Declustering Objects for Visualization*

Ling Tony Chen Doron Rotem†

Lawrence Berkeley Laboratory
Berkeley, CA 94720

Abstract

In this paper we propose a new declustering method which is particularly suitable for image and cartographic databases used for visualization. Our declustering method is based on algebraic techniques using vectors. The algorithm which computes the disk assignment requires $O(K^{\frac{3}{2}} \log K)$ time where K is the number of parallel disks in the system. The resulting disk assignment maximizes the area that can be visualized without accessing any disk more than once. The method is easy to implement and works for any number of parallel disks. Our mathematical analysis show that for common visualization queries our declustering method performs within seven percent from optimal for a wide range of practical multiple disk configurations.

1 Introduction

Declustering files across multiple disk units is a well known technique for enhancing I/O parallelism. The idea is to partition the file among K disks such that the work involved in retrieving the answer to a query is balanced among the disks, i.e., retrieving an answer of B bytes will involve accessing approximately $\frac{B}{K}$ bytes from each disk.

Declustering of large files on multiple disk units has received great attention recently. In [2] it was shown that declustering leads to significant savings in response times for single attribute partitioned files. Declustering for range queries is presented in [5] where it is proposed to use a Hilbert curve linearization algorithm and then assign records to disks in a round-robin fashion. Declustering for partial match queries using Cartesian Product Files is described in [10, 4, 3]. In [6] declustering methods using error-correcting codes are developed.

In this paper we present and analyze a declustering method which is particularly suitable for image and cartographic databases often used for visualization in GIS and other scientific applications. In such applications the database contains a large collection of aerial or satellite photographs or maps which cover some area of interest. Users of such databases are typically interested in visualizing specific areas around points of interest. For example an area of radius r around some pollution source or epicenter of an earthquake. Other types of access may require visualizing a scenery along a path of a moving airplane or land vehicle. An excellent source of information about such databases and their use in various applications is given in the report of a recent NSF workshop on Visual Information Management Systems [9].

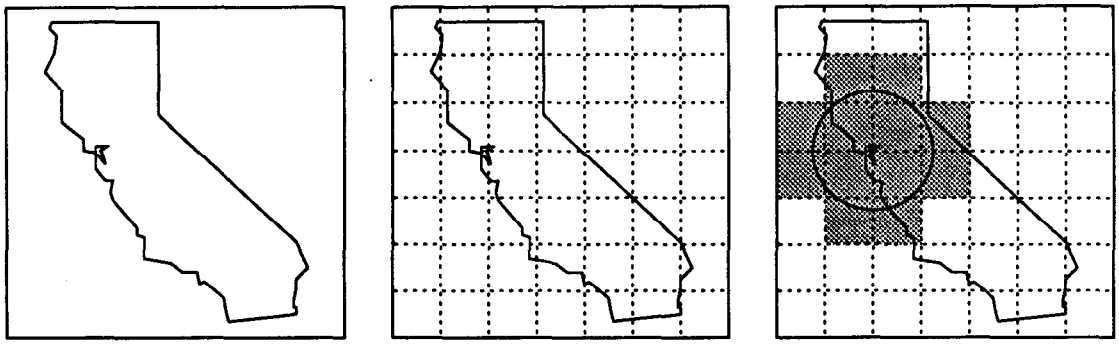
The original maps or satellite images can be quite large. For example a high resolution color aerial photograph of a 20 by 20 km region at 1 meter per pixel may require over 1.6 gigabytes of disk storage. For that reason, the original image is further divided into tiles of regular size which are stored on disk as atomic files as shown in Figure 1(b). Typical tile sizes range from 16 KB to 1 MB and depend on the storage device and query characteristics used by the application.

An answer for visualization queries requires accessing a set of image tiles that overlap circles of some radius (See Figure 1(c)). As the number of tiles needed to answer a query may be large, it is desirable to decluster them among the disks to optimize response time. Furthermore, as explained in the next section, restrictions on performance may require that each disk

*The support of the Defense Advanced Research Projects Agency, as well as the support of the Department of Energy under contract DE-AC03-76SF00098 is gratefully acknowledged.

†Also with Department of MIS, School of Business, San Jose State University

**Proceedings of the 19th VLDB Conference
Dublin, Ireland 1993**



(a) (b) (c)

Figure 1: (a) An image (b) Tiling of an image (c) Tiles overlapping a fixed radius query

will be hit at most once per each query.

In this paper we present a new class of declustering methods which we call *vector based declustering*. We then present an $O(K^{\frac{3}{2}} \log K)$ algorithm that finds a member of this class with final declustering results which are within a few percent from optimal. One difference between the problem we are considering here and other declustering methods such as the one described by [5] is that we require worst case performance guarantees rather than methods which behave well on the average. These guarantees are needed for applications that need to visualize a scenery in real-time. For example, we can show that in the Hilbert curve linearization scheme, some very small radius queries may hit the same disk twice for any value of K which is not acceptable in our applications. Another difference is that we are interested in fixed radius queries rather than arbitrary shaped orthogonal range queries.

It is interesting to note that a closely related problem was considered in [1] for the purpose of displaying raster graphics images stored on M parallel memory modules. The problem there was that of placing pixels on memory modules such that no two pixels come from the same memory module when displaying rectangular windows of some fixed area. The approach of that work is not suitable for our purposes as it assumes that M is a very large number, and the solution only works for a specific set of values of M.

The paper is organized as follows. In Section 2 we present some preliminaries and a more precise formulation of the problem. In Section 3 we present our algorithm for finding an optimal declustering scheme. In Section 4 we discuss practical implications of our results to the physical design of visualization database systems. In Section 5 we present our conclusions and present some further problems raised by this work. Proof of the asymptotic time bounds of our algorithm is given in the Appendix.

2 Problem Formulation and Notation

Let the plane be divided by horizontal and vertical grid lines into square tiles each occupying unit area (i.e. the sides of each tile are of unit length). Also, let the coordinates of a tile be represented by the coordinates of its lower left hand corner. Our tile placement problem is basically that of assigning each tile to a disk in such a way that only K disks are used, and fixed radius queries on the plane will result in the shortest response time required to read in all tiles overlapping the query. This means that we should try to minimize the maximum number of tiles read from the same disk for each query.

In this paper, we will only consider the problem of finding the best tile allocation method, under the restriction that at most one tile is read from each disk. There are two reasons why we are only considering the one hit per disk case.

1. For most practical purposes, in order to guarantee smooth visualization without hiccups [12, 7] we can only afford a response time which involves at most one disk access per each parallel disk.

2. Usually, the number of disks K, and the size of tiles (relative to the query radius r) are changeable parameters. Thus it is usually possible to transform a fixed radius query problem that will require multiple tile reads from each disk, into one that just reads at most one tile per disk by increasing the tile size or adding more disks.

Specifically, in this paper we focus on the solution to the following two problems.

1. Given a system with K disks, what is the maximum query radius r, that would not result in two tiles being read per disk.

2. Given a query radius r, what is the minimum number of disks required to place all the tiles such that for all possible queries, at most one tile is read per disk.

In Section 4 we will also discuss our observations about the more general problem in which the one hit per disk restriction is removed.

Problem number 1 above, can be solved by letting the tiles that are assigned to the same disk be geometrically as far apart as possible. In such a case the maximum query radius r would simply be the minimal distance between two tiles assigned to the same disk divided by 2. Note that throughout this paper we make the simplifying assumption that a tile need not be read unless the center of the tile is accessed by the query circle. This assumption is not restrictive in any way, as in the case where a tile needs to be read in when any small part of it is accessed, we can simply subtract the constant $\frac{\sqrt{2}}{2}$ (half the diagonal length of a tile) from r to compute the real query radius. Problem number 2 has a complexity within a factor of $O(K)$ of problem 1 as it can be solved by solving problem number 1 for increasing values of K starting with $K = 1$, until we encounter a value for K that will work for a query radius larger or equal to r. In practice however, we need only start the search from $K = \lceil 2\sqrt{3}r^2 \rceil$ (this will become obvious in Section 3.1), and usually the smallest satisfying K will be found to be just slightly larger than $\lceil 2\sqrt{3}r^2 \rceil$.

In this paper we propose to use, what we call, a *vector based declustering* method of tile placement. This method takes a pair of linearly independent two dimensional vectors $\mathbf{u} = (a, b)$ and $\mathbf{v} = (c, d)$, and places all tiles with a relative position of $m\mathbf{u} + n\mathbf{v}$ (for all integer m, n) on the same disk. This is illustrated in Figure 2 for the vector pair $(4, -1)$ and $(3, 3)$. Another way of viewing this method is to imagine that we are tiling the plane with parallelograms with sides $\mathbf{u}$ and $\mathbf{v}$. Tiles that are at the same relative position on the parallelograms are assigned to the same disk. Note that since tiles are unit squares, a, b, c and d must all be integers. Furthermore, since exactly K disks must be used to hold all the tiles, the area of each parallelogram must be equal to K (i.e. $|ad - bc| = K$). Given a pair of vectors $\mathbf{u} = (a, b)$ and $\mathbf{v} = (c, d)$ which satisfy the above restriction, we can label tiles with their respective disks as follows:

For $0 \leq i \leq K - 1$
  Label a random unlabeled tile, say, (x, y) with i.
  Label all tiles with coordinates of the form
    $(x + ma + nc, y + mb + nd)$ with i
Endfor

It can be proved formally that this placement scheme is feasible, i.e., each tile will be labeled with a number from 0 to $K - 1$ and no tiles will be left unlabeled. Intuitively, this can be seen by observing that each tile is either totally contained in a parallelogram (such as tiles numbered 3,4,6,7,A,B,D,E in Figure 2) or is intersected by an edge of the parallelogram. Although tiles which lie along edges might only be partially covered by the parallelogram one can always find the corresponding other part of that particular tile at the opposite side of the parallelogram. For that reason, the number of tiles occupied by the parallelogram will always add up to the integer value K.

At first thought, it seems restrictive to only consider vector based tile placement methods, since there are other methods that can be considered (see Faulotsos [5]). But as we will show in the following section, this method is very attractive both for its performance and simplicity. In fact, for fixed radius queries in which we wish to read at most one tile from each disk, vector based methods usually produce tile placement results that are very close to optimal.

At this point, problem 1 has been transformed into the problem of finding a pair of vectors $\mathbf{u}, \mathbf{v}$, such that the distance between the closest endpoints among $m\mathbf{u} + n\mathbf{v}$ are as far apart as possible. Under vector based tile placement methods, the disks are treated symmetrically, i.e., for a specific tile (x, y) assigned to disk i, the relative positions of all other tiles assigned to disk i are the same regardless of the position of (x, y) as well as the value of i. Because of this we need only consider the relative positions of tiles assigned to the same disk as tile $(0, 0)$. Let the set of vectors spanned by $\mathbf{u}$ and $\mathbf{v}$ be denoted as $S(\mathbf{u}, \mathbf{v})$. In other words, let $S(\mathbf{u}, \mathbf{v}) = \{\mathbf{w} \mid \forall m, n \in \mathbf{Z},\ \mathbf{w} = m\mathbf{u} + n\mathbf{v}\}$. Also define $L(W) = \min\{|\mathbf{w}| \mid \mathbf{w} \in W, \mathbf{w} \neq (0, 0)\}$. In other words, $L(W)$ is the length of the shortest non-zero vector in the set W. Thus problem 1 is transformed into the problem of finding a pair of vectors that maximize $L(S(\mathbf{u}, \mathbf{v}))$, under the constraint that the parallelogram spanned by $\mathbf{u}$ and $\mathbf{v}$ has an area of K. Note that the solution will not be unique, since $S(\mathbf{u}, \mathbf{v}) = S(-\mathbf{u}, \mathbf{v}) = S(\mathbf{u}, -\mathbf{v}) = S(-\mathbf{u}, -\mathbf{v})$

Table 1 lists all the Major notations used in this paper along with their meaning. Some of them have already been defined, whereas others will be defined later in this paper.

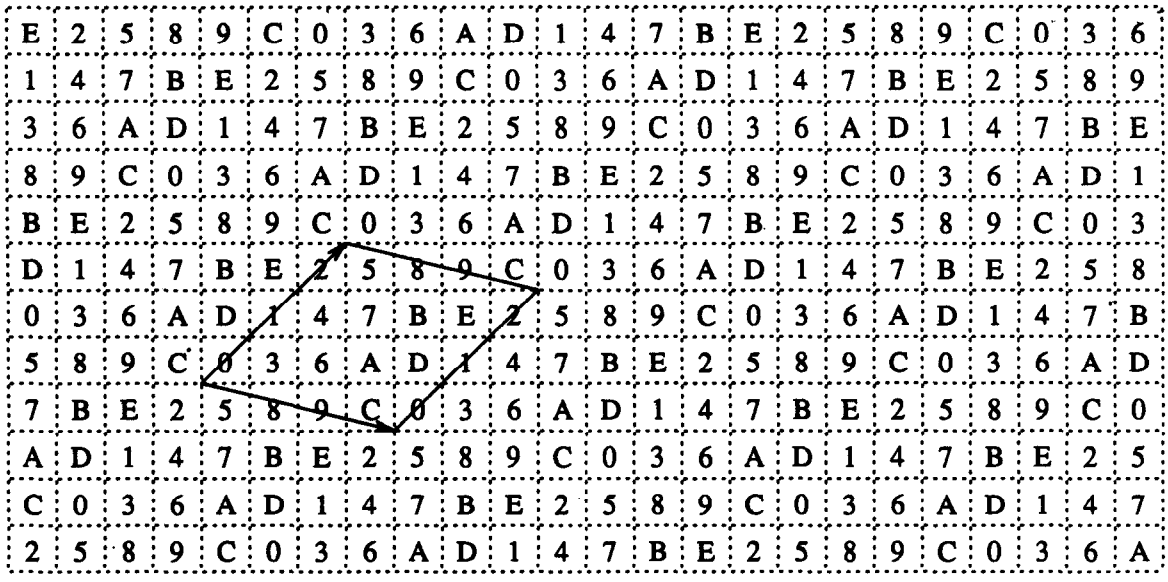


Figure 2: Tile placement on 15 disks with base vectors $(4, -1)$ and $(3, 3)$

Notation	Meaning
K	Number of parallel disks
r	Radius of query in tile length units
$\mathbf{u} = (a, b)$	2-D vector and its x, y coordinates
$\lvert\mathbf{u}\rvert$	the length of $\mathbf{u}$, $\sqrt{a^2 + b^2}$
$S(\mathbf{u}, \mathbf{v})$	$\{\mathbf{w} \mid \forall m, n \in \mathbf{Z},\ \mathbf{w} = m\mathbf{u} + n\mathbf{v}\}$
$L(W)$	$\min\{\lvert\mathbf{w}\rvert \mid \mathbf{w} \in W, \mathbf{w} \neq (0,0)\}$
$\mathbf{w}_i$	Temporary vector used at stage i
θ_i	Angle between vectors $\mathbf{w}_i$ and $\mathbf{w}_{i-1}$

Table 1: List of all notations used in this paper

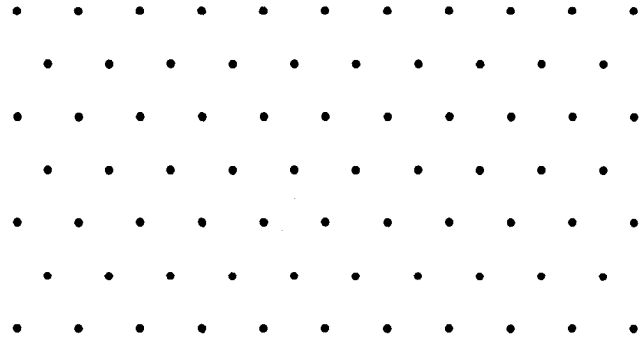


Figure 3: Hexagon Pattern

3 Algorithm to find best u, v pair

In this section we will discuss how to find the best $\mathbf{u}, \mathbf{v}$ pair such that $L(S(\mathbf{u}, \mathbf{v}))$ is maximized, under the constraint that the parallelogram spanned by $\mathbf{u}$ and $\mathbf{v}$ has an area of K. In order to get some bounds against which we can measure the quality of our solution, we start by discussing the optimal tile placement solution obtained by removing all restrictions, i.e., we allow tiles to be placed at non integer coordinate positions, and also lift the assumption that we only consider vector based solutions. Based on this optimal solution, we then investigate how close to optimal we can get with the restrictions. Next, we show that even with our restrictions we were able to come up with closed form solutions to vectors $\mathbf{u}, \mathbf{v}$ for a large range of practical values of K which take a special form. Following that we present our general solution in the form of a fast algorithm that can compute the best $\mathbf{u}, \mathbf{v}$ pair for any given K.

3.1 Optimal solution

Let us assume that the coordinate space we are dealing with is continuous, i.e., that the tile centers can be placed anywhere on the plane. Let us consider the points which represent tiles residing on the same disk. The problem is to place these points on a 2-D plane in such a way that the distance between the closest points are as far apart as possible. Also, since we are using K disks, we must have one point per area K on the average. In formal mathematical terms, this means that the number of points contained in a circle with radius R approaches $\pi R^2/K$ as R goes toward infinity.

This problem is a variant of the classical circle

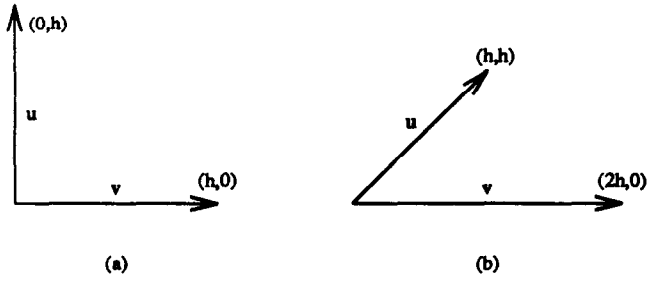


Figure 4: Closed form solutions when $K = h^2$ and $K = 2h^2$

packing problem and has been long solved [11]. Its solution states that the points must be distributed according to the hexagon pattern as shown in Figure 3. Note that the problem did not assume that the solution needed to be vector based, though the solution turned out to be vector based. The optimal solution is for $\mathbf{u}$ and $\mathbf{v}$ to be of the same length with a 60 (or 120) degree angle between them. Note that the smallest distance between any two points is $\sqrt{\frac{2K}{\sqrt{3}}}$. Thus the maximum query radius allowed would be $\sqrt{\frac{2K}{\sqrt{3}}}/2$.

3.2 Solving the problem for certain K values

Though the problem of finding the best $\mathbf{u}, \mathbf{v}$ pair can be solved optimally by letting $\mathbf{u} = (\sqrt{\frac{2K}{\sqrt{3}}}, 0)$ and $\mathbf{v} = (\frac{1}{2}\sqrt{\frac{2K}{\sqrt{3}}}, \frac{\sqrt{3}}{2}\sqrt{\frac{2K}{\sqrt{3}}})$ in continuous space. In our tile placement problem, the x and y components of $\mathbf{u}$ and $\mathbf{v}$ must be integers.

Although we have not been able to come up with closed form solutions for $\mathbf{u}$ and $\mathbf{v}$ for all K, we have been able to find closed form solutions for some often used subsets of K. These special cases are illustrated in Figure 4.

As in most practical applications the number of disks used is a power of 2, we are interested in the quality of a vector based declustering solution for $K = 2^l$ for an integer l. In fact we will deal with an even more general class of values for K of the form h^2 and $2h^2$. Note that any number which is a power of 2 can be represented as h^2 or $2h^2$ for some integer h. If $K = h^2$ for some integer h, the base vector pair could be set to $\mathbf{u} = (0, h)$ and $\mathbf{v} = (h, 0)$ as shown in Figure 4(a). This would result in a maximum query radius of $\sqrt{K}/2$, which is 6.94 percent off from the optimal of $\sqrt{\frac{2K}{\sqrt{3}}}/2$ achieved by the hexagonal pattern.

If $K = 2h^2$ for some integer h, the base vector pair could be set to $\mathbf{u} = (h, h)$ and $\mathbf{v} = (2h, 0)$ as shown in Figure 4(b). This would also result in a maximum query radius of $\sqrt{K}/2$, thus being 6.94 percent off from the hexagon optimal.

Also note that we are not saying that these solutions for $K = h^2$ and $K = 2h^2$ are the best we can do. We are merely saying that being off from optimal by 6.94 percent is a worst case upper bound for these sets of K's. In the following subsection, we will describe an algorithm that will compute the best possible $\mathbf{u}, \mathbf{v}$ vector pair for a given K. As will be seen later, we can often do better than the 6.94 percent for most values of K.

3.3 Solving the problem for any K

Let $\mathbf{u} = (a, b)$ and $\mathbf{v} = (c, d)$, where a, b, c and d are all integers. Under these constraints, trying to find the optimal vector pair by exhaustively considering all $\mathbf{u}, \mathbf{v}$ pairs would involve a search through 4 dimensional integer space. This search space is actually reduced to 3 dimensional space since we must have $|ad - bc| = K$. For each $\mathbf{u}, \mathbf{v}$ pair that is encountered in this search, $L(S(\mathbf{u}, \mathbf{v}))$ would need to be computed. A naive approach to computing $L(S(\mathbf{u}, \mathbf{v}))$ would involve a 2 dimensional search through m, n space, thus leading to a 5 dimensional search for each K. In this subsection, we introduce an algorithm that can search for the best possible $\mathbf{u}, \mathbf{v}$ pair for a fixed K in $O(K^{\frac{3}{2}} \log K)$ time. We first discuss how to further reduce the search space size to $O(K^{\frac{3}{2}})$, then we will describe an algorithm that can compute $L(S(\mathbf{u}, \mathbf{v}))$ in $O(\log K)$ time. We begin by presenting the following theorems.

THEOREM 3.1 *For all integer j, it will be true that*

$$S(\mathbf{u}, \mathbf{v}) = S(\mathbf{u}, \mathbf{v} + j\mathbf{u})$$

$$S(\mathbf{u}, \mathbf{v}) = S(\mathbf{v}, \mathbf{u} + j\mathbf{v})$$

PROOF: We shall only prove the case for $\mathbf{u}$ and $\mathbf{v} + j\mathbf{u}$. The other proof is similar.

In order to prove this we need to show that all vectors in the set $m'\mathbf{u} + n'(\mathbf{v} + j\mathbf{u})$ are found in the set $m\mathbf{u} + n\mathbf{v}$. This can be easily achieved by setting $m = m' + jn'$ and $n = n'$ for any m', n' pair.

The inverse can just as easily be shown. All vectors in the set $m\mathbf{u} + n\mathbf{v}$ can be found in the set $m'\mathbf{u} + n'(\mathbf{v} + j\mathbf{u})$ by setting $n' = n$ and $m' = m - jn$. □

Note that the area of the parallelogram spanned by these vector pairs is also preserved. This can be easily verified by computing $|ad - bc|$ for the vector pairs.

THEOREM 3.2 *For any pair of $\mathbf{u}, \mathbf{v}$ vectors with integer x, y components, we can always find another pair of vectors $\mathbf{u}', \mathbf{v}'$ such that $S(\mathbf{u}, \mathbf{v}) = S(\mathbf{u}', \mathbf{v}')$ and the y component of $\mathbf{v}'$ is always 0.*

PROOF: Based on Theorem 3.1 we know that we can transform the vector pair $\mathbf{u}, \mathbf{v}$ into a different pair that spans the same vector set. The question now is just how to select the j values while repeatedly performing this transformation, such that at the end, the y component of one of the two vectors will be 0.

This can be easily done by applying the Euclidian method of finding the greatest common divisors to the original vector pair in such a way that we end up with a vector in which its y component is equal to the gcd of the original two y components. When this happens, we only need to apply this procedure one more time in order to bring the y component of one of these two vectors to 0. □

Because of this we can always transform the vectors $\mathbf{u}, \mathbf{v}$ into a pair of vectors such that d is always 0, and this pair of new vectors would span the exact same set of vectors as the original $\mathbf{u}, \mathbf{v}$ pair. Thus it is sufficient to consider only the vector pairs with $d = 0$. Since $d = 0$, we must have $|bc| = K$, and since b, c and K are all integers, K must be divisible by b and c. Due to the fact that $S(\mathbf{u}, \mathbf{v}) = S(-\mathbf{u}, \mathbf{v}) = S(\mathbf{u}, -\mathbf{v}) = S(-\mathbf{u}, -\mathbf{v})$, it is sufficient to consider only positive b and c.

We have now reduced the search space to 2 dimensions. We proceed with the search by setting b to a divisor of K. For each such b that we try (there will be at most $\lceil 2\sqrt{K} \rceil$ of them), c can be determined by computing K/b. Then, because $S(\mathbf{u}, \mathbf{v}) = S(\mathbf{v}, \mathbf{u} + j\mathbf{v})$, we need only try all values of a between $-c + 1$ and $c - 1$. Since there are only $O(\sqrt{K})$ possibilities to try for b, and only $O(K)$ for a. The total number of $\mathbf{u}, \mathbf{v}$ pairs we would need to search is of $O(K^{\frac{3}{2}})$.

For each $\mathbf{u}, \mathbf{v}$ pair that we come across in this search, we need to compute $L(S(\mathbf{u}, \mathbf{v}))$. We find this value by applying an $O(\log K)$ time algorithm that iteratively transforms the vector pair $\mathbf{u}, \mathbf{v}$ into different vector pairs that preserve the positions of the spanned vectors $m\mathbf{u} + n\mathbf{v}$. We terminate the algorithm when we finally come to a vector pair $\mathbf{u}', \mathbf{v}'$ such that

$$|\mathbf{u}' \cdot \mathbf{v}'| \leq \frac{1}{2}(\min(|\mathbf{u}'|, |\mathbf{v}'|))^2 \qquad (1)$$

When this is true, we shall prove that $L(S(\mathbf{u}, \mathbf{v})) = \min(|\mathbf{u}'|, |\mathbf{v}'|)$.

THEOREM 3.3 *If $\mathbf{u}'$ and $\mathbf{v}'$ are two 2 dimensional vectors that satisfy Equation 1, then the vector with the minimal length among the set of vectors $m\mathbf{u}' + n\mathbf{v}'$, where m and n are integers that are not simultaneously 0, will have length equal to* $\min(|\mathbf{u}'|, |\mathbf{v}'|)$.

PROOF: Without loss of generality, we shall make the assumption that $|\mathbf{u}'| \geq |\mathbf{v}'|$. Clearly by setting $(m, n) = (0, 1)$ we will have found a vector with length $\min(|\mathbf{u}'|, |\mathbf{v}'|) = |\mathbf{v}'|$. So all we need to do is show that for all m, n pairs not simultaneously 0

$$|m\mathbf{u}' + n\mathbf{v}'| \geq |\mathbf{v}'|$$

This is true if and only if

$$(m\mathbf{u}' + n\mathbf{v}') \cdot (m\mathbf{u}' + n\mathbf{v}') \geq |\mathbf{v}'|^2$$

The left side of the equation can be rewritten as:

$$m^2|\mathbf{u}'|^2 + 2nm\mathbf{u}' \cdot \mathbf{v}' + n^2|\mathbf{v}'|^2 \geq$$

$$m^2|\mathbf{v}'|^2 - |nm||\mathbf{v}'|^2 + n^2|\mathbf{v}'|^2 =$$

$$(|m| \cdot |m| - |n| \cdot |m| + |n| \cdot |n|)|\mathbf{v}'|^2$$

The last term can be easily verified to be greater or equal to $|\mathbf{v}'|^2$, by considering the three cases where $|n| = |m| \neq 0$, $|n| > |m|$, and $|m| > |n|$. □

Now let us describe the algorithm to find a pair of vectors $\mathbf{u}', \mathbf{v}'$ that satisfy Equation 1. The algorithm is as follows:

1. Let $\mathbf{u}, \mathbf{v}$ be assigned to $\mathbf{w}_1$ and $\mathbf{w}_2$, where $\mathbf{w}_1$ is assigned the longer of the two vectors and $\mathbf{w}_2$ is assigned the shorter.
2. Compute $\mathbf{w}_{i+1}$ to be the vector $\mathbf{w}_{i-1} + j\mathbf{w}_i$ where j is an integer selected to minimize $|\mathbf{w}_{i+1}|$.
3. Repeat Step 2 until either $|\mathbf{w}_{i+1}| \geq |\mathbf{w}_i|$ or $j = 0$ for a certain step.
4. If the algorithm was terminated because $j = 0$, the two vectors $\mathbf{w}_{i-1}$ and $\mathbf{w}_i$ that caused j to be zero in Step 2 are the resulting vectors $\mathbf{u}'$ and $\mathbf{v}'$. If the algorithm was terminated because $|\mathbf{w}_{i+1}| \geq |\mathbf{w}_i|$, then $\mathbf{w}_{i+1}$ and $\mathbf{w}_i$ will be the resulting vectors.

The fact that the vectors spanned by $\mathbf{w}_{i+1}$ and $\mathbf{w}_i$ are exactly equal to the vectors spanned by $\mathbf{w}_i$ and $\mathbf{w}_{i-1}$ has already been shown in Theorem 3.1. It should also be obvious that at Step 2, we will always have $|\mathbf{w}_i| < |\mathbf{w}_{i-1}|$, otherwise the algorithm would have terminated previously. As an example, consider an initial vector pair of $(55, 0)$ and $(-39, 1)$ Table 2 shows the sequence of j's selected and the computed $\mathbf{w}_i$'s. The final $\mathbf{u}', \mathbf{v}'$ pair in this example are $(2, 7)$ and $(-7, 3)$, and thus $L(S(\mathbf{u}, \mathbf{v})) = |(2, 7)| = \sqrt{2^2 + 7^2}$

Also, note that every step of the algorithm, including Step 2, can be performed in constant time. This is because in Step 2, we can select j as follows: Let $\mathbf{w}_{i-1} = (a, b)$ and $\mathbf{w}_i = (c, d)$. We know that

$$|\mathbf{w}_{i+1}|^2 = |\mathbf{w}_{i-1} + j\mathbf{w}_i|^2 = (a + jc)^2 + (b + jd)^2 \qquad (2)$$

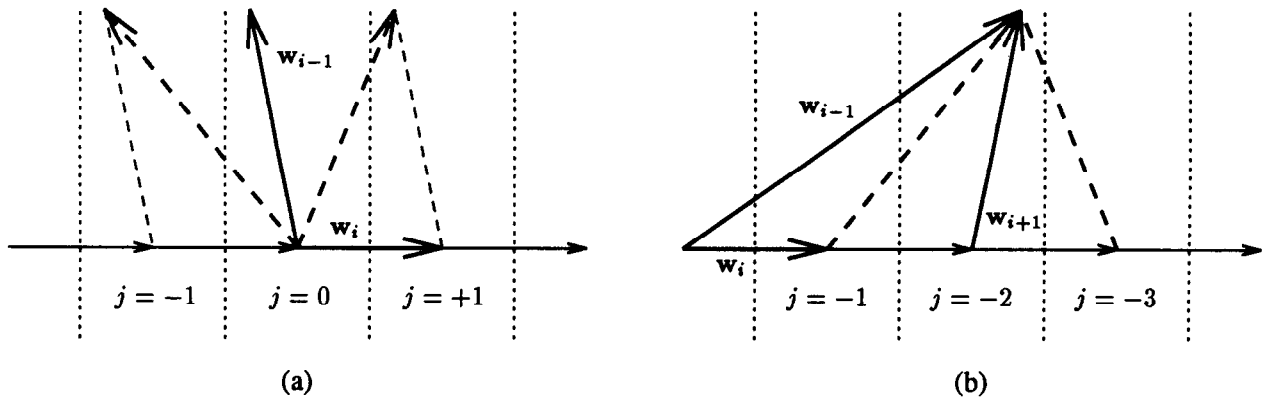


Figure 5: Proof that equation holds when algorithm terminates

$\mathbf{w}_i$	j	vector	$\mathbf{w}_{i-1}+j\mathbf{w}_i$
$\mathbf{w}_1$		$(55,0)$	
$\mathbf{w}_2$		$(-39,1)$	
$\mathbf{w}_3$	$+1$	$(16,1)$	$(55,0)+1\cdot(-39,1)$
$\mathbf{w}_4$	$+2$	$(-7,3)$	$(-39,1)+2\cdot(16,1)$
$\mathbf{w}_5$	$+2$	$(2,7)$	$(16,1)+2\cdot(-7,3)$
$\mathbf{w}_6$	0		$(-7,3)+0\cdot(2,7)$

Table 2: Example of algorithm to compute $L(S((55,0),(-39,1)))$

To find the value of j that minimizes this equation, we take its derivative and set it to zero. The result is that j should be $-\frac{ac+bd}{c^2+d^2}$. But this value will not necessarily be an integer. However, since Equation 2 is a parabola, we know that the integer j that will minimize Equation 2 must be one of the two integers next to $-\frac{ac+bd}{c^2+d^2}$. Thus the correct j value to use can be easily found in constant time.

We shall now prove that when the algorithm terminates, the condition in Equation 1 will hold. But first we show the following lemma.

LEMMA 3.1 *Equation 1 will hold if the following is true:*

$$|\mathbf{u}'+\mathbf{v}'| \geq \max(|\mathbf{u}'|,|\mathbf{v}'|)$$

$$\textit{and} \qquad |\mathbf{u}'-\mathbf{v}'| \geq \max(|\mathbf{u}'|,|\mathbf{v}'|) \qquad (3)$$

PROOF: Without loss of generality, let us assume that $|\mathbf{u}'| \geq |\mathbf{v}'|$.

From $|\mathbf{u}'+\mathbf{v}'| \geq |\mathbf{u}'|$ we can deduce that:

$$(\mathbf{u}'+\mathbf{v}')\cdot(\mathbf{u}'+\mathbf{v}') \geq |\mathbf{u}'|^2$$

$$\mathbf{u}'\cdot\mathbf{v}' \geq -\frac{1}{2}|\mathbf{v}'|^2$$

Similarly, we deduce from $|\mathbf{u}'-\mathbf{v}'| \geq |\mathbf{u}'|$ that $\mathbf{u}'\cdot\mathbf{v}' \leq \frac{1}{2}|\mathbf{v}'|^2$

And from these two results, we have:

$$|\mathbf{u}'\cdot\mathbf{v}'| \leq \frac{1}{2}|\mathbf{v}'|^2$$

□

Now let us prove that the algorithm works correctly. In other words, we shall show that Equation 1 will hold for $\mathbf{u}', \mathbf{v}'$ when the algorithm terminates.

PROOF: First we shall consider the case where the algorithm terminated because $j=0$ at Step 2. Under this condition, $\mathbf{w}_{i-1}$ and $\mathbf{w}_i$ become $\mathbf{u}'$ and $\mathbf{v}'$, and we know that $|\mathbf{w}_{i-1}| \geq |\mathbf{w}_i|$. This is illustrated in Figure 5(a). The fact that j was selected to be 0 means that if j were -1 or $+1$, the vector $\mathbf{w}_{i-1}+j\mathbf{w}_i$ will be greater or equal to $\mathbf{w}_{i-1}$ (These vectors are illustrated by dashed lines). This is basically the same as Equation 3. Hence, based on the lemma, we know that Equation 1 will hold for the output vectors $\mathbf{w}_{i-1}$ and $\mathbf{w}_i$.

Now we consider the case where the algorithm terminated because $|\mathbf{w}_{i+1}| \geq |\mathbf{w}_i|$ at Step 2 (as illustrated in Figure 5(b)). Under this circumstance, $\mathbf{w}_{i+1}$ and $\mathbf{w}_i$ will become the output vectors $\mathbf{u}'$ and $\mathbf{v}'$. The fact that j was not one more or one less than what it was at Step 2 also implies that Equation 3 holds for $\mathbf{w}_{i+1}$ and $\mathbf{w}_i$. Hence, based on the lemma, we know that Equation 1 will hold for the output vectors too. □

Now the only thing left to show is that the algorithm does indeed terminate and does so in logarithmic time bounds. Because the details of this proof are tedious and diverge away from the major topic of this paper, we have put the detailed proof in Appendix A. A sketch

of the proof can be seen from the following lemmas and theorems.

Let θ_i be the angle between $\mathbf{w}_{i-1}$ and $\mathbf{w}_i$ measured in the direction that makes $\theta_i \leq \pi$.

LEMMA 3.2 *If during Step 2, the newly computed* $\mathbf{w}_{i+1}$ *resulted in* $\frac{\pi}{3} \leq \theta_{i+1} \leq \frac{2\pi}{3}$. *Then the algorithm will at most execute Step 2 one more time before terminating.*

LEMMA 3.3 *If* j *was not selected to be 0 when computing* $\mathbf{w}_{i+1}$ *at Step 2 of the algorithm, then it will either be true that* $|\tan\theta_{i+1}| \geq 3 \cdot |\tan\theta_i|$ *or it will be true that* $\frac{\pi}{3} \leq \theta_{i+1} \leq \frac{2\pi}{3}$.

THEOREM 3.4 *If the* x *and* y *components of the initial* $\mathbf{u}, \mathbf{v}$ *pair are less than or equal to* K *in absolute value, then the entire algorithm will terminate in* $O(\log K)$ *time.*

4 Implementation Results and Discussion

We have implemented this algorithm and found the best vector pairs for all K values from 4 through 1024. Table 3 lists the results for K from 4 through 32.

One interesting observation about our results, which seems a bit counter-intuitive, is that the maximal query radius r is not guaranteed to increase monotonically with the number of disks K. For example, we can come up with a tile placement method that can satisfy any query with radius less than 2.062 using only 15 disks. But with 16 disks, the best we can do is place tiles in such a way that queries with radius less than 2.0 be satisfied. Though queries with radius between 2.0 and 2.062 might result in two hits per disk. This does not always mean that we should always throw one disk away from a 16 disk system. There are other benefits for using this extra disk, such as added storage capacity (each disk only needs to store $\frac{1}{16}$ of the entire tile set, instead of $\frac{1}{15}$), and smaller number of disks being hit when the query radius is much larger than 2. The system integrator would need to weigh these issues and determine whether to use the 16th disk or not.

The difference in percentage from the maximal query radius r to that of the optimal hexagon pattern is illustrated in Figure 6. As one can see the solution gets better as K becomes larger. This is due to the fact that as K becomes larger the resolution of grid points relative to the vector lengths become more and more like continuous space, and thus we are able to approximate the hexagon vectors more and more closely.

In Figure 7 we show the same data in a different way. We show the maximum achievable query radius r (the dots) compared with the optimal r in continuous space (the line) for all values of K from 4 through 1024.

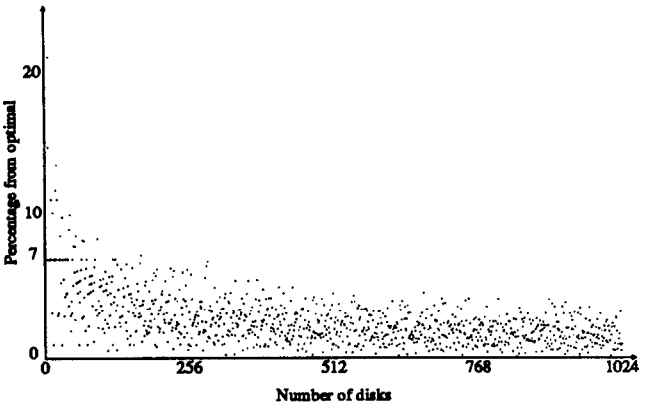


Figure 6: Percentage off from hexagon optimal for different values of K

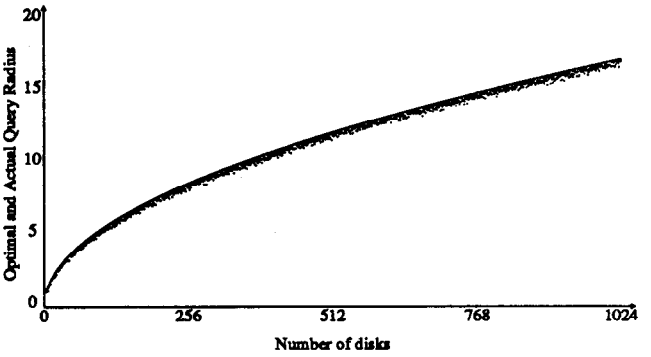


Figure 7: Optimal and actual query radius for different values of K

One interesting question is: What will happen under these allocations when the query radius is larger than the maximum allowed to guarantee at most one hit per disk? We can easily show that regardless of the vector base used, the average number of tiles from the same disk hit by a query of radius r will always be $\pi r^2/K$ (See Chapter 1 of [8]). Thus saying that no matter what pair of base vectors are used, the average number of hits per disks will always be equal to the area of the query divided by the area of the parallelogram. This might lead one to come to the conclusion that when r is large (relative to $\sqrt{K}$), it does not make a difference what vector base to choose, since "on the average" the number of hits per disk is always the same. This, however, is incorrect because we are not concerned with the average number of times a disk is hit, but with the number of times the maximally hit disk is accessed. This is due to the fact that the query response time on that disk will dominate the total response time.

K	Initial $(a,b),(c,d)$	Final $\mathbf{u}', \mathbf{v}'$	maximal r	optimal r	% difference
4	$(-2,1),(4,0)$	$(0,2)(-2,1)$	1.000	1.075	6.94
5	$(-3,1),(5,0)$	$(2,1)(-1,2)$	1.118	1.201	6.94
6	$(-4,1),(6,0)$	$(2,1)(-2,2)$	1.118	1.316	15.05
7	$(-5,1),(7,0)$	$(2,1)(-1,3)$	1.118	1.422	21.35
8	$(-5,1),(8,0)$	$(-2,2)(-1,-3)$	1.414	1.520	6.94
9	$(-6,1),(9,0)$	$(0,3)(3,1)$	1.500	1.612	6.94
10	$(-7,1),(10,0)$	$(3,1)(-1,3)$	1.581	1.699	6.94
11	$(-8,1),(11,0)$	$(3,1)(-2,3)$	1.581	1.782	11.27
12	$(-3,2),(6,0)$	$(-3,2)(3,2)$	1.803	1.861	3.14
13	$(-8,1),(13,0)$	$(-3,2)(-2,-3)$	1.803	1.937	6.94
14	$(-10,1),(14,0)$	$(-2,3)(4,1)$	1.803	2.010	10.32
15	$(-11,1),(15,0)$	$(4,1)(1,4)$	2.062	2.081	0.93
16	$(-12,1),(16,0)$	$(0,4)(4,1)$	2.000	2.149	6.94
17	$(-13,1),(17,0)$	$(4,1)(-1,4)$	2.062	2.215	6.94
18	$(-13,1),(18,0)$	$(-3,3)(2,4)$	2.121	2.280	6.94
19	$(-15,1),(19,0)$	$(4,1)(-3,4)$	2.062	2.342	11.97
20	$(-12,1),(20,0)$	$(-4,2)(0,-5)$	2.236	2.403	6.94
21	$(-15,1),(21,0)$	$(-3,3)(3,4)$	2.121	2.462	13.84
22	$(-17,1),(22,0)$	$(-2,4)(5,1)$	2.236	2.520	11.27
23	$(-18,1),(23,0)$	$(-3,4)(5,1)$	2.500	2.577	2.98
24	$(-19,1),(24,0)$	$(5,1)(1,5)$	2.550	2.632	3.14
25	$(-20,1),(25,0)$	$(0,5)(5,1)$	2.500	2.686	6.94
26	$(-21,1),(26,0)$	$(5,1)(-1,5)$	2.550	2.740	6.94
27	$(-22,1),(27,0)$	$(5,1)(-2,5)$	2.550	2.792	8.68
28	$(-22,1),(28,0)$	$(2,5)(4,-4)$	2.693	2.843	5.29
29	$(-17,1),(29,0)$	$(-5,2)(-2,-5)$	2.693	2.893	6.94
30	$(-5,3),(10,0)$	$(-5,3)(5,3)$	2.915	2.943	0.93
31	$(-19,1),(31,0)$	$(2,-5)(-5,-3)$	2.693	2.991	9.99
32	$(-25,1),(32,0)$	$(-4,4)(3,5)$	2.828	3.039	6.94

Table 3: Listing of best vectors for $K = 4$ through 32

We have thought about the question: What is the optimal way to place tiles such that the query radius is maximized when we allow to hit each disk at most p times? So far we have not been able to come up with a definite answer for each individual value of p. Our initial findings indicate that tile placement methods good for one value of p will not necessarily be good for other values of p. Although we only solved the problem for $p = 1$ in this paper, we still feel that this is a very important case, since most fixed radius query problems can be transformed into one in which $p = 1$ by simply increasing the tile size. This also makes sense from the stand point of decreasing total access time, since the total disk access time involved in reading a tile of size X is usually much smaller than that of reading n tiles of size $\frac{X}{n}$ each. This is due to the fact that the number of seeks in the latter case increases by a factor of n although transfer time remains the same.

Based on these observations, our recommendation to system designers is to tune-up the parameters of the system carefully by allowing the maximum tile size such that $p = 1$ and the total time to seek and transfer a tile is within the application specified allowable response time. Other options include of course adding more disks or using faster disks.

5 Conclusion and Future Work

In this work we presented a new method of declustering image tiles in databases used for the purpose of visualization in GIS and scientific applications. Our algorithm was shown to find tile placement methods within 7 percent from optimum for a large range of system architectures used in practice. Unlike some other declustering algorithms, our method provides worst case guarantees such that for all queries up to a certain radius at most one disk will be hit. The declustering obtained follows a repetitive pattern which is easily computable for every value of K, the number of disks in the system.

It is interesting to note that the same declustering

method is not limited to image tiles but can also be applied to databases which store information about objects which are geometrically positioned on a grid. Such databases are used for example in high energy physics for storing information about sensors in a detector.

Some questions raised by this work which we plan to investigate are:

- Solving the declustering problem for heterogeneous environments that contain non-symmetrical disks. In such environments we may require that some of the faster disks be "hit" more than others to balance the response time of all disks.
- How can we use image tile replication (i.e. duplicating tiles on multiple disks) to further optimize the declustering algorithm.
- From a theoretical point of view it is interesting to find a closed form solution for the optimal vector pair for each given K.
- Finding a tile allocation method that can optimize (in a global sense) the number of times the maximally hit disk is accessed, regardless of the query radius r.
- Finding declustering schemes (vector based or otherwise) for 3-D visualization.

6 Acknowledgement

The authors wish to thank William Johnston head of the Imaging Technology Group at LBL for useful discussions regarding the motivation and results included in this work.

7 Appendix A: Proof of Algorithm's Asymptotic Bounds

In this appendix we shall show that the algorithm to compute $L(S(\mathbf{u},\mathbf{v}))$ for any $\mathbf{u},\mathbf{v}$ pair (encountered in the search for the optimal tile placement method for K disks) does indeed terminate and does so in logarithmic time bounds. We shall first prove two lemmas before showing this.

Let θ_i be the angle between $\mathbf{w}_{i-1}$ and $\mathbf{w}_i$ measured in the direction that makes $\theta_i \leq \pi$.

LEMMA 3.2 *If during Step 2, the newly computed* $\mathbf{w}_{i+1}$ *resulted in* $\frac{\pi}{3} \leq \theta_{i+1} \leq \frac{2\pi}{3}$. *Then the algorithm will at most execute Step 2 one more time before terminating.*

PROOF: We shall only consider the case where $\frac{\pi}{3} \leq \theta_{i+1} \leq \frac{\pi}{2}$. When $\frac{\pi}{2} < \theta_{i+1} \leq \frac{2\pi}{3}$, the proof is similar.

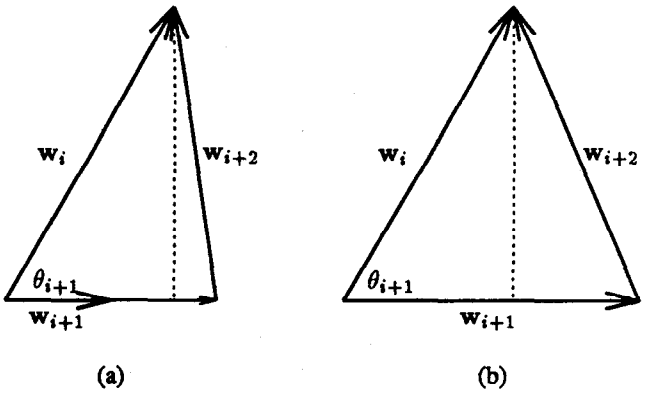


Figure 8: Proof that algorithm terminates after 60 degree vector angle

If $|\mathbf{w}_{i+1}| \geq |\mathbf{w}_i|$ then the algorithm would terminate immediately, and there would not be a need to execute Step 2 one more time. So in the following discussion, we will only consider the case where $|\mathbf{w}_{i+1}| < |\mathbf{w}_i|$. If when computing $\mathbf{w}_{i+2}$ on the following step, it turns out that $j = 0$ then the algorithm will also terminate. The only thing left to show is that when $j \neq 0$, it will be true that $|\mathbf{w}_{i+2}| \geq |\mathbf{w}_{i+1}|$, which would also terminate the algorithm.

We show this by considering two cases. If $|\mathbf{w}_{i+1}| \leq |\mathbf{w}_i| \cdot \cos\theta_{i+1}$ (i.e. The length of the projection of $\mathbf{w}_i$ onto $\mathbf{w}_{i+1}$), then we have

$$|\mathbf{w}_{i+2}| \geq |\mathbf{w}_i| \cdot \sin\theta_{i+1} > |\mathbf{w}_i| \cdot \cos\theta_{i+1} \geq |\mathbf{w}_{i+1}|$$

This is illustrated in Figure 8(a). The first inequality holds because the newly computed vector $\mathbf{w}_{i+2}$ must be longer than the perpendicular distance from the tip of $\mathbf{w}_i$ to the line spanned by $\mathbf{w}_{i+1}$. The second inequality holds due to the fact that $\frac{\pi}{3} \leq \theta_{i+1} \leq \frac{\pi}{2}$.

On the other hand, if $|\mathbf{w}_{i+1}| > |\mathbf{w}_i| \cdot \cos\theta_{i+1}$ (as illustrated in Figure 8(b)) then the only possible j value that could be selected besides 0 is -1. Any other j value would result in a longer $\mathbf{w}_{i+2}$. But when $j = -1$, by the Law of Cosines we know that

$$|\mathbf{w}_{i+2}|^2 = |\mathbf{w}_{i+1}|^2 + |\mathbf{w}_i|^2 - 2 \cdot |\mathbf{w}_{i+1}| \cdot |\mathbf{w}_i| \cdot \cos\theta_{i+1}$$

$$\geq |\mathbf{w}_{i+1}|^2 + |\mathbf{w}_i|^2 - |\mathbf{w}_{i+1}| \cdot |\mathbf{w}_i| \geq |\mathbf{w}_{i+1}|^2$$

□

LEMMA 3.3 *If* j *was not selected to be 0 when computing* $\mathbf{w}_{i+1}$ *at Step 2 of the algorithm, then it will either be true that* $|\tan\theta_{i+1}| \geq 3 \cdot |\tan\theta_i|$ *or it will be true that* $\frac{\pi}{3} \leq \theta_{i+1} \leq \frac{2\pi}{3}$.

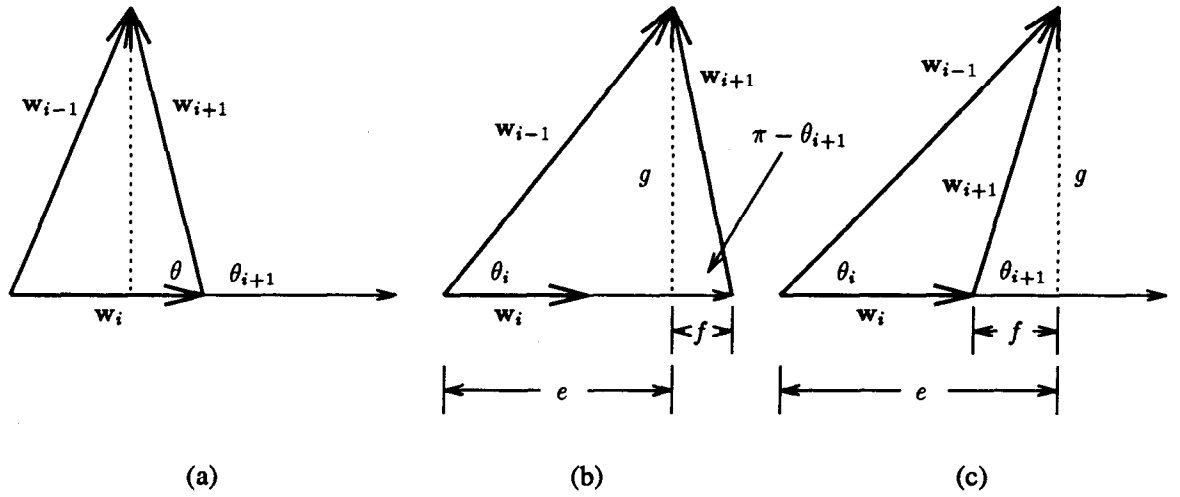


Figure 9: Proof that tangent increases by a factor of 3

PROOF: We shall only consider the case where $\theta_i \leq \frac{\pi}{2}$. When $\theta_i > \frac{\pi}{2}$, the proof is similar.

We shall also prove this lemma by considering two cases. If $|\mathbf{w}_i| \geq |\mathbf{w}_{i-1}| \cdot \cos\theta_i$, it must be true that $j = -1$. Consider the triangle formed by $\mathbf{w}_{i-1}$, $\mathbf{w}_i$, and $\mathbf{w}_{i+1}$ as illustrated in Figure 9(a). Because we know that $|\mathbf{w}_{i-1}| \geq |\mathbf{w}_i|$ and $|\mathbf{w}_{i-1}| \geq |\mathbf{w}_{i+1}|$, by the Law of Sines θ must be the largest angle of the three inner angles of the triangle. This implies that $\frac{\pi}{3} \leq \theta \leq \frac{\pi}{2}$, which means that $\frac{\pi}{2} \leq \theta_{i+1} \leq \frac{2\pi}{3}$.

On the other hand if $|\mathbf{w}_i| < |\mathbf{w}_{i-1}| \cdot \cos\theta_i$, we can easily see from Figure 9(b) and 9(c) that $|\tan\theta_i| = \frac{g}{e}$ and $|\tan\theta_{i+1}| = \frac{g}{f}$. Because j was selected to minimize $|\mathbf{w}_{i+1}|$, it must be true that $f \leq \frac{1}{2}|\mathbf{w}_i|$. Thus in both of the cases shown in the figure, it will be true that $e \geq 3f$, which in turn leads us to the conclusion that $|\tan\theta_{i+1}| \geq 3 \cdot |\tan\theta_i|$. □

THEOREM 3.4 *If the x and y components of the initial $\mathbf{u}, \mathbf{v}$ pair are less than or equal to K in absolute value, then the entire algorithm will terminate in $O(\log K)$ time.*

PROOF: Because of the restrictions on the x and y components of the initial $\mathbf{u}, \mathbf{v}$ pair, and the fact that they must be linearly independent. It can be easily shown that the tangent value of the angle between $\mathbf{u}$ and $\mathbf{v}$ must be at least $\frac{1}{2K}$. Based on this fact and the previous two lemmas, it should be obvious that Step 2 of the algorithm will be executed at most $\log_3 2\sqrt{3}K+2$ times before the algorithm terminates. Since every step of the algorithm can be performed in constant time, the entire algorithm can be executed in $O(\log K)$ time. □

References

[1] B. Chor, C. E. Leiserson, R. L. Rivest, and J.B. Shearer. "An Application of Number theory to the Organization of Raster-Graphics Memory". *Journal of the ACM*, 33(1):86–104, 1986.

[2] David J. DeWitt and S. Ghandeharizadeh. "Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machine". In *Proc. 16th international Conference on VLDB*, pages 481–492, August 1990.

[3] H. C. Du. "Disk Allocation Methods for Binary Cartesian Product Files". *BIT*, 26:138–147, 1986.

[4] H. C. Du and J. S. Sobolewski. "Disk Allocation for Cartesian Product Files on Multiple Disk Systems". *ACM Trans. on Database Systems*, 7(1):82–101, 1982.

[5] C. Faloutsos and P. Bhagwat. "Declustering Using Fractals". In *Second International Conference on Parallel and Distributed Computing (PDIS)*, pages 18–25, January 1992.

[6] F. Faloutsos and D. Metaxas. "Disk Allocation Methods Using Error Correcting Codes". *IEEE trans. on Computers*, 40(8):907–914, August 1991.

[7] S. Ghandeharizadeh, I. Ramos, Z. Asad, and W. Qureshi. "Object Placement in Parallel HyperMedia Systems". In *Proc. 17th international Conference on VLDB*, pages 243–254, September 1991.

[8] Peter Hall. *Introduction to the Theory of Coverage Processes.* John Wiley and Sons, New York, NY, 1988.

[9] Ramesh Jain. *NSF Workshop on Visual Information Management Systems.* Available by anonymous ftp from ftp.eecs.umich.edu, 1992.

[10] M. H. Kim and S. Pramanik. "Optimal File Distribution for Partial Match Retrieval". In *Proc. ACM SIGMOD Conf.*, pages 173–182, June 1988.

[11] Herbert Meschkowski. *Unsolved and Unsolvable Problems in Geometry.* Oliver and Boyd, Edinburgh, London, 1966.

[12] C. Yu, W. Sun, D. Bitton, Q. Yang, R. Brunno, and J. Tullis. "Efficient Placement of Audio on Optical Disks for Real-time Applications". *Communications of the ACM*, 32(7):862–871, 1989.