Using Spatial Sorting and Ranking in Model-Based Object Recognition

Gísli R. Hjaltason Manjit Ray Hanan Samet
Isaac Weiss
Computer Science Department
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742
{grh,manjit,hjs}@cs.umd.edu, weiss@cfar.umd.edu

Abstract

A new method has been developed recently for overcoming some of the major obstacles of object recognition. In this method, both 2D images and 3D models from a data-base are represented by view-point invariant descriptors in the same invariant space. Since there is a large number of possible matches between the image and model invariants, there is a crucial need to perform this match very efficiently. In this paper we answer this need by using an adaptation of spatial sorting based on the octree representation of the model points and image lines in three-dimensional space. In particular, the points and lines are sorted with respect to the space that they occupy using techniques similar to hashing. Once the points and lines have been sorted, we evaluate the usage of two alternatives matching methods. The first is based on finding the n nearest model points to each line, while the second is based on finding the m nearest line-point pairs. Both of the methods are incremental in the sense that n and m can be increase without having to restart the matching process. We show the advantage of these alternatives over the traditional "brute-force" approach and we compare the merits of the two methods. We demonstrate that the performance gain of our incremental algorithms increases substantially with the number of matches to be performed. This makes it possible for the recognition task to be performed in a reasonable time for a large number of models.

Keywords: invariance, object recognition, hierarchical methods.
Track: Pattern Recognition and Analysis

1 Introduction

In a typical recognition task, one has an image of an unknown object, and the objective is to match it with a known set of objects, given in an appropriate representation such as models or reference images. Since the viewpoint from which the image was taken and other camera parameters are unknown, the matching involves searching in a prohibitively large search space.

To avoid the search for the correct viewpoint, we can match viewpoint invariant descriptors of the models and images rather than the original representations. Such invariants exist for projective transformations, including viewpoint changes, between spaces of the same dimensionality, e.g. a projection of a planar object onto an image. However, when we project a 3D object onto a 2D image, the depth information is lost. This loss creates difficulties in two ways:

1. There are no full invariants, only invariant constraints. That means that the search space cannot be eliminated completely as in the 2D case, although it is far smaller than the non-invariant description.

2. General objects cannot be identified uniquely. In principle, many different objects, differing only in their depth coordinates, can yield the same 2D image.

In this paper we deal with the two issues above. In the first issue, since a search is unavoidable, our goal is to reduce the complexity of the search as much as possible. Two key steps are involved here: a) representation of both the models and the images as entities in an invariant space b) using spatial sorting and ranking algorithms to perform efficient matching in this space.

The second issue is dealt with by the use of modeling assumptions. Such assumptions should make up for the missing depth information. One such assumption is the use of a predefined set of known models rather than trying to recognize a general unconstrained object. Such modeling is already partly implied in our definition of object recognition. However, we have to be careful that our set of models do not contain many models that can project the same image.

Our method is described briefly as follows. We represent each 3D model as a set of points in a 3D invariant space. An image is represented as a set of invariant *lines* in the same invariant space. The fact that we have lines rather than points reflects the fact that the image is missing the depth information. The goal is now to find the model point set which is closest to the set of lines representing the image. This involves ranking distances between lines and points in 3D. The ranking is facilitated by sorting the points and lines with respect to the space that they occupy. The sorting makes use of a hiearchical spatial data structure known as an octree which is a three-dimensional variant of the quadtree.

In contrast to usual methods, we make use of an incremental approach termed *distance ranking* [2]. Given an object, we find the $k+1^{st}$ nearest neighbor without having to recompute the k nearest neighbors. When we deal with pairs of objects such as point-line pairs, we can find a global ranking of the distances over all pairs by what is termed a join algorithm (e.g., [1]).

2 Object Recognition and Modeling

Most work in invariants have concerned transformations between spaces of equal dimensionality [9, 7]. For a single view, invariants were found for planar curves, while for 3D objects, multiple views with known correspondence were involved.

Since it has been shown that there are no invariant functions of a single projection from 3D to 2D [5], modeling assumptions are required to recover 3D shape. Such assumptions can take the form of generalized cylinders [4, 10], surfaces of revolution or orthogonal vertices [6]. Another approach is to characterize an object by local properties like tangencies and inflection points [8]. Our method does not make any such specific modeling assumption, but uses a set of given models.

2.1 The Method

Given a set of representations of 3D models and a set of image feature configurations, it is required to determine which models appear in the image. The mechanism ought to ensure that the set of hypothesized models is generated fairly quickly.

An outline of the method follows. A 3D invariant space based on triplets of invariants is defined and such triplets extracted from the models, so that each model is represented by a collection of points in this invariant space. Given an image of a model, we derive a set of lines in this space. If a sufficient number of these lines intersect the points corresponding to a model, then we can hypothesize that the image is indeed of that model. These intersections can be determined fairly quickly thus circumventing an expensive search for correspondences. Once a model has been identified its pose can be determined and it can be projected to the image in order to verify the match.

Thus, this method extends the use of model-based invariant functions to the task of recognizing general 3D models from a single perspective view. The recognition task is reduced to a search for intersections between simple subspaces (lines,points) in an invariant space.

2.2 3D Invariants

A 3D affine transformation A is uniquely determined by the mapping of 4 non-coplanar points. Hence, we can define a canonical frame by mapping any four non-coplanar points $\{\mathbf{X}_i\}_{i=1}^4$ to the

points $\{\mathbf{I}_O = (0\ 0\ 0\ 1^T), \mathbf{I}_x = (1\ 0\ 0\ 1)^T, \mathbf{I}_y = (0\ 1\ 0\ 1)^T, \mathbf{I}_z = (0\ 0\ 1\ 1)^T\}$. The coordinates of any point $\mathbf{X}$ in this canonical frame will be 3D affine invariants. This invariant representation $\mathbf{I}$ is given by $\mathbf{I} = A\mathbf{X}$ where $A = (\mathbf{I}_O, \mathbf{I}_x, \mathbf{I}_y, \mathbf{I}_z)(\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4)^{-1}$. Since $\mathbf{I}$ will be of the form $(a\ b\ c\ 1)^T$ we use the triplet (a, b, c) to define the 3D invariant space.

2.3 Perspective Projection

A 3D point $\mathbf{X}$ is related to its image $\mathbf{x}$ as $\mathbf{x} = T\mathbf{X}$ where T is a 3×4 projection matrix incorporating the effects of the unknown camera geometry and the viewpoint transformation. Since $\mathbf{X}$ is linearly related to its invariant representation $\mathbf{I}$, we can write the equation as $\mathbf{x} = \tilde{T}\mathbf{I}$ where $\tilde{T} = TA^{-1}$ is of the same dimensions as T.

Let $\tilde{T} = \{t_{ij}\}_{i=1,\ldots,3}^{j=1,\ldots,4}$. In order to eliminate these unknown parameters, we require a set of at least six image points $\{\mathbf{x}_i\}_{i=1}^{6}$. Let the invariant representations of the corresponding model points be $\{\mathbf{I}_i\}_{i=1}^{6}$. Assume that $\{\mathbf{I}_i\}_{i=1}^{4}$ correspond to the points mapped to the canonical frame and $\{\mathbf{I}_i\}_{i=5}^{6}$ are $\{(a\ b\ c\ 1)^T, (\alpha\ \beta\ \gamma\ 1)^T\}$.

If $\mathbf{I} = (p\ q\ r\ 1)^T$ is an invariant representation of a model point and $\mathbf{x} = (x, y)$ its image point, then

$$x = \frac{t_{11}p + t_{12}q + t_{13}r + t_{14}}{t_{31}p + t_{32}q + t_{33}r + t_{34}}, \quad \text{and} \quad y = \frac{t_{21}p + t_{22}q + t_{23}r + t_{24}}{t_{31}p + t_{32}q + t_{33}r + t_{34}} \tag{1}$$

Since we have 12 equations from the 6 image points $\{\mathbf{x}_i\}_{i=1}^{6}$, we can eliminate the parameters of $\tilde{T}$ to obtain an invariant relation involving the unknowns $\{a, b, c, \alpha, \beta, \gamma\}$. However, this is a fourth-order surface and of limited practicality.

2.3.1 A simplification

A significant simplification can be achieved by a simple modeling assumption. We can then reduce the problem to finding the intersection between a line and a point in invariant space, using only five rather than six image points. We make the assumption that *vanishing points* in the image are an indication of parallelism in space. The *vanishing points* in the directions defined by the four image points forming the basis, can be assumed to correspond to model points with invariant representations $\{(1\ 0\ 0\ 0)^T, (0\ 1\ 0\ 0)^T, (0\ 0\ 1\ 0)^T\}$. Using these together with the basis points, we can directly estimate $\tilde{T}$ by solving the resultant linear system with $t_{34} = 1$. Once, $\tilde{T}$ is determined, the invariant representation of an additional point $\mathbf{I} = (a\ b\ c\ 1)^T$ is related to its image $\mathbf{x} = (x, y)$ by

$$\begin{aligned}(t_{11} - xt_{31})a + (t_{12} - xt_{32})b + (t_{13} - xt_{33})c &= x - t_{14} \\ (t_{21} - yt_{31})a + (t_{22} - yt_{32})b + (t_{23} - yt_{33})c &= y - t_{24}\end{aligned} \tag{2}$$

which, being the equations of two planes, define a line passing through the point (a, b, c) in invariant space. The invariant space is illustrated in Figure 3.

2.4 The Recognition Process

The recognition algorithm proceeds as follows:

1. *Extraction of 3D invariants*: Invariant model points are generated using a small number of basis sets to define the canonical frame. Different basis sets are utilized in order to ensure that all the points of at least one set are visible in an image.

2. *Extraction of image features*: Corners need to be extracted in order to define invariant lines. In this implementation, intersections of image lines are used to localize corners.

3. *Purging of feature points and estimation of vanishing points*: Since quintuples of corners are required to define invariant lines, the number of such lines can be very large. To reduce this, only the lines lying on principal directions are retained. Such closely parallel image lines can be assumed to correspond to parallel lines in space, so that their intersections can be used to estimate the vanishing point in a particular direction.

4. *Determination of line-point intersections in invariant space*: Points lying close to an invariant line derived from the image are extracted using spatial sorting and ranking (see Section 3). Once these points are obtained, a set of possible models together with their respective poses can be hypothesized.

5. *Verification*: In order to select the correct model from among the hypothesized ones, it is necessary to project each model onto the image and evaluate the match. Such a projection is achieved by estimating the projection matrix from the hypothesized matches between the image and model points.

3 Ranking and Join

3.1 Octrees

We use two different hierarchical sorting algorithms for the different spatial features — that is, the points that make up the models and the infinite lines that make up the image. The sorting methods are based on octrees (a three dimensional form of quadtrees). The idea is that we recursively decompose the underlying three-dimensionaly space into eight cube blocks until each block contains

one or a very small number of points. This is the basis of the PR octree [3]. This technique may result in much decomposition when two points are very close to each other. This problem is overcome by increasing the number of points for which the decomposition halts. Figure 1a is an example of the block decomposition resulting from a PR quadtree for points in two-dimensional space.

The octree sorting technique can also be adapted to non-point data such as collections of line segments. However, we must be careful when the line segments are connected as a decomposition rule that only halts when each block contains one line segment will never halt. To overcome this, we use a different splitting strategy. Whenever the insertion of a line segment causes the block to have more than s line segments, then the block is decomposed once and only once into eight cube blocks. If the subsequent insertion of other line segments causes block b to contain more than s line segments, then the block is split again but just once. PMR octrees can also be used for collections of points as well as other spatial objects such as faces of three-dimensional polyhedra, etc. Figure 1b is an example of the block decomposition resulting from a PMR quadtree for line segments in a two-dimensional space inserted in alphabetical order. In a quadtree/octree, non-point objects may be stored in more than one leaf block. For example, segment A in Figure 1b is associated with three leaf nodes.

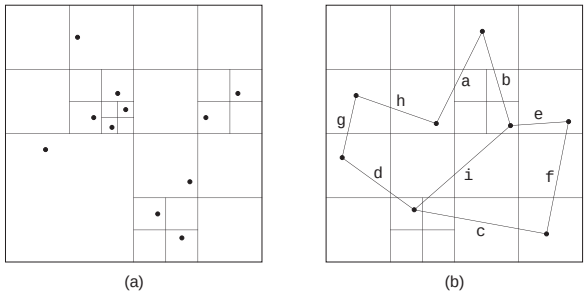

Figure 1: (a) A PR quadtree for points, and (b) a PMR quadtree for line segments with a splitting threshold of 2, where the line segments have been inserted in alphabetical order.

3.2 Incremental Nearest Neighbor Algorithm

The incremental nearest neighbor algorithm is a method for obtaining objects from a set of spatial objects S in the order of distance from a given query object q (termed *ranking*). The set of objects must be organized with some sort of a hierarchical spatial data structure, octrees in our case. The algorithm is incremental in the sense that the nearest neighbors are reported one-by-one. That is, if we have found k nearest neighbors and we need an additional one, we do not need to recalculate the current k neighbors as other methods do. Thus, it is especially useful when the exact number of neighbors needed is not known in advance.

The algorithm uses a priority queue to organize nodes from the spatial data structure as well as

objects. The key used to order the elements on the queue is their distance from the query object. Special treatment is done when two elements are at equal distance from the query object.

Initially, the node spanning the whole index space is the sole element in the priority queue. In the main loop of the algorithm, the element at the head of the queue (i.e., the closest element not yet examined) is retrieved. If the element is an object from the set S it can be reported. If the element is a node, then its contents (i.e., child nodes or objects, depending on whether the node is an internal node or a leaf node) is inserted into the queue. One way of visualizing the workings of the algorithm when the query object q is a point is as follows: The algorithm first locates the leaf node in the spatial data structure that contains q. Then a search circle is expanded around the q. When the circle hits a new node or an object, that node or object will have reached the front of the priority queue.

3.3 Incremental Distance Join

When the ranking is with respect to a set of reference objects, the process is more complex than finding the nearest object to one reference object. The distance join operation is an extension of the ranking operation that operates on two sets of objects S_A and S_B. Informally, it is an ordering on all pairs of objects from S_A and S_B, based on distance. If one of the sets contains only one element, the distance join operation is equivalent to the ranking operation.

The incremental distance join algorithm is based on the same principles as the incremental nearest neighbor algorithm. Thus, the two sets must be organized with a hierarchical spatial data structure. The incremental join algorithm may be thought of as applying the incremental nearest neighbor algorithm simultaneously to the two spatial data structures.

The input to the incremental join algorithm is two spatial indexes, A and B. The algorithm maintains a set of pairs P, with one item from each of A and B, each item being either a node or an object. A priority queue is used to organize the set of pairs P in order of distance. Initially, P contains just one pair corresponding to the root nodes of A and B. At each step in the algorithm, the element at the head of the priority queue is retrieved, i.e., the element with the smallest distance key. If the element stores a pair of data objects, then the pair is reported as the next closest pair. If one of the items in the dequeued element is a node, then the algorithm pairs up the entries of the node (objects for leaf nodes, child nodes for non-leaf nodes) with the other item. If both items the pair are nodes, we have a choice which one to process. The best way of making this choice depends on the spatial data structure and the application.

Point Sets	#points	Line Sets (Small)	#lines	Line Sets (Large)	#lines
P_1	8,896	Mod. 1 Im. 1 (M_1L_{11})	1,120	Mod. 1 Im. 1 (M_1L_{12})	2,144
		Mod. 1 Im. 2 (M_1L_{21})	420	Mod. 1 Im. 2 (M_1L_{22})	2,624
P_2	26,687	Mod. 2 Im. 1 (M_2L_{11})	814	Mod. 2 Im. 1 (M_2L_{12})	12,191
		Mod. 2 Im. 2 (M_2L_{21})	902	Mod. 2 Im. 2 (M_2L_{22})	20,760

Table 1: Dimensions of invariant point and line sets

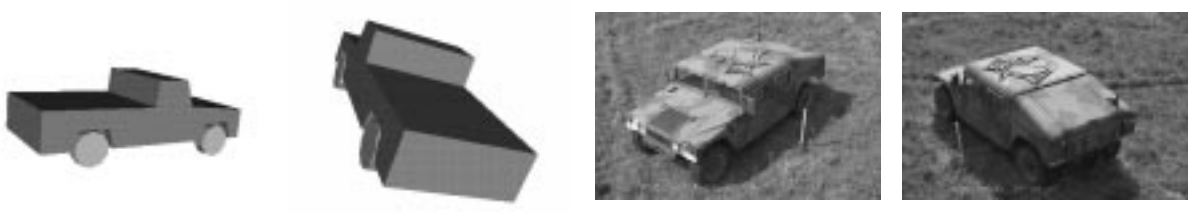

Figure 2: Images of Model 1 and Model 2.

4 Experiments

Experiments were performed on two images each of a truck and of a HMV vehicle (Fig. 2). From each of the two corresponnding models, two invariant point sets, P_1 and P_2, are extracted, the smaller P_1 under a more restrictive selection of feature sets. From each image, two invariant line sets are defined, the smaller from true corner features detected in the image and the larger from both true and false corners (Fig. 5). The dimensions of these sets are shown in Table 1. Intersections between these lines and the invariant points derived from the models are demonstrated in Figure 4.

Points lying close to each line in the invariant space are obtained by a variety of hierachical methods described subsequently. The points are classified into subsets according to the model and the basis from which they were obtained. The lines are classifiied according to the basis used in the image. The point subsets lying closest to a line subset are hypothesized as matches, and verified by projection on the image. Since only two models were used, the recognition was simple for all four images, but more models need to be incorporated into this framework.

4.1 Timing Results

All of our experiments were run on a Sun Ultra 1 Model 170E machine, rated at 6.17 SPECint95 and 11.80 SPECfp95, with 64MB in main memory. The software was compiled with a GNU C++ compiler set for maximum optimization (–O3). The results are shown in Table 2. All times are in seconds and the speedup is shown with respect to the brute force approach.

Three search methods are used in the results shown in Table 2: 1) The incremental join algorithm

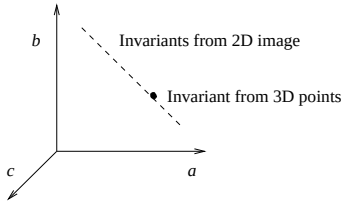

Figure 3: Invariant space.

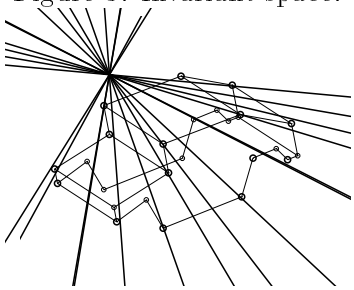

Figure 4: Intersections in invariant space between lines and points of Model 3 derived from matching bases.

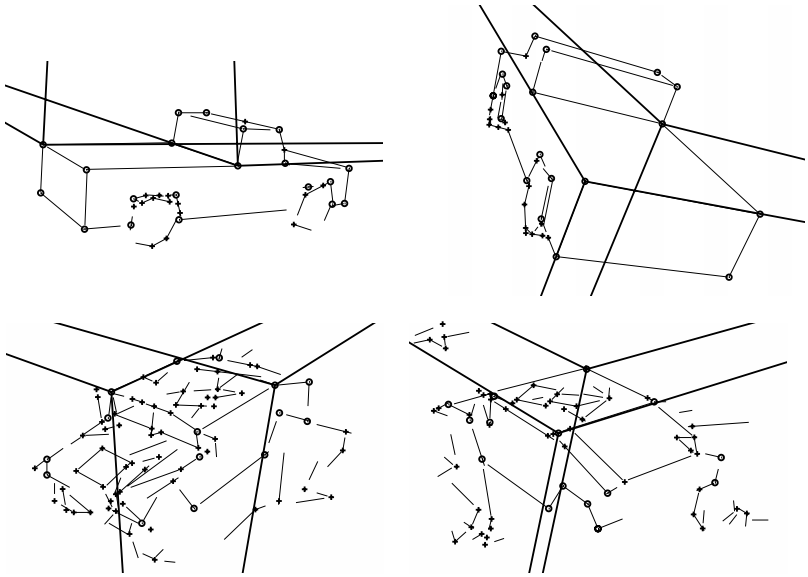

Figure 5: True corners (circles), false corners (crosses) and vanishing points(thick lines) in the images.

Test	*A*		*B*		*C*		*D*
	time	*speedup*	*time*	*speedup*	*time*	*speedup*	*time*
$P_1 : M_1L_{11}$	8.1	1.8	7.6	1.9	3.4	4.4	14.8
$P_1 : M_1L_{21}$	3.2	1.7	3.7	1.5	1.2	4.6	5.5
$P_1 : M_2L_{11}$	6.1	1.8	6.2	1.7	2.2	4.9	10.7
$P_1 : M_2L_{21}$	6.2	1.9	6.4	1.9	2.4	5.0	11.9
$P_1 : M_1L_{12}$	6.7	4.2	6.0	4.7	3.0	9.5	28.4
$P_1 : M_1L_{22}$	10.3	3.4	8.9	3.9	3.9	8.9	34.9
$P_1 : M_2L_{12}$	42.8	3.8	36.9	4.4	17.3	9.4	162.5
$P_1 : M_2L_{22}$	73.7	3.8	60.8	4.6	30.0	9.3	277.7
$P_2 : M_1L_{11}$	14.0	3.2	14.8	3.0	5.3	8.5	44.8
$P_2 : M_1L_{21}$	8.8	1.9	8.3	2.0	1.7	9.8	16.7
$P_2 : M_2L_{11}$	12.6	2.6	12.8	2.5	3.5	9.3	32.6
$P_2 : M_2L_{21}$	12.4	2.9	13.3	2.7	3.7	9.8	36.2
$P_2 : M_1L_{12}$	13.1	6.6	14.3	6.0	5.8	14.8	86.0
$P_2 : M_1L_{22}$	18.2	5.8	18.9	5.6	6.3	16.8	105.6
$P_2 : M_2L_{12}$	81.1	6.1	74.8	6.6	29.3	16.8	492.5
$P_2 : M_2L_{22}$	130.5	6.1	117.7	6.7	52.2	15.1	789.9

A: Join1 (max distance = 0.2) *C*: Nearest Neighbor (#neighbors = 50)
B: Join2 (max distance = 0.2) *D*: Brute Force

Table 2: Timing Results

(Join1 and Join2) 2) the incremental nearest neighbor algorithm, and 3) brute force. The times for Join1 and Join2 include the cost of building a line octree, which are built using two different methods. The first is based on the PMR quadtree rules, with a splitting threshold ranging from 128 to 512, while the second is to build the line octree based on the same spatial decomposition as for the point octree. The second method leads to faster octree build times, but sometimes slower search times. The nearest neighbor experiment results are for computing a fixed number of neighbors for each line, using the point octree.

The nearest neighbor approach appears to be superior to the join approach. However, we have not yet explored the full advantages of the join method.

The advantage of the nearest neighbor approach is that it enables the addition of lines and finding their nearest n points. In this case, we have the ability to converge on a solution in an incremental manner. An alternative approach is to use the incremental join algorithm and find the closest m pairs of points and lines. This approach has the advantage of enabling a quick convergence on a solution in the sense that if the points that are the closest to the lines are from the same model, then we know that we have hit on a good solution. The advantage of the join method over the nearest neighbor method is that it operates simultaneously over all the lines in the line set. With the nearest method, once a nearest neighbor query has been terminated for line l_1 and a new one started for the line l_2, we cannot request more neighbors for l_1 without computing all the nearest neighbors from scratch, even the ones that have already been determined.

Another advantage of our incremental approach is that it is not so sensitive to the model size. Addition of more models is not so expensive as the PR quadtree for the points is relatively fast to compute and only needs to be done once. Thus an extensive model library can be built up over time. In addition, the storage requirements of the PR quadtree are proportional to the number of points in each model.

5 Concluding Remarks

The main conclusion from this study is that the object recognition task is greatly speeded up by the use of hierarchical methods for ranking distances in an invariant space. The speadup factor increases as the number of image and model features increase. These hierarchical methods would not be possible without the invariant representation of both the models and the images.

An interesting by product of this study has been the evaluation of the ranking and join algorithms on collections of lines. Most previous studies that have dealt with lines have used collections of very short line segments that are found on maps. Thus in our PMR octree a line needs to be decomposed

into many more q-objects. Also, alternative spatial representation such as R-trees which would represent each spatial object (i.e, line) by its minimum bounding rectangle would not result in a sort of the objects as many of the bounding boxes would overlap thereby leading to little pruning of objects from consideration when performing the join operation.

References

[1] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems.* Benjamin Cummings, Redwood City, CA, 1989.

[2] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In M. J. Egenhofer and J. R. Herring, editors, *Advances in Spatial Databases — Fourth International Symposium, SSD'95*, pages 83–95, Portland, ME, August 1995. (also Springer Verlag Lecture Notes in Computer Science 951).

[3] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.

[4] T. O. Binford and T. S. Levitt. Model-based recognition of objects in complex scenes. In *Proceedings of the DARPA Image Understanding Workshop*, pages 89–100, 1996.

[5] J. B. Burns, R. S. Weiss, and E. M. Riseman. The non-existence of general case invariants. In J. L. Mundy and A. Zisserman, editors, *Geometric Invariance in Machine Vision*, pages 120–134. MIT Press, Cambridge, MA, 1992.

[6] J. E. Hopcroft, D. P. Huttenlocher, and P. C. Wayner. Affine invariants for model-based recognition. In J. L. Mundy and A. Zisserman, editors, *Geometric Invariance in Machine Vision*, pages 354–374. MIT Press, Cambridge, MA, 1992.

[7] E. Rivlin and I. Weiss. Local invariants for recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:226–238, 1995.

[8] B. Vijayakumar, D. J. Kriegman, and J.Ponce. Structure and motion of curved 3D objects from monocular silhouettes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 327–334, 1996.

[9] I. Weiss. Geometric invariants and object recognition. *International Journal of Computer Vision*, 10:207–231, 1993.

[10] M. Zerroug and R. Nevatia. Using invariance and quasi-invariance for the segmentation and recovery of curved objects. In J. L. Mundy, A. Zisserman, and D. Forsyth, editors, *Applications of Invariance in Computer Vision*, pages 317–340. Springer-Verlag, Berlin, 1994.