Optimal cell size for efficient retrieval of sparse data by approximate 2D position using a coarse spatial array

Leslie John Kitchen
Michael Callahan

COINS Technical Report 86-26

June 1985

Optimal cell size for efficient retrieval of sparse data by approximate 2D position using a coarse spatial array

Leslie John Kitchen
Michael Callahan
Computer and Information Science Department
University of Massachusetts
Amherst, MA 01003

Abstract

Many applications require the retrieval of data by approximate position: geographical databases, and in computer vision such processes as perceptual grouping, matching, and hypothesis verification. A standard method for doing this retrieval efficiently is to store the data in lists in a coarse spatial array, so only the few array cells near a query position need be searched. Under reasonable assumptions, we characterize this problem in terms of its parameters, and find analytically the cell size that gives fastest expected retrieval time. The method is shown to be robust, in that, near the optimum, the expected retrieval time is not sensitive to the problem parameters, particularly for sparse data.

This work was supported by the Air Force Office of Scientific Research under grant number AFOSR-86-0021, and by the National Science Foundation under grant number DCR-8318776.

1 INTRODUCTION

Commonly in computer vision, it is necessary to retrieve data objects efficiently by their approximate spatial position in an image. These data objects might possibly be image features to be grouped together partly on the basis of proximity in a mid-level perceptual-organization process. In a high-level, top-down process of matching or hypothesis verification, predictions to be tested will often be in terms of the approximate spatial position of perceptual tokens or hypotheses. Similar problems can arise in geographic databases; they are all special cases of *region searching*, as defined by Knuth [1973]. Henceforth the data objects to be retrieved will be referred to merely as *points*, since it is only their position that concerns us here, not other information that may be associated with them in particular applications.

Often this retrieval is done by laying a coarse 2D array over the image space. Each cell of this array corresponds to a rectangular block of the image. Points located in a block are stored in a list in the corresponding cell. To retrieve points lying within a certain locus, first all blocks which intersect this locus are determined, and the lists of points in the corresponding cells are scanned to determine whether any of these candidate points lie within the given locus.

This method can have very poor worst-case performance (when points are clumped so that almost all of them lie in a single block, and all query loci overlap this block, or when the shape of query loci is greatly different

from that of a block). This has led many people to use hierarchical structures, such as quadtrees and k-d trees, for such problems. (See [Bentley and Friedman 1979] and [Samet 1984].) However, for many practical problems (particularly those for which points and queries are reasonably uniformly distributed), this method of coarse arrays is the simplest and fastest in expected retrieval time. Under a reasonable set of assumptions, we determine what block size for the coarse array gives shortest expected retrieval time.

2 SETTING UP THE PROBLEM

Suppose that we have P points distributed in a square image space, N length units on a side. These length units are arbitrary, but would typically be the size of a pixel in the underlying digital image. Suppose further that for a query at a given position we would like to retrieve all points that lie within a certain radius r of that position. (This radius r characterizes our uncertainty or tolerance about the actual position of points.)

We speed up retrieval by storing the points in lists in a coarse array whose cells correspond to square blocks of the image, each block being n length units on a side. These cells will be called *bixels* (for "big pixels"). Where no confusion can arise, a bixel will be identified with its corresponding square block in the image space. The geometric setup can be seen in Figure 1. Say that it takes a time units to access a bixel (largely indexing a cell in the coarse array), and that it takes b time units to process a point for

retrieval (largely determining whether it lies within r of the given position, that is, computing a sum of squares of coordinate differences, and comparing against an already squared radius).

For a given query, we draw a circle around the given position, and find all bixels that might intersect the disk of this circle. How many such bixels are there? Consider first the corresponding problem in one dimension, and assume for the purposes of exposition that $n = 1$, or alternatively, that all distances are measured in units of the length of the side of a bixel. Visualize the number line, with the integer positions marked, and take an arbitrary position x on the line. To find all points that lie within distance r of x, we have to scan the unit intervals that lie between $\lfloor x - r \rfloor$ and $\lceil x + r \rceil$. (See Figure 2.) While these *floor* and *ceiling* operators may seem to make the analysis intractable, it is fairly easy to see that if the position x is uniformly distributed within the unit intervals then the expected number of unit intervals that must be scanned is actually $1 + 2r/n$.

Now consider the problem in two dimensions. The straightforward extension of the 1D analysis would give us a rectangle of bixels to search. (It might be thought that we would get a square of bixels, but for certain positions the different behavior of *floor* and *ceiling* in the two dimensions may make the width differ from the height. As an illustration, consider a search disk whose diameter is slightly greater than the bixel size. If the search position is at the center of a bixel, the disk will overlap three bixels in each

dimension, but if it is moved slightly to the left it will overlap only two bixels in the horizontal dimension.) Again, assuming that the query position is uniformly distributed, the expected number of bixels in the rectangle is

$$(1 + 2r/n)^2 \tag{1}$$

This rectangle of bixels is actually too big. First, there may be bixels towards the corners of the rectangle that do not at all intersect the disk of radius r about the query point. These bixels need not be scanned, since no point inside them could satisfy the query. Second, there may be bixels that lie entirely within the search disk. Points inside these bixels need not be tested individually, since they must all satisfy the query. This suggests that for each bixel in the rectangle we determine whether it lies entirely within, entirely without, or only partly within the search disk. Only in the last case would we need to test the points inside the bixel one by one.

However, in order to simplify the retrieval algorithm and its analysis, we assume that all bixels in this rectangle are scanned for points. Later it will be shown that this assumption is quite justified, in that for all reasonable parameter settings the selective processing of bixels would actually degrade performance.

So, for each bixel in the rectangle, we scan its list of points to determine which of them lie within r of the given position. Now, the average density of points in the image space is $\rho = P/N^2$, so the average number of points

contained in a bixel is

$$\rho n^2 \tag{2}$$

and the average time to process a bixel is $a + b\rho n^2$, or $a + \beta n^2$, where $\beta = b\rho$ can be regarded as the spatial density of point-processing effort. Notice that the absolute number of points P and the absolute area of the image N^2 drop out; all that matters really is the spatial density of points.

Putting these together, we have that for each query we must scan $(1 + 2r/n)^2$ bixels on average, and each bixel requires $a + \beta n^2$ expected computation time. Therefore, the expected time to answer a query is

$$(1 + 2r/n)^2(a + \beta n^2) \tag{3}$$

Another index of the efficiency of the retrieval, called here the *redundancy*, can be derived as follows. As shown above, the average number of bixels examined for each retrieval is $(1 + 2r/n)^2$. Each of these bixels can be expected to contain ρn^2 points. So each retrieval will examine

$$\rho n^2(1 + 2r/n)^2 = \rho(n + 2r)^2$$

points. Now, since the search disk has area πr^2, we can expect that it will contain, on average $\rho\pi r^2$ points. Therefore, the retrieval will look at too many points, by a factor of the ratio of these two expressions, namely

$$\frac{\rho(n + 2r)^2}{\rho\pi r^2} = \frac{1}{\pi}\left(\frac{n}{r} + 2\right)^2 \tag{4}$$

which approaches its smallest, and best value of $4/\pi = 1.2732\ldots$, as n approaches zero. (Obviously, $4/\pi$ is the ratio of the area of a square to the area of its inscribed circle.) For convenience, we will also define the *relative redundancy*, as the ratio of the redundancy to its smallest possible value, that is,

$$\frac{1}{4}\left(\frac{n}{r}+2\right)^2 \tag{5}$$

The redundancy tells only half the story about the retrieval process. It is at its best when the bixels are very small, but this ignores the cost of accessing many empty bixels. However, in conjunction with the expected number of bixels scanned (from (1)), it gives a better idea of what trade-offs are going on than the composite retrieval time, which takes into account both these contributions, modified by the implementation-dependent parameters a and b.

3 FINDING THE OPTIMAL BIXEL SIZE

Intuitively, it is clear that there must be some value for n, the bixel size, which minimizes the query time. Consider extreme values of n. If bixels are very large compared with r, then a query will examine usually only one bixel (four at most), but these bixels will contain many irrelevant points that are far from the query position. If bixels are very small compared with r, a query will mostly look only at relevant points, but at the expense of

scanning many empty bixels. So somewhere in between these extremes there must lie an optimum value for n. We find this minimum by the standard method of finding the zeroes of the derivative of expected query time with respect to n.

Expanding the expression for query time yields

$$a + \beta n^2 + \frac{4ar}{n} + 4\beta rn + \frac{4ar^2}{n^2} + 4\beta r^2$$

Differentiating with respect to n yields

$$2\beta n - \frac{4ar}{n^2} + 4\beta r - \frac{8ar^2}{n^3}$$

Equating this derivative to zero, and multiplying by $n^3/(2\beta)$ gives the quartic equation

$$n^4 + 2rn^3 - 2\gamma rn - 4\gamma r^2 = 0$$

where $\gamma = a/\beta$.

As it turns out, this quartic has only one positive root, namely

$$n = \sqrt[3]{2\gamma r} = \sqrt[3]{\frac{2aN^2r}{bP}} = \sqrt[3]{\frac{2ar}{b\rho}} \tag{6}$$

which is therefore the bixel size that minimizes query time. The second derivative is

$$2\beta + \frac{8ar}{n^3} + \frac{24ar^2}{n^4}$$

or, at the minimum

$$6\left(\beta + \sqrt[3]{\frac{4\beta^4 r^2}{a}}\right) \tag{7}$$

which is obviously positive, and small if points are sparse. Thus, small changes from the optimum bixel size will only slightly increase the retrieval time.

4 AN EXAMPLE

Suppose that we have a 256-by-256 image containing 100 feature points, that we wish to make queries with a radius of 10 pixel units, that it takes 1 time unit to index into a bixel, and 16 time units to check whether a point lies within the query radius. (In our notation, $N = 256$, $P = 100$, $r = 10$, $a = 1$, $b = 16$, from which $\rho = P/N^2 = 0.00153$, $\beta = b\rho = 0.0244$, $\gamma = a/\beta = 40.96$.) Then by (6) the bixels should ideally be 9.4 by 9.4 pixel units in size for fastest retrieval, giving an expected computation cost of 30.9 time units (by (3)).

This computation time represents scanning, on average, 9.8 bixels per query (1), and processing an expected number of 0.13 points in each (2), or 1.32 points per query. By (4) the redundancy is 2.74, and by (5) the relative redundancy 2.15. That is, at this optimum bixel size, we access 9.8 times more bixels than in the one extreme ideal case (of just a single bixel), and examine 2.15 times more points than in the other extreme ideal case

(of vanishingly small bixels, which contain at most one point each). But it is this trade-off that gives the best compromise retrieval time.

The second derivative is quite small here (0.46 by (7)), so that choosing a more convenient bixel size, such as 8 by 8, or 16 by 16, does not have any undue cost, these sizes having computation costs of 31.4 and 36.7, respectively. The extreme cases, of storing all the points in a single list and of making bixels as small as pixels, have expected costs per query of 1601 and 451.8, respectively.

5 DISCUSSION

We discuss here some further issues related to this analysis and the assumptions behind it. We also consider some other aspects of this problem, and its connections with other work.

A striking aspect of the results presented here is the simplicity and elegance of the formula for the optimum bixel size (6). From this it is easy to understand the sensitivity of the optimum to changes in the parameters. Important from a practical point of view is that the cube root ameliorates the effect of relative errors in the problem parameters. Also, the retrieval time (3) varies only slightly for small changes from the optimum (particularly if points are sparse), further reducing the effects of such errors, and also making it safe to choose a nearby, convenient integral bixel size instead of the exact optimum. These observations do not mean that this analysis

can be ignored; rather that the method is robust and predictable.

The treatment above quietly ignored the matter of edge effects: for positions sufficiently close to the border of an image, the query disk will extend beyond the actual image data. (See Figure 3.) Put another way, the analysis implicitly assumes that the size of the image is large enough in comparison with the search radius that these edge effects are negligible. That is why the derivation can be done in terms only of the *density* of points. Since the edge effects can be regarded as causing a small mis-estimation of this density, the observations about sensitivity above imply that the edge effects can indeed be safely ignored.

The derivation of optimal bixel size is couched in terms of continuous image space. This may seem inconsistent with the discrete nature of digital images, but causes no real problem. First, even when points are derived from digital images and so are ascribed only to positions on a discrete grid, we can still imagine them as embedded in a continuous image space. Having non-integral bixel sizes causes no conceptual difficulties here. The only effect of the digitization of point positions is to introduce local inhomogeneities into the continuous distribution of point positions, in that points can appear only at grid positions. These inhomogeneities will be significant only when the bixel size is comparable to the underlying pixel size, or smaller: something that will not arise in practice. Even if they were significant, the periodicity of the grid and the overall symmetry of the structure would ensure that

these small-scale inhomogeneities were evened out in the long run, leading to the same analysis for expected retrieval time.

Fundamentally, this bixel method has only two essential parameters: the ratio of the bixel-access time to the point-processing time, and the point density. The effects of the other parameters can be accounted for merely by scale changes. This can be best shown by choosing suitable units for measurement. If all lengths are measured in units of the search radius r (this implies also that the density ρ is measured in points per r^2 area), and all times are measured in units of the point-processing time b, then the formula for expected retrieval time becomes simply

$$\left(1+\frac{2}{n}\right)^2(a+\rho n^2)$$

and the optimum bixel size (relative to the search radius) becomes

$$n=\sqrt[3]{2a/\rho} \tag{8}$$

While this dimensionless formulation is not quite so convenient in practice, it brings out the underlying properties of the method. In particular it makes it easy to see that, for reasonable cases, the optimum bixel size will be larger than the search radius. Under this formulation, the area of the query disk is π, so a query will on average retrieve $k=\pi\rho$ points. From (8), it is easy to see that the optimum bixel size will be greater than the search radius when $k<2\pi a$. The bixel access and the point checking are roughly

similar computations, so a (in this formulation) will be a ratio near unity. (The ratio of 1/16 used in the example was based on a syntactic count of function calls in interpreted Lisp code, and is not really representative of true relative computational cost.) Thus, so long as we do not expect to retrieve a large number of points per query in a given application, we can conclude that the optimum bixel size will indeed be greater than the search radius. Even when this density constraint is violated, the cube root in (8) keeps the effect small. For instance, it would take an eightfold increase in density to halve the optimum bixel size.

This observation confirms our assumption that when scanning the rectangle of bixels it is not worthwhile to test for and specially treat bixels that lie entirely inside or entirely outside the search disk. Since the bixels will not be small compared to the search disk, fully interior or exterior bixels will occur infrequently, if at all. Even when such bixels occur, the expected number of points in each, namely

$$\sqrt[3]{(2a\rho)^2} \tag{9}$$

would typically be small enough that the cost of testing the disposition of the bixel would be greater than the cost of testing the points inside it directly. (In any particular instance these conditions can be checked in order to verify that the assumptions hold.)

The analysis also made certain other assumptions about the parameters.

It was assumed, quite reasonably, that the computation times for bixel access and for point checking are fixed ahead of time, and, less reasonably, that the query radius r is likewise fixed. It was also assumed that both points and query positions are uniformly distributed.

Actually, these assumptions are overly restrictive: most of them can be relaxed. For example, the same treatment for the expected retrieval time obviously carries over to the case where the search radius varies, merely by using its expected value for r, provided that the distribution of query radii is independent of any inhomogeneities in the distributions of point and query positions. (However, if the point distribution were significantly non-uniform, then some other method, such as k-d trees or point quadtrees, would be more appropriate.) Similarly, the same analysis can be performed in terms of the expected values of other varying parameters, so long as there are no significant interactions between them. Properly accounting for significant interactions would be difficult, and probably of little practical use in general.

Two issues that have not been addressed are the setup time required to build the bixel array and fill it with points, and its storage requirements. These should not be significant, since they would normally be dominated by the time and space required to process and store the original image.

However, in some applications it might be desirable to minimize the amount of storage space taken up by the bixel array. Bentley and Friedman

[1979] suggest using a hash-table for the bixel array under these circumstances. This would seem to be advantageous, since even the bixel array will tend to be sparse (that is, most bixels will contain an empty list of points), but it leaves unaddressed the questions of choosing a hashing method and table size. It is also possible to retain the bixels in a 2D array (particularly since the data points are assumed uniformly distributed), and explicitly minimize the size of this array, subject to some constraint on acceptable retrieval time. This requires finding the largest positive solution for n of the quartic equation:

$$n^4 + 4rn^3 + (4r^2 + \gamma - c)n^2 + 4\gamma rn + 4\gamma r^2 = 0$$

where c is β times the desired retrieval time. The analytic solution of this equation is an obvious goal.

All the above stem from further analysis and refinement of this same storage scheme for sparse points. It would be of great interest to compare this scheme, both for retrieval time and storage requirements, with other schemes usable for the same purpose, such as those described in [Bentley and Friedman 1979] and [Samet 1984] (particularly k-d trees and point quadtrees), in order to discover the conditions under which one scheme might be preferable to others. The analyses by Bentley and Stanat [1975] of point quadtrees and that by Eastman [1981] of k-d trees, while applied to slightly different problems, would be good starting points for such a detailed comparison.

A couple of observations can be made even now. Our demonstration that

the optimum bixel size will be comparable to the search radius is consistent with remarks in [Bentley et al. 1977] and [Bentley and Friedman 1979] that, for the related problem of range searching, it is close to optimal for the cells in the coarse array to be the same size and shape as the rectangular search locus. However, this cited work seems not to take into account the dependence on relative computation times and point density. Both Bentley and Friedman [1979] and Samet [1984] assert that a coarse array is not a suitable data structure if the range of searches varies greatly. The results here suggest a qualification of this, that a coarse array will handle varying ranges gracefully, provided the shape of the query locus does not change much. For the radius searches presented here, retrieval time grows with the area of the search disk, which is proportional to the number of points retrieved. In this respect, bixel arrays have the same behavior as k-d trees and quadtrees.

6 CONCLUSION

We have shown how to determine the best bixel size for storing points for fast retrieval. This optimum bixel size is simple to compute, and moreover the resulting expected retrieval time is reasonably insensitive to the problem parameters. So we can choose a convenient bixel size, confident that it will give close to optimum performance even if some of the problem parameters are not accurately known. The assumptions of the analysis are valid over

a broad range of problems, and can easily be verified if necessary. These results illustrate the understanding that can be provided by an analytic, as opposed to merely numeric or empirical, solution to a problem.

7 REFERENCES

J. L. Bentley and J. H. Friedman, "Data structures for range searching", *Computing Surveys*, vol. 11, no. 4, December 1979, pp. 397–409.

J. L. Bentley, D. F. Stanat and E. H. Williams, Jr, "The complexity of finding fixed-radius near neighbors", *Information Processing Letters*, vol. 6, no. 6, December 1977, pp. 209–212.

J. L. Bentley and D. F. Stanat, "Analysis of range searches in quadtrees", *Information Processing Letters*, vol. 3, no. 6, July 1975, pp. 170–173.

C. M. Eastman, "Optimal bucket size for nearest neighbor searching in k-d trees", *Information Processing Letters*, vol. 12., no. 4, August 1981, pp. 165–167.

D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.

H. Samet, "The quadtree and related hierarchical data structures", *Computing Surveys*, vol. 16, no. 2, June 1984, pp. 187–260.

8 FIGURES

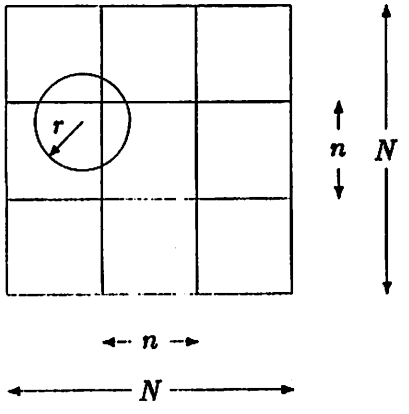

Figure 1: Geometric setup for the coarse array and search disk.

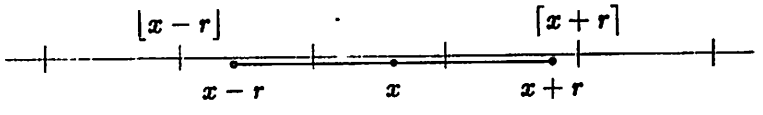

Figure 2: Intersection of search range with cells in one dimension.

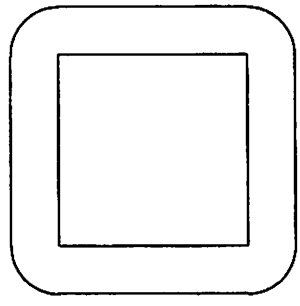

Figure 3: Search disks extend beyond borders of image.