Design, Implementation, and Evaluation of a Parallel Image Shape Indexer

Tzi-cker Chiueh *Dimitris Margaritis* *Srinidhi Varadarajan*

Computer Science Department
State University of New York at Stony Brook
Stony Brook, NY 11794-4400
{*chiueh, dmarg, srinidhi*}@*cs.sunysb.edu*

Abstract

This paper describes the design, implementation, and evaluation of a parallel indexer called PAMIS [1] *for a polygonal 2D shape image database.* PAMIS *is based on a shape representation scheme called the* turning function*, which exhibits the desirable properties of position-, scale-, and rotation-invariance, and has a similarity metric function that satisfies the triangular inequality, which is required for efficient database indexing. Because the goal of the* PAMIS *project is to support "like-this" image queries, the indexing scheme we chose, the vantage-point tree (VPT), uses relative rather than absolute distance values to organize the database elements for efficient nearest-neighbor search. We have successfully implemented* PAMIS *on a network of workstations to exploit the I/O and computation parallelism inherent in the VPT algorithm. We found that it is preferable to make the VPT node size as small as possible in order to have a lean and deep VPT structure, and the best-case scheduling strategy performs the best among the scheduling strategies considered. Overall, the performance of the VPT algorithm scales very well with the number of processors, and the indexing efficiency, defined as the percentage of database elements touched by the search, of PAMIS is 6% and 39% for "good" queries that ask for 1 and 50 nearest neighbors, respectively.*

1 Introduction

The most popular form of content-based image queries is the so-called *like-this* query because it bypasses the difficult problem of specifying the desired data objects in terms of formal query languages, which seems neither technically nor ergonomically feasible. Supporting content-based multimedia object access involves two issues: *feature extraction*, which extracts features with distinguishing power from the original multimedia objects, and *feature indexing*, which organizes the objects in the database according to their extracted features in such a way that at run time, only a small portion of the database needs to be explored to retrieve the target objects. *Feature extraction* is a very old research area and the literature abounds in well-known techniques to extract meaningful features from various types of signals such as images, video and speech. *Feature indexing*, on the other hand, is less studied and is the subject of this paper.

In this paper, we describe the design, implementation, and evaluation of an experimental image index system called *PAMIS*, developed at the Computer Science Department of State University of New York at Stony Brook. The first *PAMIS* prototype is fully operational now and can support content-based image retrieval by returning the most similar image in the database to the query image entered by the user. *PAMIS* employs a shape representation scheme called the *turning function*, which is equipped with such desirable properties as position-, scale-, and rotation-invariance, and has a similarity metric function that satisfies the triangular inequality. Based on this representation, *PAMIS* uses with a parallel version of the *vantage-point tree* (VPT) algorithm [3], which is implemented on a network of high-end PC's to exploit the I/O and computation parallelism in the tree search process, and supports the capability of searching for N ($N \geq 1$) nearest neighbors.

The rest of this paper is organized as follows. In Section 2, we briefly review previous works in this area to set the contributions of this work in perspective. In Section 3, we describe the shape representation scheme used in *PAMIS* and the associated similarity metric function. Section 4 discusses the design and the parallel implementation details of the VPT algorithm. In Section 5, the experimental methodology and the results and analysis of our experiments on the first *PAMIS* prototype are presented in detail. Section 6 concludes this paper with a summary of the main results from this work as well as an outline of on-going research.

[1] **PA**rallel **M**ultimedia **I**ndex **S**erver

2 Related Work

[10] proposed the vantage-point tree for efficient nearest-neighbor search by organizing database elements based on their relative distances rather than their absolute feature values. [3] extended that work by developing an optimistic σ value adjustment mechanism that at once solves the problem of choosing the initial σ value and achieves the optimal balance between recall and precision rates by dynamically tailoring the range of search to the characteristics of the given queries. The work described in this paper further improves the previous two efforts by introducing the idea of *accumulative* vantage-point tree, which allows the propagation of useful information from parents to children to attain better pruning efficiency. In addition, a parallel implementation on a network of workstations is developed that is capable of exploiting the inherent I/O and computation parallelism in the search procedure.

Perhaps the most ambitious image database project so far is the QBIC (Query by Image Content) [5] [6] from IBM Almaden, which uses both automatic and manual image analysis techniques to extract feature vectors, and then uses conventional multi-dimensional indexing methods such as R^*-trees as the indexing mechanism for low-dimensional feature vectors. In the case of high-dimensional feature vectors, a pre-processing step based on principal component analysis is used to reduce the dimensionality to two or three. The pre-processed feature vectors are then indexed by R^*-trees. [2] proposes another algorithm to the multi-dimensional nearest-neighbor search problem based on a randomization approach. However, this algorithm doesn't seem to be as effective as the Vantage-Point Tree method. [7] and [8] described experiments to retrieve image objects based on their shapes. However, the emphasis of both is on the image representation schemes to support occlusion or partial matching, rather than on efficient indexing mechanisms to speed up the access. The FIBSSR project described in [9] used a K-D-B tree index mechanism.

3 Polygonal Shape Description

3.1 Representation

The first *PAMIS* prototype is focused only on polygonal shape images. We choose the *turning function* representation [1], which exhibits the desirable properties of position-, rotation- and scale-invariance. The turning function of a polygon represents the tangent of a point on the boundary with respect to certain reference axis of arbitrary orientation (Figure 1). As the perimeter of a polygon is traversed, the tangent at each point is computed, thus effectively transforming a 2D shape into a 1D turning function while preserving all information. The point on the shape's contour from which

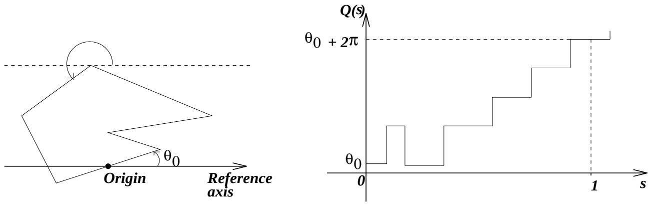

Figure 1. The left is the original polygon while the right is its corresponding turning function. Note that there are two degrees of freedom in the turning functions: The choice of the the origin, **O**, and the directionality of the reference axis.

the traversal starts corresponds to the 0 position on the turning function diagram and is called the *origin*. The origin is also chosen arbitrarily. The above definition is general enough to apply to any type of shape. For closed shapes, the function repeats itself shifted up by 2π in each successive period, which is equal to the length of the boundary. In other words, the first derivative of the turning function of a closed shape is periodic. Figure 1 shows the turning function of a simple polygon. To support scale invariance, the length of one cycle of the traversal around a polygon is normalized to 1 in the turning function.

Note that rotating the reference axis by θ degrees shifts the entire turning function by θ up or down, depending on the direction of rotation. The same effect occurs if the polygon is rotated around a point on the plane. Sliding the polygon's origin along the perimeter effectively shifts the turning function left or right by the amount. We will see in the next section that these two parameters, namely the choice of reference axis and the position of the origin along the boundary, can be tuned to accomplish a "maximal" match between two polygons' turning functions. Their distance is then computed at the maximally matched configuration.

3.2 Similarity Metric Function

To compare two polygons whose turning functions are $f(s)$ and $g(s)$, $s \in [0, 1]$, we compute a function of the area *between* the functions, minimized over all choices of reference axis (angle θ) and origin for the second polygon (position along the boundary t), while keeping the first one's reference axis and origin fixed. The distance metric $D(f, g)$ between two turning functions $f(.)$ and $g(.)$ is

$$h(\theta, t) = \int_0^1 (f(s+t) - g(s) + \theta)^2 ds$$

$$D(f, g) = \min_{\theta,\ t} \sqrt{h(\theta, t)}$$

An important property of this metric function is that it satisfies the triangular inequality, i.e., $D(f, g) \leq D(f, h) +$

$D(h, g)$. Modifying the orientation of the reference axis corresponds to rotating the polygon or, equivalently, shifting the turning function up or down. The optimal angle θ^*, as a function of the origin position can be computed in constant time

$$\theta^* = -2\pi t + \int_0^1 (g(s) - f(s))ds$$

This is evident immediately by taking the derivative of $h(\theta, t)$ with respect to θ, setting the derivative to 0 and solving for θ. Minimizing $h(\theta^*, t)$ over all choices of t takes more than constant time though. In particular, $h(\theta^*, t)$ reaches a minimum or a maximum for shifts of $g(s+t)$ that result in one breakpoint of g coinciding with a breakpoint of f. A *breakpoint* is a point of discontinuity of the turning function, and corresponds to a vertex along the perimeter. Between breakpoint coincidences, $h(\theta^*, t)$ varies linearly with t. The naive way to compute D is by calculating the integral over all possible mn choices of shifts that produce a breakpoint coincidence, where m and n are the number of vertices in the two polygons. Since computation of the integral takes $O(m+n)$ time, the time complexity is $O((m+n)mn)$, or $O(n^3)$ if $m = n$. A faster algorithm, also presented at [1], making this time $O(n^2 \log n)$ is used in the *PAMIS* prototype.

4 Parallel Nearest-Neighbor Search

Given the turning function representation for polygonal shapes, *PAMIS* organizes the shapes in the database in a specific form of search tree called the *Vantage-Point Tree* (VPT) [3], to speed up the content-based "like-this" queries. Moreover, the search tree is striped across a network of workstations to exploit the computation and I/O parallelism inherent in the nearest-neighbor search process.

4.1 Vantage-Point Tree

A VPT organizes the database elements into a tree similar to a B-tree. To construct a VPT, one first chooses a database element, say V, as the vantage point, computes the distance metrics between every other database element and V, and then classifies the database elements into M approximately equal-sized partitions (in terms of number of database elements) based on the distance metric values, as shown in Figure 2. Essentially all the database elements are re-mapped onto a one-dimensional axis based on their distances with respect to the vantage point. Each partition i is characterized by its two boundary distance values, $low[i]$ and $high[i]$, where $low[i] = high[i-1], i = 2, \ldots, M$. This process repeats for each of the partitions recursively until the number of database elements is small enough to fit into a disk page. A copy of the VP polygon is stored at each internal

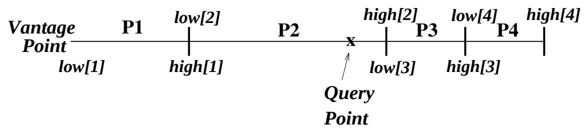

Figure 2. Mapping and partitioning of the database elements into disjoint subsets in the distance metric space based on their distances with respect to the vantage point. Each of the subsets such as P_1 corresponds to a tree branch.

node for efficiency. The leaf nodes usually contain most of the database elements. In *PAMIS*, the database consists of polygons, whose representation takes a significant amount of storage space, and therefore each leaf node actually contains only a small number of polygons.

When a query polygon q is presented to a VPT rooted at v, the system first computes the distance $D(q, v)$ between q and v. Based on this value, the best-case and worst-case distance estimates between each partition and q can be derived as follows:

$$best\_case[i] = \begin{cases} D(q,v) - high[i] & \text{if} \quad D(q,v) \geq high[i] \\ 0 \quad \text{if} \quad high[i] & \geq D(q,v) \geq low[i] \\ low[i] - D(q,v) & \text{if} \quad low[i] > D(q,v) \end{cases}$$

$$worst\_case[i] = D(q,v) + high[i]$$

With these estimates, a partition will be pruned if and only if its best-case estimate is larger than δ. The value of δ is set to the distance metric between the currently known nearest neighbor and q and is updated when that distance changes. Intuitively, δ represents the maximum threshold that users[2] are willing to accept as the distance between q and its nearest neighbor. In other words, if it turns out that the distance between q and its nearest neighbor is larger than δ, users are not interested. The same search procedure is applied recursively at each level of the VPT to determine which branches at that subtree can be pruned until reaching the leaf nodes. At that point, a sequential search through the polygons at the leaf nodes is required to complete the search.

4.2 Database Organization

The VP tree is stored on the disks in the same way as a B-tree. However, in this case, each internal node n stores a vantage point V^k, a set of distance values, $D_i^k, i = 1, \ldots, B^k - 1, D_0^k = 0$ (implicitly assumed), B^k pointers to lower-level VPT nodes, and B^k counters that store the number of database elements under each child branch. B^k is called the *branching factor* and D_i^k is the i-th median among the distance values between the vantage point

[2] However, users are not responsible for choosing the δ value explicitly.

and those database elements under this node. Thus D_i^k's are used to separate these database elements into approximately equal-numbered partitions, $P_i^k, i = 0, \ldots, B^k - 1$, each of which becomes a child branch from this internal node. A database element e belongs to the j-th partition P_j^k if and only if $D_{j+1}^k > D(V^k, e) \geq D_j^k$. The leaves contain the actual polygons in the database, together with a small amount of administrative information.

4.3 Parallel Implementation

The nodes of the vantage-point tree described in the previous section are striped across a network of workstations. *PAMIS* distributes the VPT using a static cyclic interleaving technique. The software architecture consists of X *compute* processes and one distinct *coordinator* process. The *coordinator* maintains two data structures: the *results table* and the *load table*. The *results table* stores the IDs of the best N nearest neighbors found so far, their associated VPT node IDs, their distances with respect to the query point and the owning processor IDs. The distance value associated with the worst of the best N nearest neighbors is δ. The *load table* contains an entry for each *compute* process, which consists of a *busy flag* to indicate whether the corresponding processor is busy or idle, and a counter that records the message balance of the processor, i.e., the difference between the number of request messages transmitted and the number of request messages received. The latter table is used by the *coordinator* to determine when the entire search process should be terminated: when all processors are idle and the sum of the differences in the *load table* is zero, i.e., when all *compute* processes have no more work to do and there is no request messages still floating on the network.

Each *compute* process maintains the most recent δ and a *request queue*, in which the incoming requests for exploring the internal or leaf nodes of the VPT are stored. The requests are stored in the queue according to certain scheduling discipline. In *PAMIS*, we have four alternatives: smallest-best-case first, smallest-worst-case first, deepest-then-smallest-best-case and deepest-then-smallest-worst-case first. The first two are based on the best-case and worst-case estimates of the nodes specified in the requests, respectively. The idea behind the former is to explore the most promising nodes first, while the latter one tries to explore the nodes with the smallest worst-case value first and thus decrease the value of δ as soon as possible. The third and fourth are based on the levels of the nodes. The deeper the node, the higher priority it gets in being scheduled. The rationale is that as the nodes get closer to the leaves, the more specific they become and will produce a smaller distance value than the current estimate derived from their ancestors that are already explored, thus providing better pruning power.

After a *compute* process explores a VPT node, it sorts the actual distance values (if at a leaf node) and/or the worst-case estimates of the children branches (if an internal node), and retains those that are better than the local δ. Only the first N of these will be sent to the *coordinator* to update the *results* table. The *coordinator* processes them in the way described earlier, possibly lowering δ. In the case that δ is indeed lowered, a messages is sent by the *coordinator* to all processors, informing them of the new δ. In order to prune as soon as possible, a processor shortcircuits this updating by broadcasting directly to all processors a new distance that is lower than its local δ, which may be out of date because of network lag and processing overhead at the *coordinator*. Each processor that receives a notification of a new value of δ adopts it only if it is smaller that its local δ (which may be smaller than the incoming one if it had been updated in the meantime by another broadcast). Messages containing new δ values are processed immediately in order to prune the search space as quickly as possible.

Two δ adjustment algorithms are implemented in the prototype and compared under various conditions. The first, called the *pessimistic* algorithm, starts with a large δ value and lowers it as the computation proceeds. The second one, called the *optimistic* algorithm, starts with a small δ value, progressively increases it, and eventually reaches the ultimate δ value. The algorithms are described in the next two sections.

4.4 Pessimistic Strategy

In the pessimistic algorithm the *results table* is initially filled with null entries with all the distance values set to infinity. The δ values is also infinity, taken from the N-th table entry. At the end of the computation, the *results table* contains the N nearest neighbors to the query point in the database. This table is kept sorted by the *coordinator* in ascending distance order at all times. Every time a polygon whose distance to the query point is smaller than δ is discovered by a *compute* process, the polygon ID, the owning processor ID, and the distance value is send to the *coordinator*. The *coordinator*, upon receipt of such a message, inserts the polygon into the *results table*, pushing down entries of higher distance values and potentially lowering the value of δ, which then gets broadcast to all *compute* processes.

To speed up the fill-up process of the *results table*, we exploit the worst-case estimates of the branches. For example, suppose there are two branches from the VPT root. After the root node has been explored, the branch containing 30 polygons has a worst-case estimate of 7, while the other with 50 polygons has a worst-case estimate of 3. Assuming that only 5 nearest neighbors are needed, then the *results table* should contain the root vantage point plus four estimate entries whose distance values are 3, rather than four null entries whose distance value fields are infinity, as would be the case

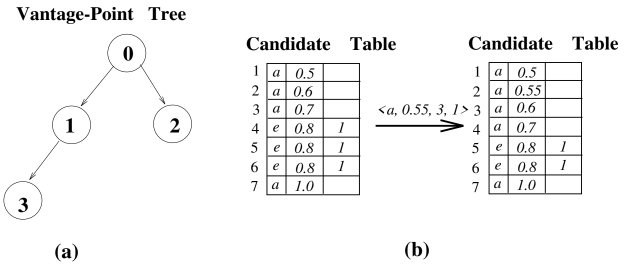

Figure 3. (a) shows a portion of the vantage-point tree. When Node 3 is explored, a new nearest neighbor candidate with a distance value 0.55 is sent to the coordinator to be integrated into the results table. (b) The left is the results table before the insertion of the new candidate, while the right is after. The first column of the results table indicates the type of the entry: **a** for actual entries and **e** for estimate entries. The second column represents the distance value and the third denotes the ID of the entry's associated VPT node, if it is an estimate entry.

when only one polygon, i.e., the vantage-point polygon at the root, has been explored.

There are three types of entries in the *results table*: *null* entries, *actual* entries that store real polygons, and *estimate* entries that store worst-case estimates from certain VPT node. Because of the existence of estimate entries, the replacement of the *results* table becomes a little bit more complicated. When a new nearest-neighbor candidate arrives at the *coordinator*, it comes with its entry type, a distance value[3], the ID of the node which it is associated with and the ID of the parent of its associated node. The *coordinator* replaces an estimate entry whose associated node ID matches the new candidate's parent Node ID, if there is any. Otherwise, it is appended to the end of the results table. Then the entire results table gets re-sorted and only the top N results are kept. This replacement mechanism ensures that all estimate entries will be eventually instantiated to actual entries. For example, in Figure 3, a new actual candidate that has a distance value of 0.55, and is associated with VPT Node 3, whose parent Node ID is 1, is to be inserted into the *results* table. Because the fourth entry is an estimate entry in the results table whose associated Node ID is 1, the new candidate will replace the fourth entry. Then the entire results table is re-sorted to arrive at the table in Figure 3(b).

Note that the approach described above is a pessimistic one. The computation starts with the worst case δ and tries to minimize it along the traversal process. The other possibility would be to optimistically start with a small initial δ. That algorithm is described in the next section.

[3] This value can be either an actual distance value or a worst-case estimate.

4.5 Optimistic Strategy

The optimistic algorithm requires multiple iterations of tree traversal, each with a different δ. Along the process, the algorithm gradually increases δ until it overshoots, and then automatically switches to the pessimistic algorithm's behavior, with the value of δ decreasing towards the ultimate value of the N-th nearest neighbor's distance from the query polygon. In each iteration of the *optimistic* phase (δ increasing), there are not enough polygons found during the search process to populate all N entries of the *results* table, so the value of δ never decreases. When δ overshoots, more than N nearest neighbors can be found to populate the table and δ will start decreasing, as in the pessimistic algorithm.

The formula used in updating the value of δ in each iteration is

$$\delta_{i+1} = \delta i + \Delta_i$$

where

$$\Delta_{i+1} = \Delta_i + \frac{\Delta_i}{2}\left(\frac{1}{1+|F_i - E_i|/E_i} - 1\right) * \mathtt{sgn}(F_i - E_i).$$

F_i is the number of additional neighbors found in the i-th iteration only, and E_i the number of expected neighbors for that iteration. The number of expected neighbors in any iteration is set to half the remaining neighbors yet to be found. Note that the above formula treats $F_i - E_i$ as an error value and tries to minimize it. It is also designed to guard against overshoots due to large step changes when the error is great, i.e., the fraction converges to $+1$ or -1 when the error goes to $-\infty$ and ∞, respectively. However, as will be shown later in Section 5.5, the performance penalty associated with an overshoot is not as high as originally expected. Therefore a more aggressive update strategy should improve the performance. The algorithm saves the results of the work already by marking the nodes that are already explored in earlier iterations and save the intermediate results.

5 Performance Evaluation

5.1 Experiment Setup

The current implementation of *PAMIS* is built on top of eleven Pentium-90MHz PCs. One of them (the coordinator) has 32MB of memory, and the rest have 16MB. The processes' memory needs are modest enough so that they never need to swap. The workstations are linked via a 100Mbit/s Fast Ethernet and the operating system used is FreeBSD version 2.1.0. The kernel is modified to provide low-latency communications among workstations using the fast communication facilities of LOCUST [4], a distributed shared virtual memory system developed in our group. The test database used consists of the outlines of the 52 characters

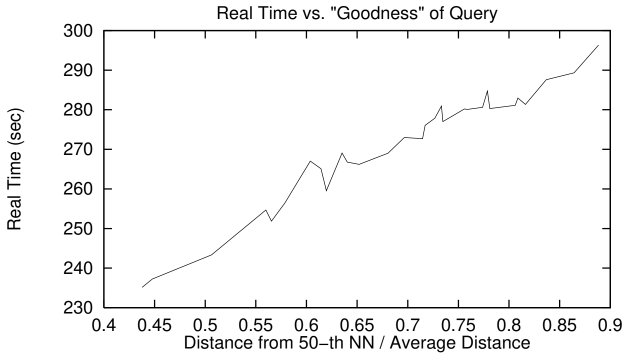

Figure 4. The correlation between the "goodness" of a query and its execution time. The experiment is performed on 10 processors with the best-first scheduling strategy to search for the 50 nearest neighbors. The database node size used is 4,096 bytes.

of the English alphabet (upper and lower case). Each letter is represented by a 50-vertex polygon, and has 1000 polygonal variations of it in the database, each perturbed by a random amount from a hand-drawn template. The perturbation method used changes the position of each vertex in the polygon according to a uniform distribution within a square centered at the original vertex position. The side of the square is 2% of the minimum of the $x-$ and $y-$spans of the original polygon that represents the letter. For each result reported, we measure the total wall-clock time on unloaded systems, the time spent in computing the polygon distance function, the time doing disk I/O, the time spent on communication and the idle time. The difference between the total time and the sum of (useful) computation, I/O and communication, is the overhead time spend on maintaining the *compute* process' scheduler queue and determining the next node to explore according to the chosen scheduling strategy.

5.2 Goodness of Queries

Unlike traditional database systems that store only alphanumeric texts, the performance of an image query is significantly affected by the "goodness" of the query. Intuitively, a query is good if the distance between it and its nearest neighbor in the database is small. An image index system such as *PAMIS* is meant to support "good" queries. A bad query sometimes will force the indexer to search almost every element of the database, thus defeating the whole purpose of indexing. We use the expression

$$\frac{D(q, N - \mathrm{th}Nearest\_Neighbor(q))}{AVG_{x \in Database} \ \ D(q, x)}$$

to quantitatively measure the "goodness" of a query. This expression measures how the nearest neighbor looks to the query point compared to others in the database. The closer the nearest neighbor looks, the faster it takes to locate it, because of the increased likelihood of using a smaller δ during the search process. Therefore, the smaller value the expression produces, the better a query is. Figure 4 shows that there is a positive correlation between the "goodness" expression value of a query and its overall execution time. Because there are other factors that affect the query performance, the proposed expression is just a heuristic to predict how "good" a query point is. For the rest of this section, unless otherwise noted, we will report only performance results from good queries, i.e., polygons that are small variations of polygons in the database. Also, the node size that will be used will be 4,096 bytes, and the smallest-best first scheduling strategy will be used.

5.3 Delay Measurements

In a traditional database system, I/O usually plays a significant role in the overall execution time. As a result, it is useful to consider both the number of polygons touched, which corresponds to the computation time, and the number of VPT nodes, which corresponds to the I/O time. Because *PAMIS* is a parallel implementation, there are also communications delays and other overheads such as interrupt processing. Table 1 shows the detailed breakdown of the overall execution time under different numbers of processors. In the multi-processor case, only the statistics of the slowest process is shown. The $Overhead$ column is computed by subtracting the I/O, communications, and computation from the total execution time. It shows that the turning function distance metric computation dominates the entire search time in all cases. The time taken to compute the distance between two 50-vertex polygons on a 90MHz Pentium is approximately 53 msec. Despite the use of $O(n^2 \log n)$ algorithm rather than the naive $O(n^3)$ one to compute the distance function, the problem remains CPU-bound rather than I/O-bound.

No. of Proc's	Max. No. of polygons touched	Total No. of polygons touched	Execution Time (sec)
1	19753	19753	1123.4
3	6587	19755	379.6
5	3972	19757	228.5
7	2848	19759	165.0
10	1991	19762	115.7

Table 2. The number of polygons touched and VPT nodes accessed during the search for the 50 nearest neighbors. The node size is 4,096 bytes and the best-first scheduling strategy is used. The query polygon is the letter D.

Table 2 shows the relationship between the number of

Pessimistic Algorithm					
Processors	**I/O (%)**	**Commun. (%)**	**Idle (%)**	**Overhead (%)**	**Computation (%)**
1	4.91	0.01	0.38	1.13	93.57
3	4.47	0.09	1.22	1.06	93.16
5	4.01	0.10	2.10	1.09	92.70
7	2.40	0.13	3.89	1.19	92.39
10	0.64	0.16	5.59	1.33	92.28
Optimistic Algorithm					
Processors	**I/O (%)**	**Commun. (%)**	**Idle (%)**	**Overhead (%)**	**Computation (%)**
1	5.23	0.02	0.37	3.41	90.97
3	4.61	0.31	1.87	4.19	89.02
5	4.33	0.37	4.03	3.96	87.13
7	4.05	0.45	6.09	3.81	85.60
10	1.76	0.49	11.24	3.45	83.06

Table 1. Detailed breakdown of the search time for the first 50 nearest neighbors. Node size is 4,096 bytes and the best-first scheduling strategy is used with both algorithms. The query polygon is letter D.

polygons touched, i.e., the number of distance metric functions called during the search, and the overall execution time. The second column is the maximum number of polygons touched by any single processor, and correlates very well with the overall execution time because it essentially represents the amount of work that the slowest process has to do. The third column represents the total number of polygons touched by all processors. The fact that the total number of polygons touched is not very different in different processor configurations means that most of the parallelism exploited is not speculative. Because the maximum number of polygons touched by any single processor is very close to the total number of polygons touched divided by the number of processors, there doesn't seem to be any serious load imbalance problem. This suggests that the cyclic interleaving data layout strategy used in *PAMIS* is reasonably efficient. For a good query that requires 50 nearest neighbors, the efficiency of the PAMIS indexing scheme is better than 39%, i.e., it touches fewer than 39% of the whole database to answer the query.

Processors	1	3	7	10
Node size 2048	900.3	302.7	131.7	92.1
Node size 4096	1121.4	375.4	166.0	115.4

Table 3. The impact of the block size or VPT node size of the database on the query performance. The experiment is performed with the pessimistic algorithm and the best-first scheduling strategy to search for the first 50 nearest neighbors and the query polygon is the letter D. The results measure real time elapsed and are in seconds.

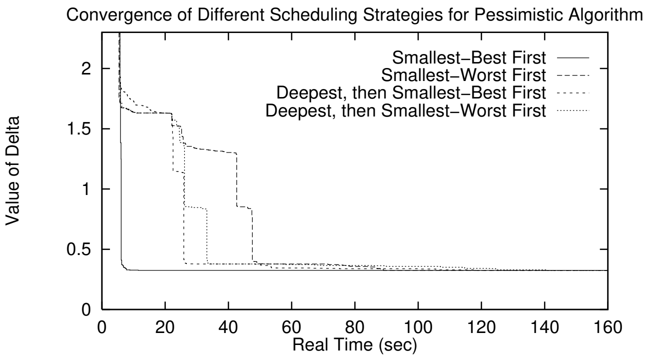

Figure 5. The impact of the scheduling strategy used in each compute process on the query performance for the pessimistic algorithm. The query asks for the 50 nearest neighbors to the polygon representing letter D.

5.4 Impact of VPT Node Size

To investigate the performance impact of the structure of the VP tree, we constructed two databases with VPT node sizes of 2,048 and 4,096 bytes. Table 3 shows the query performance decreases as the VPT node size increases. It seems that because of the dominance of the distance computation time, it is better to have a lean and deep tree rather than a wide and shallow tree. The reduction in the amount of sequential search at the leaf nodes outweigh the increase in I/O time due to smaller block sizes.

5.5 Impact of Scheduling Strategy

As mentioned in Section 4.3, there are four scheduling strategies that the *compute* process uses in determining the

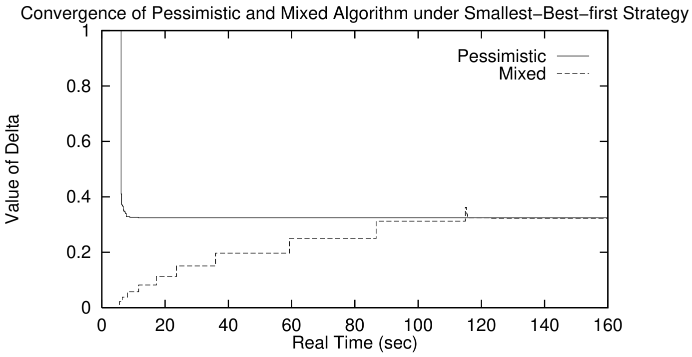

Figure 6. Comparison of the pessimistic and the optimistic algorithms under the smallest-best -first scheduling strategy. The query polygon is the letter D.

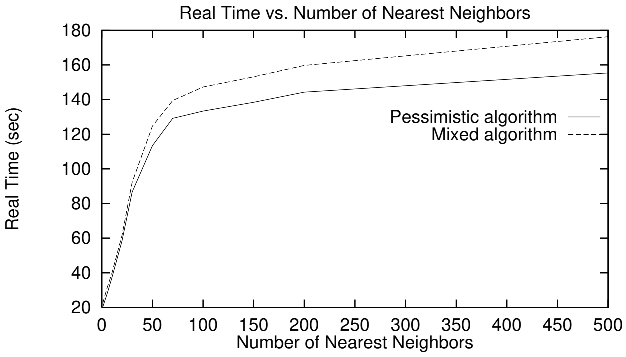

Figure 7. The search performance vs. the number of nearest neighbors required by the query. The experiment is performed with the best-first scheduling strategy with 10 *compute* processors. The query polygon is letter D.

service order of the entries in the request queue: best-case ascending, worst-case ascending, deepest-then-smallest-best and deepest-then-smallest-worst first. Figure 5 shows that among the four, the best-case ascending scheduling strategy performs significantly better than the other three in the case of the pessimistic algorithm. Scheduling strategies do not make much difference for the optimistic algorithm and the results are thus omitted. It is worthwhile to note that in all our experiments the numbers of polygons touched with the pessimistic algorithm with best-case scheduling, and the optimistic algorithm are identical, and are in fact minimal. That is, running the experiment with the initial δ set to the ultimate 50-th nearest neighbor distance would explore exactly the same number of polygons. This result is surprising because it suggests that the pessimistic algorithm with the right scheduling strategy can perform as good as the optimistic algorithm.

A comparison between the pessimistic and optimistic algorithms, under the best-case scheduling strategy, is shown in 6. In all our experiments, the pessimistic algorithm quickly lowers the value of δ from infinity to a value very close to the ultimate one. This fact explains why it explores only the minimal number of polygons. The optimistic one slowly increases δ until it overshoots, and then quickly converges to the ultimate value. It should be noted that the optimistic algorithm is always expected to perform better than the pessimistic one in theory, barring the overhead of multiple iterations, because it starts its pessimistic-like behavior from a δ value that is much smaller than infinity. In practice though it is slightly slower than the pessimistic one because of the additional per-iteration overhead and larger communication overhead. Therefore it would be advantageous to execute as few iterations as possible, and an aggressive policy for increasing δ between iterations, such as increasing the expected percentage of nearest neighbors found in each iteration, would be preferable.

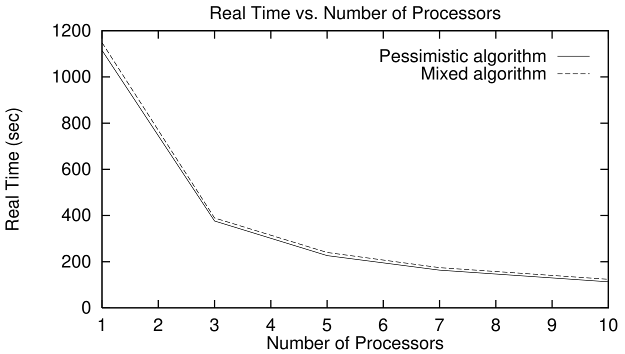

Figure 8. The search performance vs. the number of *compute* processors used. Both strategies are optimal. The slight difference is due to the increased overhead of the optimistic algorithm. The experiment is performed with the best-first scheduling strategy. The query polygon is letter D.

5.6 Impact of Query Size

Because *PAMIS* is mainly used to prune undesirable images based on the query, users usually request more than one nearest neighbor and browse through them to identify the desirable ones. Figure 7 shows the relationship between the execution time and the number of nearest neighbors required by the queries. As expected, the more nearest neighbors required, the larger the overall execution time. This is because the minimum δ will increase when the number of nearest neighbors requested increases. Moreover, the rate of increase in the execution time slows down as the number of nearest neighbors required increases. That is, the execution time grows slower than linear to the number of nearest neighbors requested. Again the difference between the optimistic and pessimistic algorithms is mainly due to the per-iteration overhead.

The VPT algorithm's scaling performance under different numbers of *compute* processors follows, as expected, the law of diminishing returns. However, the performance does not seem to have saturated at the 10-processor configuration, because the parallelization overhead is still relatively insignificant compared to the cost of distance function computation, as shown in Table 1.

6 Conclusion

This paper describes the design, implementation, and evaluation of a parallel index system called *PAMIS*, specifically for 2D polygonal image shape databases. *PAMIS* employs the turning function representation to represent polygonal image shapes, and uses the accumulative vantage-point tree algorithm to index image shape databases. We have successfully implemented the first *PAMIS* prototype on a network of Pentium workstations, and carried out a comprehensive performance evaluation of the prototype against a large-scale test database, which to our knowledge is the first such experiments whose results have ever been reported in the area of image shape database. We examine the performance impacts of such architectural parameters as VPT node size, scheduling strategies, and the detailed breakdown of the query performance cost. Because of the dominance of the polygon distance function computation, it is preferable to make the VPT node size as small as possible in order to have a lean and long VPT structure. We also found that the best-case scheduling strategy performs the best among the scheduling strategies considered, and the pessimistic algorithm with best-case scheduling performs as well as the optimistic one. The performance of the VPT algorithm scales reasonably well with the number of processors, and load imbalance doesn't appear to cause performance problems. Overall, the indexing efficiency of *PAMIS* is better than 39% for a good query that request 50 nearest neighbors.

We are currently pursuing the notion of *multi-resolution indexing*, where different degrees of approximations to the image shapes are stored and indexed. Because distance function computation accounts for the dominant portion of the query execution time, and the distance function computation complexity depends on the number of vertices in the polygons, it seems appealing to use low-resolution indices during the early pruning stage, and only resort to high-resolution indices when reaching the leaves of the search tree. Along a similar line, we are also interested in experimenting with spline-based rather than piecewise-linear representations to approximate smooth shapes, hoping to reduce the number of vertices needed and thus the distance computation time. Also, we will experiment the VPT algorithm with other feature extraction techniques such as color histograms or wavelet coefficients.

Acknowledgement

This research is supported by an NSF Career Award MIP9502067 and a contract 95F138600000 from Community Management Staff's Massive Digital Data System Program.

Reference

[1] E. Arkin, et al., "An efficiently computable metric for comparing polygonal shapes," IEEE Transactions on Pattern Analysis and Machine Intelligence, (March 1991) vol.13, no.3, p. 209-16.

[2] S. Arya, D. Mount, "Approximate Nearest Neighbor Queries in Fixed Dimensions," Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, p. 271-280, Orlando, Fla., 1992.

[3] T. Chiueh, "Content-based image indexing," In Proceedings of VLDB '94, p. 582-593, Santiago, Chile, September, 1994.

[4] T. Chiueh, Manish Verma, "A compiler-directed distributed shared memory system," In Proceedings pf the 9-th ACM Supercomputing Conference, Barcelona, Spain, July 1995.

[5] C. Faloutsos et al., "Efficient and effective querying by image content," Technical Report RJ9453, IBM Research, San Jose, California, 1993.

[6] M. Flicker, et al., "Query by image and video content: the QBIC system," IEEE Computer, p. 23-32, Vol.28, No.9, September, 1995.

[7] Jagadish, H.V. "A retrieval technique for similar shapes." 1991 ACM SIGMOD International Conference on Management of Data, Denver, CO, USA, 29-31 May 1991. SIGMOD RECORD (June 1991) vol.20, no.2, p. 208-17.

[8] R. Mehrotra, J. Gary, "Feature-based retrieval of similar shapes," International Conference on Data Engineering, '93, pp. 108-115.

[9] R. Mehrotra, J. Gary, "Similar-shape retrieval in shape data management," IEEE Computer, p. 57-62, Vol.28, No.9, September, 1995.

[10] P. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 311-321, Orlando, Fla., 1992.