, the line is skipped. A similar technique would be used to assign the appropriate intersections to the horizontal split line. This probe would catch all of the silent intersections in the cell, and some of the line segments that leave one endpoint in the cell. To include the missing lines leaving one endpoint requires a minor modification to this procedure. That is, two traversals around the cell must be made in tandem: (in the vertical split case) one from bottom to top along to the left of the vertical split and one from bottom to top along the right of the vertical split. It is simple to note that the silent intersections will intersect both traversals

in the same order. As such, if a line segment is detected to cross the split line in one traversal below the crossing of the split line by the line in the other traversal then the line segment from the first traversal should be included in the tree for the split line. Whenever it is detected that the same object is referred to by both traversals, the intersection should be recorded on the split line, and each border traversal moved to the next intersection. In this fashion, almost all of the intersections will be pegged to the split lines. The only remaining possible intersections are those objects in the list of free line segments. For each of these segments, we are willing to spend $O(\log n)$ time if they cross a split line to add them to the appropriate split tree. Otherwise, the line will be added to the free list of the subcell that contains it. This will add an $O(n \log n)$ factor to the time complexity since each line will be pegged to a border at most once. Overall, we have a complexity of $O(\log n)$ for the searches to locate the split points in the original trees plus $O(ni(\mathcal{C}))$ to create the new trees and lists. This gives the needed $O(\log n + ni(\mathcal{C}))$ complexity.

Shrinking splits can be dealt with very simply. The intersections stored in the trees along the border should be traversed in clockwise order. For each segment that is encountered, it should be checked if it intersects the shrunken cell. If so, the intersection should be noted in the tree corresponding to the border of the shrunken cell where the intersection is located (if there are two intersections, only the one closest to the border being traversed should be recorded at this time - the second intersection will be detected later). Each intersection will be found in order, since the line segments do not intersect one another. Thus the trees in the subcell will be built in order, and hence can be built in linear time in the number of intersections. Afterward the list of free line segments will be traversed. Each object that intersects the shrunken cell should be added to the appropriate border tree or the free list.

This can be done in $O(\log n)$ time per line segment intersection with the border of the shrunken cell, or $O(1)$ time for adding to the free list. Overall we have a complexity of $O(\log n + ni(\mathcal{C}))$ for updating the border trees.

To see how this leads to $O(n \log n + QTC(\mathcal{S}))$ complexity overall, note that since there are $O(n)$ cells in the final quadtree there are also $O(n)$ interior nodes of the quadtree. Together $O(\log n)$ work in each intermediate cell leads to an $O(n \log n)$ overall contribution. As well, each line segment, l, will be copied into a new tree $O(wt(l))$ times, where $wt(l)$ represents the number of times l gets split in the created quadtree. As such, summing this work over all the objects in the scene gives a complexity of

$$\sum_{l \in \mathcal{S}} O(wt(l)) = O\left(\sum_{C \in QT} (\mathcal{C})\right) = O(QTC(\mathcal{S}))$$

Putting the contributions together gives the needed result.

After creating the quadtree, with the object intersection information, computing the quadtree complexity of the scene will take an additional $O(n+QTC(\mathcal{S}))$ time to determine the number of silent intersections in each of the resulting cells of the quadtree.

Overall we have proven the following theorem:

Theorem 19 *Let $\mathcal{S}$ be a planar scene. We can compute the quadtree complexity of $\mathcal{S}$, written $QTC(\mathcal{S})$, in $O(n \log n + QTC(\mathcal{S}))$ time.*

Thus we can determine the quadtree complexity in time proportional to the quadtree complexity of the scene plus an $n \log n$ factor. If the quadtree complexity of the scene is linear in the number of objects or even $O(n \log n)$, then it takes $O(n \log n)$ time to determine the quadtree complexity of the scene. This is further evidence that the quadtree complexity for a scene is easier to determine than either of the disc-based complexity measures of the scene.

3.4 Weighted Complexity of Tightly Packed Parallel Lines

In this section we see how well the various weighted models, discussed in this chapter, measure the complexity of tightly packed parallel lines. Precisely, the scene we will be looking at consists of n identical parallel lines of length l that start and end in alignment where the distance between the leftmost and the rightmost line is λ. We assume $l > \lambda$, that is the length of the lines is larger than the separation between the leftmost and the rightmost lines.

One of our criteria of goodness for a model is that it should assign a low complexity to this scene. This section allows the various models to be compared under this criteria.

In the introduction we proved theorem 9 which showed that the models of the previous chapter give this scene a high complexity. That is, it does not consist of fat-objects, it has density and clutter factor equal to the number of lines, it requires $\Omega(n^2)$ discs in a simple cover and $\Omega(n^2)$ guards in a guarding set. The following theorem shows the complexities assigned to this scene by the weighted models of this chapter.

Theorem 20 *Let $\mathcal{S}$ be a planar scene consisting of n identical vertical parallel lines each of length l that start and end in alignment and are equally spaced. Let $\lambda < l$ be the separation between the leftmost and the rightmost lines. Let $\delta > 0$ be the parameter limiting the number of representatives inside a covering object.*

- *$\mathcal{S}$ has $(\delta, \Omega(\log \frac{nl}{\lambda}))$-expanded disc complexity.*
- *$\mathcal{S}$ has $(\delta, \Theta(n \log n))$-quadtree complexity.*

- *$\mathcal{S}$ has $(\delta, \Theta(n))$-unexpanded disc complexity.*

Proof: To show a lower bound on the expanded disc complexity of $\mathcal{S}$, we will describe the number of discs needed to cover just the centre line in the collection. The reason we can not cover this line with just one disc is because the ϵ-expansion of the disc would contain too many other endpoints. The first disc we use will have radius $\frac{\epsilon l}{(\epsilon+1)2}$. This disc will leave a hole of size ϵr on either end of the line that we must cover. Focusing on covering one end of the line, we repeatedly place the largest disc possible in the remaining space until some disc leaves less space than the separation between the central line and its neighbours. If we were to stop before this point, we would not be able to guarantee that one more disc would be needed to cover the final segment of the line nor that the next disc would intersect less than δ object endpoints. After getting to this point, one more disc can be placed to cover the endpoint, without intersecting any other endpoints. Since the lines are equally spaced, the space between each line is $\frac{\lambda}{n}$. It is simple to establish (see lemma 21 below) that the space left at the end to cover after the placement of m discs is

$$\frac{\epsilon^m l}{(\epsilon+1)(\epsilon+2)^{m-1}2}$$

Thus we need at least

$$2\left\lceil \log_{\frac{\epsilon+2}{\epsilon}} \frac{nl}{2\lambda} \right\rceil$$

discs to ensure that the centre line is covered. Thus $\Omega(\log \frac{nl}{\lambda})$ discs are needed to cover $\mathcal{S}$. This quantity can be made arbitrarily large by increasing l and or decreasing λ.

The δ-quadtree complexity of $\mathcal{S}$ is $\Theta(n \log n)$. The reason for the improvement in this model as compared to the expanded disc model is the shrink-wrapping

operation used in the quadtree decomposition algorithm. We will proceed by showing that the quadtree complexity of $\mathcal{S}$ can be as bad as $\Omega(n \log n)$. Then we will argue that the quadtree complexity is never worse than $O(n \log n)$.

We start with an origin square that is oriented 45° to the parallel lines, and contains the parallel lines, as in figure 3.9. The origin square will have one binary split performed on it. Then in each of the two cells containing endpoints, a shrunken cell will wrap around the endpoints of the lines. Inside each of the shrunken cells, we will see further binary splitting in the cells that contain more than δ endpoints. Note that every binary split in a cell that contains m endpoints creates a subcell that silently intersects $\lfloor \frac{m}{2} \rfloor$ line segments. Each binary split also creates two subcells that each contain $\lfloor \frac{m}{2} \rfloor$ endpoints. After $q > 0$ levels of splitting, a simple induction shows that there will be 2^q cells each with $\frac{n}{2^q}$ endpoints. Thus, there will be a total of no more than $\log n$ levels of splitting. On each of the $\log n$ levels of splitting, cells were created that in aggregate silently intersected a total of $\frac{n}{2}$ line segments. This leads to a quadtree complexity of $\Omega(\frac{n}{2} \log n)$ for each of the shrunken cells, and hence a total quadtree complexity of $\Omega(n \log n)$.

Now to see that the quadtree complexity of this scene can never be worse than $O(n \log n)$, note that regardless of the orientation of the origin square the with respect to the parallel lines, no more than four cells will be used to shrink-wrap to the endpoints of the lines. Inside each of the shrunken cells, there will be $O(\log n)$ levels of splitting, since after at most one unbalanced binary split, there must be a balanced binary split.

This can be proved by a simple case by case analysis the details of which we omit. The idea in proving this is to note that the endpoints inside a shrunken cell form a line. If this line touches two opposite boundaries of a cell, then a binary

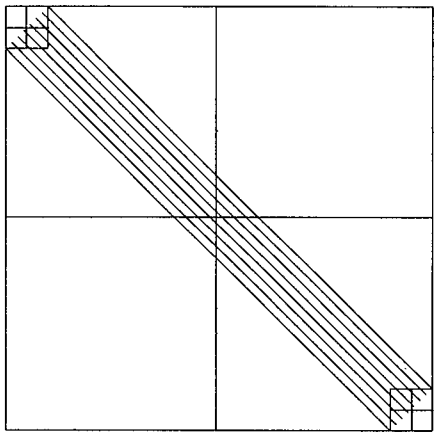


Figure 3.9: A quadtree cover of a set of 8 parallel lines.

split of that cell will be balanced. If the line cuts off a corner, then there are a number of cases. It may be that a binary split of the cell will be balanced in this case. However, if a binary split is not balanced but still makes progress, then most of the points will be located in one cell after the binary split (if the binary split does not make progress, then a shrinking split will be made, which will force a balanced binary split at the next level of splitting). Looking at the possible locations for the line of points in the subcell after the binary split reveals that the line must cut through a large portion of the subcell. This implies that the next binary split of the subcell, with the majority of points, will be balanced.

For each of the $O(\log n)$ levels of splitting, there can be no more than n silent intersections present. Together this puts an upper bound on the δ-quadtree complexity of $\mathcal{S}$ of $O(n \log n)$. Putting the lower and upper bounds together shows that the δ-quadtree complexity of the scene is $\Theta(n \log n)$.

Under the unexpanded disc complexity model, $\mathcal{S}$ has $\Theta(n)$ complexity. For the upper bound, note that in this model, one disc can come to within at least $\frac{\lambda}{n}$

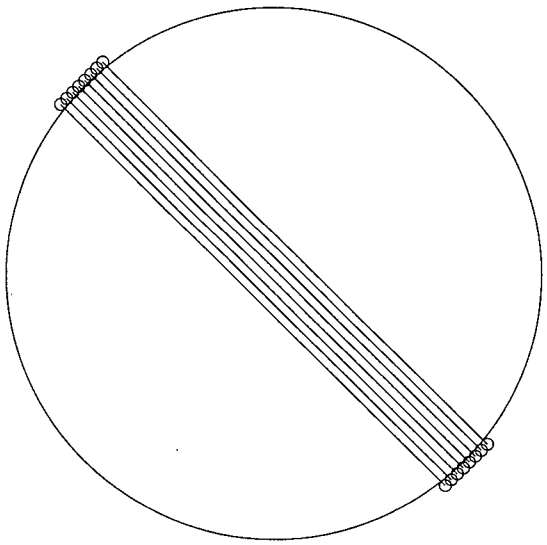


Figure 3.10: A low complexity unexpanded disc cover of a set of 8 parallel lines.

of the ends of the lines, without intersecting more than δ endpoints of the lines (we are implicitly assuming that l is much larger than λ, the separation between the leftmost and rightmost lines, for this to be true). The remaining ends of the lines can then easily be covered by a constant number of discs per endpoint . See figure 3.10 for a picture. The result is that the scene is covered by at most a linear number of discs, and hence the scene has $O(n)$ unexpanded disc complexity. For the lower bound, it is enough to note that the unexpanded disc complexity cannot be less than n, the number of line segments. $\mathcal{S}$ therefore has $\Omega(n)$ unexpanded disc complexity. Together, this proves that the δ-unexpanded disc complexity of $\mathcal{S}$ is $\Theta(n)$. □

Lemma 21 *Let l_m be the remaining distance to cover on the centre line after placing m discs using the method described in the discussion of expanded disc complexity in the above theorem. Then*

$$l_m = \frac{l\epsilon^m}{2(\epsilon+1)(\epsilon+2)^{m-1}}$$

Thus to get a disc of radius smaller than $\frac{\lambda}{n}$ we need at least

$$\left\lceil \log_{\frac{\epsilon+2}{\epsilon}} \frac{nl}{2\lambda} \right\rceil$$

discs to cover one half of the line.

Proof: The first disc placed has a radius of $r_0 = \frac{l_0}{\epsilon+1}$. This leaves

$$l_1 = \frac{l}{2} - r_0 = \frac{\epsilon l}{(\epsilon+1)2}$$

to be covered. To cover this space we will use a disc that fits on top of the previous disc, and extends as high as possible without the expansion of the disc extending beyond the end of the line. The radius of this next disc must satisfy

$$2r_1 + \epsilon r_1 = l_1$$

or

$$r_1 = \frac{l_1}{2+\epsilon}$$

This covers $2r_1$ of l_1. We still have to cover

$$l_2 = l_1 - 2r_1 = \frac{\epsilon}{\epsilon+2} l_1 = \frac{\epsilon^2 l}{(\epsilon+1)(\epsilon+2)^{2-1}2}$$

This pattern continues with subsequent circles, to give us the following function:

$$l_m = \begin{cases} \frac{l}{2} & m = 0 \\ \frac{\epsilon^m l}{(\epsilon+1)(\epsilon+2)^{m-1}2} & m > 0 \end{cases}$$

Using this we can determine for what m, l_m is smaller than $\frac{\lambda}{n}$, the interline distance, by solving for m:

$$\frac{\lambda}{n} > l_m = \frac{\epsilon^m l}{(\epsilon+1)(\epsilon+2)^{m-1}2} > \left(\frac{\epsilon}{\epsilon+2}\right)^m \frac{l}{2}$$

and thus

$$\left(\frac{\epsilon+2}{\epsilon}\right)^m > \frac{nl}{2\lambda}$$

taking logarithms we see that

$$m > \frac{\log \frac{nl}{2\lambda}}{\log \frac{\epsilon+2}{\epsilon}} = \log_{\frac{\epsilon+2}{\epsilon}} \frac{nl}{2\lambda}$$

Thus using the ceiling of this value for m will ensure that we have enough discs to cover one half of the line. □

From the theorem it is clear that the expanded disc complexity assigns a complexity that is proportional to the logarithm of the n times the ratio between the length and the separation of the parallel lines. In this way, the expanded disc complexity of the scene can be made arbitrarily large by increasing the length of the lines and/or decreasing the width of the set of parallel lines. However, in practice, assuming that the ratio between the length of the parallel lines and the width is in $\left(\frac{\epsilon+2}{\epsilon}\right)^{O(n)}$, the expanded disc complexity model will measure a low complexity for the scene. As such the model is better in practice than this criterion would predict especially given its additional property of being resistant to small perturbations in the scene. However, to have the assurance of a reasonably low complexity assignment to the scene regardless of the ratio between the length to width, one must use either the unexpanded disc complexity model or the quadtree complexity model. The unexpanded disc complexity model gives the lowest complexity assignment to the scene, but is hard to compute. The quadtree complexity model gives a complexity assignment to this scene that is in-between the expanded disc and unexpanded disc models, with the advantage of being simple to compute. As such the quadtree complexity model is our preferred model.

3.5 Application: Binary Space Partitions

In this section, we look at the application of binary space partitions. We give a brief introduction to the problem, and then proceed to show that the size of a binary space partition for a scene is at worst proportional to its quadtree complexity. After we present a possible modification to the quadtree decomposition algorithm that lowers the size of the binary space partition for many scenes below that which is given by our quadtree decomposition algorithm.

3.5.1 Binary Space Partitions of Linear Size

Suppose you have a scene of n disjoint objects in the plane. The binary space partition (BSP) problem asks for a partition of the scene into a number of regions so that each region contains no more than one object from the scene. More specifically, the partition is created by starting with one region, i.e. the entire plane; recursively, each region is subdivided with a dividing subplane (a line) until each region intersects at most one object from the scene. See figure 3.11 for an example of a scene showing one possible binary space partitioning. A tree, known as the *binary space partition tree* (BSP tree), is formed to keep a record of the splits made by the algorithm. Each internal node represents a splitting line used to decompose a region into two sections. If the splitting line contains any objects, they are stored in the internal node along with the splitting line. The children of an internal node represent the resultant regions after splitting the internal node. A leaf in the tree represents a region that intersects at most one object from the scene. The leaf stores the object that intersects the interior of the region.

There are two measures of the complexity of this problem. The first is the amount of time it takes to compute what the binary space partition for a scene of

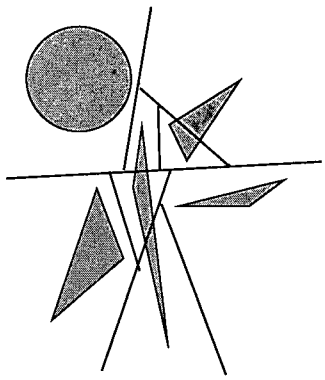


Figure 3.11: A binary space partition of a scene.

objects is. We will look at this later. The second issue concerns the size of the binary space partition. The size can be viewed as the number of regions in the partition, or the number of nodes in the BSP tree. Note that the number of regions is the number of leaves in the BSP tree. This means that the two measures are related by a factor of two. The number of nodes of the BSP tree, is related to the complexity of many of the applications of the BSP tree. Many applications of the BSP tree create the tree once, and use it over and over many times. Thus we want to ensure that the size of the BSP tree is small. We are not as concerned with keeping the time complexity to create the BSP tree small, although we would like to be able to compute such a tree in a reasonable time on most input scenes for the structure to be practical on large inputs.

3.5.2 Previous Results

Planar binary space partitions have been well studied in the literature. Paterson and Yao [15] proved that any set of n polygons in the plane admits a binary space partition of size $O(n \log n)$; if the polygons are all axis parallel then the scene admits

a size $O(n)$ or linear sized binary space partition. It is still unknown whether every scene in the plane admits a linear sized binary space partition.

There has been a number of papers classifying specific types of inputs for which linear sized binary space partitions can be constructed. De Berg et al. have shown that uncluttered scenes admit linear sized binary space partitions [7]. They give an algorithm to compute such a binary space partition that runs in time $O(n \log n)$ if the input is uncluttered. An obvious consequence of this is that scenes of fat objects and low density scenes admit linear sized binary space partitions. In de Berg et al. [8] they prove that the following inputs admit linear sized binary space partitions: scenes of n planar fat objects, scenes of n planar convex homothets (a set of objects is a set of homothets if all the objects are scaled and translated copies of one another) and scenes of n line segments with bounded ratio between the longest and shortest line segment (the size of the binary space partition is actually dependent on the ratio between the longest and shortest line segment in their development).

These results all describe a number of different types of planar scenes that admit linear sized binary space partitions. The gap between the best known general upper bound to the size of a binary space partition and the linear sized lower bound is better understood. We will show in the next subsection that scenes that have linear quadtree complexity have linear sized binary space partitions that can be computed in $O(n \log n)$ time. In fact we will show that the linear sized bound holds even if we relax the splitting criterion to allow a cell to have an unlimited number of non-silent intersections with objects that leave exactly one endpoint in the cell so long as there is not a single silent intersection (nor a single object entirely contained in the cell).

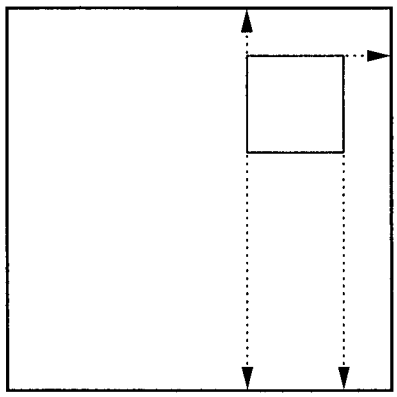


Figure 3.12: Arrows indicate the extension of a shrunken cell to the boundary of the parent to create valid binary space partition splits.

3.5.3 Relation to Quadtree Complexity

Lets assume that we have a planar scene $\mathcal{S}$ that consists of n non-intersecting (except possibly at the endpoints) line segments. We will denote the quadtree complexity of $\mathcal{S}$ as $QTC(\mathcal{S})$. Let the particular quadtree cover that leads to $QTC(\mathcal{S})$ be denoted as $QT(\mathcal{S})$.

The first thing to note is that $QT(\mathcal{S})$ only needs to be modified slightly to become the start of a valid binary space partition. The order in which splits occurred in the creation of $QT(\mathcal{S})$ can be followed to create the binary space partition, noting the following modifications. Each binary split represents three splits of a region in the current binary space partition: a vertical one and two horizontal ones (or vice versa). A shrinking split can also be viewed as up to four splits of a binary space partition: one split along each of the boundaries of the shrunken cell extending as far as each can go, not crossing boundaries. See figure 3.12 for an example. Less than four splits will be made if the shrunken cell is adjacent to any of the boundaries.

The quadtree decomposition of the scene with the specified modifications

are the start of a binary space partition. At this point, we must determine the set of line segments that intersect each cell in the intermediate BSP. We know, from section 3.3.4 that these can be computed in $O(n \log n + QTC(\mathcal{S}))$ time. The next step in building the BSP is to make splits along any silent intersections between line segments and the resultant regions in the BSP. To determine the number of cells that have been created up to this point, we focus on the cells that contain a shrunken cell. The reason for this is that meshing the shrunken cell into the BSP has the effect of creating more divisions than are present in the original quadtree decomposition; these extra cells can increase the number of regions in the scene beyond the quadtree complexity of the scene. Binary splits do not require extra splits to insert them into the BSP, and therefore do not increase the number of regions beyond the quadtree complexity.

Let P represent a cell that contains a shrunken cell. P is divided into four subcells (not counting the shrunken cell, and not looking at any silent intersections). It is possible for each of the silent intersections with P to intersect all four of the subcells. If this happens, then each silent intersection will create four additional cells inside P. The maximum number of cells in P is then $\#C(P) = 4(wt(P))$, where $\#C(P)$ represents the number of cells in P and $wt(P)$ represents the weight associated with P. These parent cells present the worst case possible increase in the number of cells. The total number of cells in the intermediate binary space partition, labeled BSP', is thus

$$\#C(BSP') \leq \sum_{c \in QT(\mathcal{S})} (4wt(c)) = 4QTC(\mathcal{S})$$

To complete the BSP, we must deal with the remaining intersections between line segments and cells of BSP'. These are the non-silent intersections. We know that there will not be more than δ such non-silent intersections in any given cell

of BSP'. Partitioning these δ objects into a valid BSP requires a subtree of size at most $O(\delta \log(\delta))$. This represents a constant increase in the overall size of the BSP. If every cell in BSP' requires $O(\delta \log(\delta))$ splits, we get that the final BSP has $O(\delta \log(\delta))4QTC(\mathcal{S}) = dQTC(\mathcal{S})$ cells, where $d = 4O(\delta \log(\delta))$ is a constant. All of the above work for translating a quadtree decomposition of a scene into a valid BSP takes a constant amount of additional time per cell in the quadtree.

Altogether we have proven the following lemma.

Lemma 22 *Given a planar scene $\mathcal{S}$ consisting of n disjoint (except possibly at the endpoints) line segments, we can construct a binary space partition for $\mathcal{S}$ with at most $d \cdot QTC(\mathcal{S})$ cells, where d is a constant. If $QTC(\mathcal{S})$ is linear in n, then the size of the binary space partition is linear.*

The complexity of the BSP algorithm on a scene $\mathcal{S}$ is $O(n \log n + QTC(\mathcal{S}))$. Since the algorithm has two parts, there are two contributors to the complexity of the algorithm. The first part of the algorithm requires the creation of a quadtree over the scene. As we have seen in section 3.3.4, the quadtree can be computed in time $\Theta(n \log n)$. Determining which line segments intersect which cell in the decomposition takes $\Theta(n \log n + QTC(\mathcal{S}))$ time. The remaining work for this computation is the conversion of the quadtree into a binary space partition. As we have seen above, this can be done in a constant amount of additional time per cell in the resulting quadtree decomposition. Putting this with the above lemma proves the following.

Theorem 23 *Let $\mathcal{S}$ be a planar scene with n disjoint (except possibly at the endpoints) objects. We can construct a BSP for $\mathcal{S}$ with size $O(QTC(\mathcal{S}))$ in time $O(n \log n + QTC(\mathcal{S}))$. If the quadtree complexity of $\mathcal{S}$ is in $O(n)$, then the binary*

space partition has linear size and is computed in $O(n \log n)$ time.

If we are interested only in creating a valid binary space partition for a scene, we can in fact relax the splitting criterion used in deciding when a cell should be split. De Berg et al., in [8], have proven the following theorem:

Theorem 24 (de Berg et al. [8]) *Let $\mathcal{C}$ be a convex polygon and let $\mathcal{S}$ be a set of line segments inside $\mathcal{C}$ that are all anchored at the boundary of $\mathcal{C}$. Then there exists a binary space partition for $\mathcal{S}(\mathcal{C})$ (the set of segments anchored at the boundary of $\mathcal{C}$) inside $\mathcal{C}$ of size at most $\frac{3|\mathcal{S}(\mathcal{C})|-1}{2}$.*

This theorem shows that any convex polygon intersected by line segments in such a way that each line segment crosses the boundary at least once, can be split a linear number of times to form a binary space partition. We can thus change the splitting criterion to the following: *split a cell only if it contains too many segments.* If the cell contains no segments, then the cell need not be further split, as a linear sized binary space partition can be formed for the scene inside the cell using the theorem. This simplifies the splitting process, resulting in a lowering of the size of most BSPs constructed in this fashion. It has the effect of increasing the number of scenes which can be proven to produce a BSP of linear size.

Interestingly, this modified quadtree algorithm gives a linear sized binary space partition to the scene consisting of n parallel lines, that was discussed in section 3.4. This in itself is not surprising, but does indicate an improvement over the quadtree algorithm used in determining quadtree complexity for a scene. Under the original algorithm the scene has $\Theta(n \log n)$ quadtree complexity, and hence will produce a BSP of size $\Theta(n \log n)$. An improvement to linearity in the size of the BSP for this scene by a quadtree based method is admirable. This modified quadtree

algorithm may be useful in other applications as well.

Chapter 4

Conclusions

We have looked at a number of different models of two dimensional scene complexity. The goal of each model is to measure the intrinsic complexity of geometric scenes. Different models look for different properties to deem the scene simple or complex. The first model, fatness, looks at each object in a scene, setting the fatness of the scene to the fatness of the least fat object. If the fatness of the least fat object is less than a fixed constant, then the scene is deemed complicated. The density model measures the maximum number of large objects that intersect a similarly sized region. If there is a region that intersects more than a fixed constant number of objects of the same or larger size, then the scene is deemed complicated.

We then looked at two models that focus on how easy it is to approximate the distribution of objects throughout the scene with a set of points. The guarding set model counts the smallest number of points needed to provide a good approximation of the distribution of the objects; if this number of points is linear in the number of objects, then the scene is deemed simple. The clutteredness model deems a scene simple if a particular set of points, namely the bounding box corners of the objects

in the scene, provide a good approximation for the set of objects.

The remaining models looked at are cover-complexity models. These models attempt to cover the scene with a set of regions, where the complexity of scene intersecting each region is somehow limited. The simple-cover-complexity model limited the covering regions to not intersect more than a constant number of objects from the scene. In this model, the number of discs needed to cover the scene is the measure of complexity; if a linear number of discs can cover the scene, then the model considers the scene to be simple. The expanded and unexpanded disc complexity models both cover the scene with discs; the quadtree complexity model covers the scene a quadtree decomposition. These models restrict the number of objects that start or end in a region of the cover. In these models, each covering region is assigned a weight that is equal to the number of objects that intersect the region silently (without leaving an endpoint). The complexity of the scene is the sum of the weights of all the regions in the cover.

The last three models of unexpanded disc complexity, expanded disc complexity and quadtree complexity were introduced by this thesis. The reason these models were introduced was to capture a different idea of what it means for a scene to be complex. It seems that there are many scenes with a number of sections of almost parallel lines in them. Some algorithms are able to efficiently process such scenes, even though the worst case complexity based on the input size indicates that such processing would be inefficient. These new models permit a small number of sections of almost parallel lines to exist in a scene, without assigning a large complexity to the scene. Only the starting and ending points of such lines are charged for. In these weighted models, a scene is complicated if there are a number of regions of parallel lines around which objects terminate.

The models described in this thesis have been shown to form a hierarchy. At the bottom of the hierarchy is the model of fatness, and at the top of the hierarchy is the model of quadtree complexity. In between are the models of density, clutteredness, the equivalent models simple-cover-complexity and guarding sets, followed by expanded disc complexity. It is not known exactly where unexpanded disc complexity fits in the hierarchy, except that it is more general than the expanded disc complexity model. The property of the hierarchy is that a scene deemed simple by a particular model will be considered simple by all models above it in the hierarchy; it may not be the case that the scene is considered simple in models below it in the hierarchy. The usefulness of this hierarchy is to aid in the analysis and design of algorithms. When solving a particular geometric problem, the hierarchy can be looked at to determine the most general model that can be used to aid in the computation of the solution. Some problems may be easier to solve when provided simple scenes according to a particular model as input. If none of the models in the hierarchy seem to fit a particular problem, perhaps a similar model to the ones described here can be created with the needed properties.

We looked at one problem in particular, namely the binary space partition problem, and noted that scenes with linear quadtree complexity have linear sized binary space partitions. The binary space partition can be computed using this method in time $O(n \log n + QTC(\mathcal{S}))$. For scenes with superlinear quadtree complexities, the size of the BSP computed with the described method will be proportional to the quadtree complexity of the scene.

We restricted our focus to the two dimensional case as it is difficult to extend the weighted models into higher dimensions. The problem lies in how object representatives are defined. The natural choice for an object representative is a facet

of dimension $d-2$; that is a point in two dimensions or a line segment in three dimensions. But dividing a three dimensional cell based on the presence of too many line segments will result in more than a linear number of cells (in the number of line segments) to be created, as each split of a line segment will result in more than one subcell containing the line segment. This does not occur for points.

In short, this work has looked at some specific properties that can be related to the complexity of some geometric algorithms. It is hoped that these properties will enable greater understanding to be achieved about the role played by the structure, as opposed to the size, of an input scene in the complexity of algorithms that process the scene.

4.1 Possible Future Work

Some questions remain open from this thesis. It is not known how the unexpanded disc complexity model relates to the quadtree complexity model. As we have shown, there is an example of a scene that has low unexpanded disc complexity, but high quadtree complexity. However, we have no proof that every scene with linear quadtree complexity will have linear unexpanded disc complexity. If such a proof could be found, then the unexpanded disc complexity model would move in to the top of the hierarchy of models. If a family of scenes is found with high unexpanded disc complexity and low quadtree complexity, then the disc complexity model will still be more general than the expanded disc complexity model, but will be unrelated to the quadtree complexity model, created a fork like structure at the top of the hierarchy.

The optimization to the quadtree algorithm presented for computing binary space partitions can be further studied and analyzed. The idea was to split a cell

only if there are objects wholly contained in the cell. This idea may extend more readily to higher dimensions that the ideas presented in this thesis.

The complexity of various existing solutions to geometric problems can be related to the models presented here. Additionally new solutions to geometric problems can be designed with lower complexity when given simple scenes as input, and higher complexities when given complicated scenes as input. When performing either of these tasks, it is advantageous to analyze the algorithm under the most general model that fits the problem. This will provide the largest selection of simple scenes for the algorithm. However, choosing a model that is too general may cause the associated time or space complexity bound for the algorithm to be larger than needed.

Bibliography

[1] P. Agarwal, M. Katz, and M. Sharir. Computing depth orders for fat objects and related problems. *Computational Geometry Theory and Applications*, 5:187–206, 1995.

[2] H. Alt, R. Fleischer, M. Kaufmann, K. Mehlhorn, S. Naher, S. Schirra, and C. Uhrig. Approximate motion planning and the complexity of the boundary of the union of simple geometric figures. *Algorithmica*, 8:391–402, 1992.

[3] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the Association for Computing Machinery*, 42(1):67–90, January 1995.

[4] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, LNCS 510, pages 661–673. Springer-Verlag, 1991.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* McGraw-Hill, 1994.

[6] M. de Berg. Linear size binary space partitions for fat objects. In *Proceedings of the 3rd Annual European Symposium on Algorithms (ESA '95)*, LNCS 979, pages 252–263. Springer-Verlag, 1995.

[7] M. de Berg. Linear size binary space partitions for uncluttered scenes. Technical Report UU-CS-1998-12, Department of Computer Science, Utrecht University, Utrecht, the Netherlands, 1998.

[8] M. de Berg, M. de Groot, and M. Overmars. New results on binary space partitions in the plane. *Computational Geometry Theory and Applications*, 8:317–333, 1997.

[9] M. de Berg, M. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. In *Proceedings of the 13th ACM Symposium on Computational Geometry*, pages 294–303, 1997.

[10] M. de Berg, M. J. Katz, M. Overmars, A. F. van der Stappen, and J. Vleugels. Models and motion planning. In *Proceedings of the 6th Scandinavian Workshop on Algorithm Theory*, LNCS 1432, pages 83–94. Springer-Verlag, 1998.

[11] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, Jun 1981.

[12] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *Journal of Algorithms*, 18:403–431, 1995.

[13] J. Matousek, N. Miller, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. In *Proceedings of the 32nd Annual IEEE Symposium on the Foundations of Computer Science*, pages 49–58, 1991.

[14] J. S. B. Mitchell, D. M. Mount, and S. Suri. Query-sensitive ray shooting. In *Proceedings of the 10th ACM Symposium on Computational Geometry*, pages 359–368, 1994.

[15] M. S. Paterson and F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Computational Geometry*, 5:485–503, 1990.

[16] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete and Computational Geometry*, 4(2):101–115, 1989.

[17] A. F. van der Stappen. *Motion Planning Amidst Fat Obstacles.* PhD thesis, Deptartment of Computer Science, Utrecht University, Utrecht, the Netherlands, October 1994.

[18] A. F. van der Stappen, M. H. Overmars, M. de Berg, and J. Vleugels. Motion planning in environments with low obstacle density. Technical Report UU-CS-1997-19, Department of Computer Science, Utrecht University, Utrecht, the Netherlands, 1997.

[19] M. van Krevald. On fat partioning, fat covering and the union size of polygons. In *Proceedings of the 3rd Workshop in Algorithms and Data Structures*, LNCS 709, pages 452–463. Springer-Verlag, 1993.

[20] J. Vleugels. *On Fatness and Fitness - Realistic Input Models for Geometric Algorithms.* PhD thesis, Deptartment of Computer Science, Utrecht University, Utrecht, the Netherlands, March 1997.