Pictorial Queries by Image Similarity

Aya Soffer *
Computer Science and EE Department
University of Maryland Baltimore County and
CESDIS, Goddard Space Flight Center and
Center for Automation Research
University of Maryland at College Park
E-mail: soffer@cs.umbc.edu

Hanan Samet †
Computer Science Department and
Center for Automation Research and
Institute for Advanced Computer Science
University of Maryland at College Park
College Park, Maryland 20742
E-mail: hjs@umiacs.umd.edu

Abstract

A method for specifying pictorial queries to an image database is introduced. A pictorial query specification consists of a query image, and a similarity level that specifies the required extent of similarity between the query image and database images that are to be retrieved. The query image is constructed by positioning objects so that the desired locational and spatial constraints hold. Two image similarity factors are considered: (1) contextual similarity*: how well does the content of one image match that of another. (2)* spatial similarity*: the relative locations of the matching symbols in the two images. Algorithms for retrieving all database images that conform to a given pictorial query specification are presented and compared.*

1. Introduction

A basic requirement of an image database is the ability to query the database pictorially. The most common method of doing this is querying via an example image. The problem with this method is that in an image database we are usually not looking for an exact match. Instead, the goal is to find images that are similar to a given query image. The main issue is how to determine if two images are similar and whether the similarity criteria that is used by the database system matches the user's notion of similarity.

In this paper we introduce a method for specifying queries to an image database pictorially that enables the user to indicate the type of similarity between the query image and the database images that is required. The query image is constructed by positioning objects so that the desired locational and spatial constraints hold. Two image similarity factors are considered: (1) *contextual similarity*: how well does the content of one image match that of another. For example, should the database image contain all of the objects in the query image or may it just contain some of these objects. (2) *spatial similarity*: the relative locations of the matching symbols in the two images. We refer to the information regarding the location of the objects and the spatial relation between these objects as *spatial-locational information* and *spatial-relational information*, respectively. By specifying the desired contextual and spatial similarity levels along with the query image, users can specify the extent of the required similarity. All images that are similar to the query image under the specified image similarity level are retrieved.

Very few commercial systems support retrieval of images by pictorial specification. The Illustra object-relational DBMS [9] provides a library for storing and managing image data. IBM's UltiMedia Manager offers content-based image query (based on QBIC [4] technology) in conjunction with standard search. In both cases image similarity retrieval is based on similarity in color and texture between the query image and the database images. However, the results are highly subjective and there is no intuitive metric that can be used to decide whether the result images are in fact those that are most similar to the query image. Some prototype research systems such as Photobook [6] and FINDIT [8] also employ such methods.

Numerous prototype research IDMS's have been reported in recent years that address the issue of how to index tagged images (images in which the objects have already been recognized and tagged with their semantic meaning) in order to support retrieval by image similarity [1, 2]. These systems are mainly concerned with spatial-relational information and do not deal with spatial-locational information. The most common data structure that is used for this purpose is the *2-D string* and its variants [1]. Another data structure called the *spatial orientation graph* is introduced in [2] and used

*The support of USRA/CESDIS and NASA Goddard Space Flight Center is gratefully acknowledged.

†The support of the National Science Foundation under Grant IRI-92-16970 is gratefully acknowledged.

for spatial similarity based retrieval of symbolic images. In [5] a pictorial query-by-example (PQBE) language that provides more expressive power is presented. The distance between objects is ignored in PQBE as it was in all other methods dealing with spatial similarity.

In contrast, our approach handles queries that deal with both spatial-relational and spatial-locational data, as well as contextual information. Thus we can deal with the distance between objects. In addition, as part of the pictorial specification, the user indicates the degree of desired similarity, and thus the results are not subjective. We have applied these methods to a symbolic image database developed by us (SYMIDB) [7]. This database converts images from a physical to logical representation, and thus we do not assume tagged images. This conversion is applicable to images where the set of objects that may appear in them is known a priori, where the geometric shapes of these objects are relatively primitive, and which convey symbolic information. Examples of such images include maps, engineering drawings, floor plans, etc.

2. Image Similarity

In this section we define the notion of image similarity in the domain of symbolic images. A *symbol* is a group of connected pixels that together have some common semantic meaning. A *class* is a group of symbols that all have the same semantic meaning.

Let I_1 and I_2 be two images. Two factors are considered in defining image similarity between I_1 and I_2. (1) *contextual similarity*: how well does the content of one image match the other (e.g., do the symbols in one image appear in the other image). (2) *spatial similarity*: the relative locations of the matching symbols in the two images. The goal is to use I_2 as a query image, and then retrieve all database images I_1, that are similar to it according to the specified contextual and spatial similarity

Four levels of contextual similarity are defined:

1. The images have exactly the same symbols.
2. Any symbol found in I_2 exists in I_1 (I_1 includes I_2).
3. Any symbol found in I_1 exists in I_2 (I_2 includes I_1).
4. There is at least one symbol common to both I_1 and I_2.

The spatial similarity level enables specification of the required database images in terms of minimum and maximum distance between symbols, and their relative locations in a directional sense. Five levels of spatial similarity are defined.

1. The matching symbols of I_1 and I_2 are in the exact same locations in both images.
2. Matching symbols have the same relations, and the distance between them is bounded from below by some given L and bounded from above by the distance between the symbols in I_2. By default $L = 0$.
3. The distance between the matching symbols may vary, but the relation between them must be the same.
4. The relation between the matching symbols may vary, and the distance between them is bounded from below by some given L and bounded from above by the distance between the symbols in I_2. By default $L = 0$.
5. The distance and the relation between the matching symbols may vary (i.e., no spatial constraints).

The total similarity between I_1 and I_2 is defined by taking the combination of the two similarity factors. For example, $I_1 \equiv_{2,3} I_2$ denotes that the contextual similarity of the two images is at level 2, and that the spatial similarity between these symbols is at level 3. That is, all symbols in I_2 appear in I_1, the location of the symbols and the distance between them may vary, but the relation between them is the same.

3. Pictorial Query Specification

To specify queries pictorially, the user creates an image containing the required symbols positioned such that the desired spatial-locational and spatial-relational constraints hold. The user must also specify the required image similarity level to achieve the desired spatial constraints.

Figure 1 demonstrates the use of different spatial similarity levels for query specification. Query Q1 requests all images that contain a "scenic view" (and maybe other symbols). Query Q2 requests images that contain a "scenic view" within 5 miles of a "picnic" site. Query Q3 requests images with a "site of interest" and any symbol within 2 miles of these sites of interest. Query Q4 requests images that contain an "airfield" northeast of a "beach". [1]

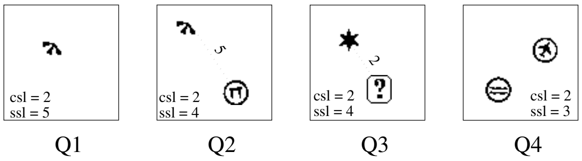

Figure 1: Pictorial queries with varying spatial similarity levels. "csl" and "ssl" denote contextual and spatial similarity levels, respectively. The "question mark" denotes a wild card (i.e., any symbol).

By varying the contextual similarity level, more complex pictorial queries can be specified as shown in Figure 2.

[1] Note that the dotted lines with the distance that appear in the query images in Figure 1 are only used to denote the distance between symbols in the figure; they are not actually part of the query image. The query image only contains symbols. The distance (and relative directions) between the symbols is specified implicitly in the query image QI by the actual distance (and relative direction) between the symbols in QI.

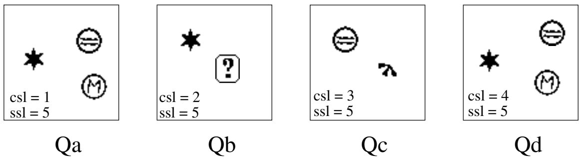

Figure 2: Queries demonstrating the use of different contextual similarity levels.

Query Qa requests all images that contain a "site of interest", "beach", "museum", and no other symbols. Query Qb requests all images that contain a "site of interest" and at least one other symbol (there may be more). Query Qc requests all images that contain a "beach" or a "scenic view" (an image may contain both) but no other symbols. Query Qd requests all images that contain a "site of interest", or a "beach", or a "museum" (an image may contain all of them as well as other symbols). No spatial constraints are specified in these queries.

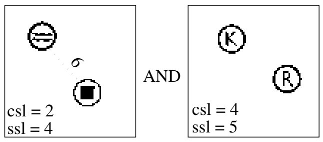

Figure 3: A query to "display all images with a hotel within 6 miles of a beach and with a cafe or a restaurant".

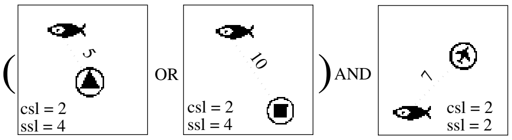

Figure 4: A query to "display all images with a camping site within 5 miles of a fishing site OR with a hotel within 10 miles of a fishing site AND with an airfield northeast of and within 7 miles of the fishing site".

More complex queries may be specified by combining query images with "AND" and "OR" operators. See Figure 3 and Figure 4 for examples of such queries.

4. Pictorial Query Processing

We present five different algorithms for the function *GetSimilarImages* that retrieves all database images that conform to a given pictorial query specification. These algorithms vary in how they use the indexes on the contextual and spatial information. To execute these algorithms efficiently, the image database must have indexes that enable the following operations: (i) retrieve all images that contain symbols of a given class; (ii) retrieve all symbols in a given image; (iii) retrieve all symbols within a given distance from a given point. In SYMIDB the first two indices are realized with a B-tree. Retrieval of all symbols within a given distance from a given point is achieved by use of an index on the locations of the set of all symbols in all of the images. This index is implemented using a PMR quadtree for points [3].

4.1. Algorithms

The input to *GetSimilarImages* is the query image (QI), the contextual similarity level (csl), and the spatial similarity level (ssl). The output of *GetSimilarImages* is a set of database images DI such that $DI \equiv_{csl,ssl} QI$. The set of images DI that is output by *GetSimilarImages* is ranked in decreasing order of the average of the certainty values of those elements of DI that are in QI.

In the algorithms presented in this section we allow only one instance of each class in the query image as well as in the database images. We briefly discuss how to deal with other cases. For the purpose of simplicity, we assume that in all queries that involve spatial distance constraints, L, the lower bound for the distance allowed between symbols in the database image, is 0. That is, $0 \leq dist(s_i, s_j) \leq dist(s_k, s_l)$, where s_i and s_j are database image symbols and s_k and s_l are query image symbols, respectively.

In the following algorithms, QI is the logical image representation of the query image. The logical image representation LI of an image I, is a list of elements for each symbol $s \in I$. Each element of LI is of the form: $\{(C, certainty)(x, y)\}$ where C is the classification of s, (x, y) is the location of s in I, and $0 < certainty \leq 1$ indicates the certainty that $s \in C$. The classification, C, of a specific element $el \in LI$ is denoted by $C(el)$. The location, (x, y), of a specific element $el \in LI$ is denoted by $loc(el)$. The contextual and spatial similarity levels are denoted by csl and ssl, respectively. $|I|$ denotes the number of elements in the logical representation of I (i.e., its cardinality).

Figure 5 summarizes *GetSimilarImages1* which is the simplest and most general. We first get the image id's of all database images that contain each symbol of the query image separately (using the index on `class`). This is followed by a union operation if we are interested in the images that contain any of these symbols (csl = 3 or 4), or by an intersection operation if we are interested in the images that contain all of these symbols (csl = 1 or 2). Next, remove images that contain extraneous symbols from the candidate-image set. Finally, remove images in which the spatial constraints do not hold from the candidate-image set. Order the final candidate-image set by the average certainty of the symbols in each image of the set that were in the query image (highest average certainty first). Return this ordered set as the result of the pictorial query. Note

```
GetSimilarImages1(QI,csl,ssl)
n ← 0
foreach el ∈ QI
   r_n ← set of images containing C(el)
         (use index on class)
   n ← n + 1
if (csl = 1) ∨ (csl = 2) then
   RI ← ⋂_{i=0}^{n-1} r_i
elseif (csl = 3) ∨ (csl = 4)
   RI ← ⋃_{i=0}^{n-1} r_i
if (csl = 1) ∨ (csl = 3) then
   RI ← RI − {I containing symbols not in QI}
         (use index on image_id)
RI ← RI − {I s.t. spatial constraints
     do not hold} (call checkSsl)
return RI ordered by average certainties
```

Figure 5: Algorithm *GetSimilarImages1*.

```
GetSimilarImages2(QI,csl,ssl)
QIS ← QI sorted by number of instances of
   its symbols in database (fewest first)
foreach elq ∈ QIS
   RI ← set of images containing C(elq)
        (use index on class)
   if (csl = 1 ∨ csl = 3) then
      RI ← RI − {I containing symbols
           not in QI} (use index on image_id)
   if (csl = 1) ∨ (csl = 2) then
      RI ← RI − {I that do not
           contain all symbols in QI}
   RI ← RI − {I s.t. spatial constraints
        do not hold} (call checkSsl)
   if (csl = 1) ∨ (csl = 2) then
      /* result must have all symbols,
        no need to look for others */
      break to top level
return RI ordered by average certainties
```

Figure 6: Algorithm *GetSimilarImages2*.

that if $ssl = 1$ (i.e., the matching symbols of the database and query image must be in the exact same locations), then the spatial constraint can be checked simultaneously with the contextual constraint. This is achieved by initially retrieving the image id's of all database images that contain each symbol of the query image in the same location (using either the index on class or location). The remainder of the algorithm is the same except that there is no need to call *checkSsl*.

Figure 6 summarizes *GetSimilarImages2*. The idea is to first narrow down the number of candidate images according to the contextual information by inserting all database images that contain one particular symbol of the query image into the initial candidate set (start with symbol with fewest instances in the database). Next, remove all

```
GetSimilarImages4(QI,csl,ssl)
el_1 ← el ∈ QI with fewest instances in database
D ← dist(loc(el_1),loc(el_2)) s.t. it is maximal
RI ← all images containing C(el_1)
     (use index on class)
RI ← RI − {I s.t.  all elements of I
     within D of loc(el_1) do not
     include all symbols of QI}
       (use index on location)
RI ← RI − {I s.t. spatial constraints
     do not hold} (call checkSsl)
return RI ordered by average certainties
```

Figure 7: Algorithm *GetSimilarImages4*. It assumes that csl=2.

images that do not conform to the contextual and spatial constraints from the candidate-image set. If $csl = 3$ or 4, then repeat this process for all other symbols in the query image. This is needed since a database image may not contain the particular symbol that was chosen for the initial search, yet still contain another symbol from the query image, and thus be a valid result.

GetSimilarImages3, is a variant of *GetSimilarImage2* which is applicable when csl = 1 or 2 (i.e., all query symbols must appear in result images), and when the spatial constraints involve distance (ssl = 2 or 4). The idea is to first narrow down the number of candidate images according to both the contextual and the spatial information by identifying the query image symbol, s_k, with the fewest instances in the database and the query symbol, s_l, that is closet to s_k. The image id's of all database images that contain symbols s_l and s_k within $dist(s_k, s_l)$ are retrieved and composed into the initial candidate-image set. At that point, images in which the contextual and spatial constraints do not hold are removed from the candidate-image set.

Figure 7 summarizes *GetSimilarImages4*, which is applicable when $csl = 2$, and ssl = 2 or 4. The idea is that since we do not need to verify that result images have only symbols that are in the query image (since $csl = 2$), we can avoid retrieving all elements of each candidate image. Instead, only those elements of candidate images that conform to the distance specification are retrieved. As in *GetSimilarImages3*, we first identify the query image symbol, s_k, with the fewest instances in the database. Compute the distance, D, to the symbol in QI that is furthest from s_k. Compose the image id's of all database images that contain s_k into an initial candidate-image set. For each image in this set, use the spatial index to find all symbols that are within D of s_k in it. Remove all images for which this set of symbols does not include an instance of each symbol in QI. A call to $checkSSl$ is still required to check if the distance constraints among the symbols found within D of s_k hold

```
checkSsl(DI, QI, ssl)
if ssl = 5 ∨ |DI| = 1 then
  return TRUE /* nothing to check */
/* compute distances and relative location
  between QI symbols*/
foreach qel1 ∈ QI
  foreach qel2 ∈ QI − {qel1}
    if (ssl = 2) ∨ (ssl = 4) then
      dists[C(qel1), C(qel2)] ←
          getDist(loc(qel1), loc(qel2))
    if (ssl = 2) ∨ (ssl = 3) then
      relDirs[C(qel1), C(qel2)] ←
          getReldir(loc(qel1), loc(qel2))
/* check that these hold in input image */
foreach del1 ∈ DI
  foreach del2 ∈ DI − {del1}
    if (ssl = 2) ∨ (ssl = 4) then
      if getDists(loc(del1), loc(del2)) >
          dists[C(del1), C(del2)] then
        return FALSE
    if (ssl = 2) ∨ (ssl = 3) then
      if getReldirs(loc(del1), loc(del2)) ≠
          relDirs[C(del1), C(del2)] then
        return FALSE
return TRUE /* everything is OK */
```

Figure 8: Algorithm *checkSsl* to check whether the spatial constraints dictated by a query image QI and spatial similarity level ssl hold in a logical image DI.

and to check the relative direction constraints (if ssl = 2).

GetSimilarImages5 is a variant of *GetSimilarImages4*, which is applicable when $csl = 2$ and ssl = 2 or 4. The idea is to take care of the spatial and the contextual constraints simultaneously, and avoid having to call a routine to check the spatial constraints separately. It performs a process similar to that of algorithm $GetSimilarImages4$ for each symbol s in the query image resulting in a set of candidate images that conform to the contextual constraints and to the distance constraints with respect to s. The intersection of these sets is the set of images in which all contextual and distance constraints hold. If ssl = 2, then the relative directions still need to be verified.

We now describe *checkSsl*. It checks whether the spatial constraints dictated by a query image QI and spatial similarity level ssl hold in a logical image DI. Figure 8 summarizes this algorithm. Assume that ssl = 2–5. The case of $ssl = 1$ can be handled in *GetSimilarImages1* and *GetSimilarImages2* directly with no need to call *checkSsl*, as described above. We also assume that the logical image DI that is passed to it, contains only elements that correspond to symbols in the Query image QI. Function *GetSimilarImages* constructs this logical image when it matches the symbols of DI and QI. The algorithm first computes the distance and/or the relative directions between the symbols of the query image QI. It then computes these for the database image DI and checks whether the needed constraints between the symbols of QI that correspond to those of DI hold[2].

Notice that the assumption that there is only one instance of each classification in the query image is needed to assure that there is exactly one pair of symbols in the query image corresponding to a pair of symbols in the database image. In addition, if we allow more than one instance of each class in the database image, then *checkSsl* as presented here is incorrect since it only checks the spatial constraints between pairs of symbols. To correct this, *checkSsl* must check every possible combination of size $|QI|$. Pictorial queries involving more than one query component are executed by performing a separate pictorial query for each component and then computing the intersection of the results for components joined by an AND operator, and the union of the results for those joined by an OR operator.

4.2. Comparison of Algorithms

Below we refer to the five algorithms presented in the previous section as GSI_1–GSI_5. The term "search by class name" refers to the process of retrieving all tuples that correspond to a given `class` using the index on class name. The term "search by image id" refers to the process of retrieving all tuples (symbols) that correspond to a given `image_id` using the index on image id's.

GSI_1 and GSI_2 are the most general. They can handle any contextual and spatial similarity levels. The difference between them is that in GSI_1, $|QI|$ "searches by class name" are always performed, while in GSI_2, if $csl = 1$ or 2, then only one search by class name is performed. However, in GSI_2 when $csl = 3$ or 4, there may be more "search by image id" operations. Thus, GSI_2 should be used if csl = 1 or 2, and GSI_1 should be used if csl = 3 or 4.

GSI_3 is only applicable if there are distance constraints (ssl = 2 or 4) and if $csl = 1$ or 2. Since GSI_2 is better than GSI_1 in this case, we compare GSI_3 with GSI_2. The difference between the two is that GSI_3 constructs a smaller initial candidate image set inserting only those images that contain the query-image symbol with fewest occurrences in the database and the query-image symbol that is closest to it, within the required distance. Therefore, GSI_3 will outperform GSI_2 if the cost of the spatial search is less than the cost of the additional "search by image id" operations resulting from a larger candidate set. In other words, if the spatial selectivity of the range query is high (few images will result from it), then GSI_3 should be used.

In GSI_4 there are no "search by image id" operations; however, the range for the spatial search is larger than it was in GSI_3 (since it is now the maximal distance, D, rather

[2] The distances and relative directions between query-image symbols actually only need to be computed once.

than the minimal distance), and thus the cost of each spatial search is higher. An additional difference is that in GSI_3 all symbols of each candidate image are retrieved in order to check the contextual constraints, whereas in GSI_4 only those symbols within D are retrieved. Therefore, GSI_4 will outperform GSI_3 if D is relatively small and there are much fewer symbols in its range than in the entire image.

The advantage of GSI_5 is that there is no need to call *checkSsl*, at the cost of more spatial range queries. In GSI_4, $N_{c_{min}}$ spatial range queries are required , where $N_{c_{min}}$ is the number of images that contain the query-symbol with fewest occurrences in the database. On the other hand in GSI_5, $N_{c_{tot}}$ spatial range queries are required, where $N_{c_{tot}}$ is the number of occurrences of all query symbols in the database. The cost of each call to *checkSsl* is $O(|DI|^2)$ where $|DI|$ is the number of symbols in the database image that also appear in the query image. Since $csl = 2$, $|DI| = |QI|$. *checkSsl* is called $|RI|$ times, where $|RI|$ is the number of images in which the contextual constraints hold. Thus, the total time saved by GSI_5 over GSI_4 is $|RI| \times |QI|^2$. The additional cost is $N_{c_{tot}} - N_{c_{min}}$ spatial range queries. The cost of each such range query depends on the size of the range in the spatial query. It is higher for larger ranges. Thus, if the contextual selectivity of the symbol with fewest occurrences is high (few images contain it), then GSI_4 should be used. In some cases, where the contextual selectivity of all query symbols is low, but the spatial selectivity is high, GSI_5 will outperform GSI_4.

5. Implementation and Experimentation

The methods described in this paper were tested on a symbolic-image database that contains the red sign layer of the GT_3 map of Finland, which is one of a series of 19 GT maps that cover the whole area of Finland. The red sign layer contains geographic symbols that mostly denote tourist sites. The map was scanned at 240dpi. The layer was split into 425 tiles of size 512×512. Each one of these tiles that contained at least one symbol was considered to be an image. These images were stored in SYMIDB.

We are currently in the process of comparing the execution time of the five image similarity algorithms. The first step in this study is to build a corpus of queries with varying spatial and contextual selectivities. At present, we have experimented with a limited number of queries. Our initial results verify that the behavior predicted by the analytical comparison holds.

6. Concluding Remarks

Our algorithms showed how to handle images in which we could fully classify the symbols that appear in them. SYMIDB can also handle images where this requirement does not hold by storing feature vectors that describe the symbols in the database rather than the classifications. An index is constructed on these feature vector that enables efficient nearest neighbor searches in feature space. Our algorithms can be adapted easily to handle this case by using the index on the feature vectors rather than the index on class for contextual search.

Our examples and experiments were from the map domain. However, images from many other interesting application domains also fall into the category of symbolic images. One possible complication that could arise in other domains is that the spatial extent of symbols may be of importance. To accommodate this, we must refine the definition of spatial similarity levels so users can specify whether the extent of objects should be considered when comparing images. The algorithms, however, will not need much change as long as we use a standard spatial data structure to index the locational information. In contrast, it is considerably harder to deal with spatial extent in methods based on 2-D strings [1].

7. Acknowledgements

We are grateful to Karttakeskus, Map Center, Helsinki, Finland for providing us the map data.

References

[1] S. K. Chang, Q. Y. Shi, and C. Y. Yan. Iconic indexing by 2-D strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(3):413–428, May 1987.

[2] V. Gudivada and V. Raghavan. Design and evaluation of algorithms for image retrieval by spatial similarity. *ACM Trans. on Inf. Systems*, 13(2):115–144, Apr. 1995.

[3] R. C. Nelson and H. Samet. A consistent hierarchical representation for vector data. *Computer Graphics*, 20(4):197–206, August 1986.

[4] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker. The QBIC project: Querying images by content using color, texture, and shape. In *Proc of the SPIE, Storage and Retrieval of Image and Video Databases*, vol. 1908, pp. 173–187, San Jose, CA, Feb. 1993.

[5] D. Papadias and T. K. Sellis. A pictorial query-by-example language. *Journal of Visual Languages and Computing*, 6(1):53–72, Mar. 1995.

[6] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Content-based manipulation of image databases. In *Proc. of the SPIE, Storage and Retrieval of Image and Video Databases II*, vol. 2185, pp 34–47, San Jose, CA, Feb. 1994.

[7] A. Soffer and H. Samet. Retrieval by content in symbolic-image databases. In *Proc. of the SPIE, Storage and Retrieval of Still Image and Video Databases IV*, vol. 2670, San Jose, CA, Feb. 1996.

[8] M. Swain. Interactive indexing into image databases. In *Proc of the SPIE, Storage and Retrieval of Image and Video Databases*, vol. 1908, pp. 95–103, San Jose, CA, Feb. 1993.

[9] M. Ubell. The montage extensible dataBlade architecture. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, page 482, Minneapolis, MN, June 1994.