Pattern Recognition Letters 13 (1992) 263-278
North-Holland

April 1992

3D object recognition from 2D images using geometric hashing

D.M. Gavrila

Department of Computer Science, University of Maryland, College Park, MD 20742, USA

F.C.A. Groen

Faculty of Mathematics and Computer Science, University of Amsterdam, Amsterdam, Netherlands

Received 8 July 1991
Revised 16 December 1991

Abstract

Gavrila, D.M. and F.C.A. Groen, 3D object recognition from 2D images using geometric hashing, Pattern Recognition Letters 13 (1992) 263-278.

In this paper, a general technique for model-based recognition is discussed, called Geometric Hashing. Its purpose is to identify an object in the scene, together with its position and orientation. This technique is based on an intensive preprocessing stage, done off-line, where transformation invariant features of the models are indexed into a hash table. This makes the actual recognition particularly efficient. The algorithm stands out for its high inherent parallelism and its ability to deal with occluded scenes. This paper focuses on the use of Geometric Hashing for the case of 3D object recognition from 2D images. An efficient method to represent a 3D model by its 2D projections is proposed. Results are presented of experiments on random data and 3D objects. It has been found that distinguishing between different types of features in a model or scene results in a very efficient implementation of Geometric Hashing using a multidimensional hash table. The filtering ratio of this scheme turns out to be high enough to allow reliable recognition with the correct feature correspondence between model and scene. The algorithm performed succesfully in dealing with scenes with up to 50% of occlusion and performed at speeds in the order of one second on a SPARC station.

Keywords. Geometric hashing, 3D object recognition.

1. Introduction

Research in robot vision has increased significantly over the years. Much effort has been devoted to the topic of object recognition, since this is an important step towards the increased autonomy of robot systems [3, 5]. The approach that has proved useful in particular, is the model-based approach in which recognition involves matching the input image with a set of predefined models in a (CAD) database. In most cases, recognition of a model is not sufficient and should include the position and orientation. An effective vision system must further be able to deal with partial occlusion and noise and still achieve an acceptable recognition rate.

This paper discusses Geometric Hashing, a general technique for model-based recognition, introduced by Lamdan and Wolfson [6]. It represents a robust object recognition method where matching is done on *local* features of an object. A voting scheme is used to evaluate the number of correct matches. This makes this scheme particularly suited

for recognition in occluded scenes. The object modeling phase of Geometric Hashing consists of describing the object by a set of points, called interest points, together with their geometrical relation. The same interest points are extracted from the scene together with their geometrical relation. The matching consists of finding a compatible transformation, between a set of points representing a model and the set of points representing the scene. Finding this transformation determines the position and orientation of the model in the scene with a certain tolerance.

A key factor in Geometric Hashing is the division between the preprocessing stage and the actual recognition stage. The processing burden has been concentrated as much as possible to the preprocessing stage, which is done off-line. This makes the actual recognition stage as fast as possible, but results in heavy memory demands. One of the major advantages of Geometric Hashing further is its high inherent parallelism both in the preprocessing stage and in the recognition stage.

This paper is organized as follows. Section 2 will start with an example of the Geometric Hashing concept followed by a summary of the general scheme. Subsequently, the complexity and error analysis of the algorithm is discussed. An efficient method to represent a 3D model by its 2D (model) projections is proposed. Section 3 will discuss an implementation of these methods together with its memory requirements. In Section 4 some characteristics of Geometric Hashing will be examined on random data, in particular its ability to filter out random solutions. The next step is to apply the algorithm to the case of 3D objects in the scene. Finally, Section 5 presents the conclusions. It will turn out that by using different types of interest points to describe models and scenes, the current implementation is able to achieve reliable recognition at high speeds.

2. Geometric hashing

2.1. *Affine transformation in 2D*

Consider flat object recognition under affine transformation (rotation, translation, scale and shear). This case occurs when we assume orthogonal projection of the object on the viewing plane. An affine transformation T can be described by a 2×2 non-singular matrix A and a 2×1 translation vector b, mapping vector x to vector $Ax+b$. Since we have six degrees of freedom, the transformation is fully determined by three point-to-point correspondences. Given a set of points describing the model and a set of points describing the scene, we have to find the affine transformation which maps a subset of the scene points onto a subset of the model points.

To compare the two pointsets, it is needed to find some property of a pointset that remains invariant with respect to the transformation performed. Let p_0, p_1, and p_2 be three non-collinear points in 2D space and let (p_1-p_0, p_2-p_0) be a basis spanning the 2D space with p_0 as origin. See Figure 1.

Let the coordinates of a fourth point p with respect to the chosen basis be (a, b). The key observation of Geometric Hashing is that if the whole system undergoes an affine transformation T, the coordinates of the transformed point Tp with respect to the transformed basis will remain (a, b). This means that we have found a transformation invariant property, i.e., the coordinates of points with respect to a basis.

The process of recognition is now divided in two stages: **preprocessing** (object representation) and **actual recognition** (matching). In the **preprocessing** stage a basis triplet is chosen from the model points and the coordinates of all the other model points are computed with respect to this basis. The coordinates serve as indices to a hash table and for

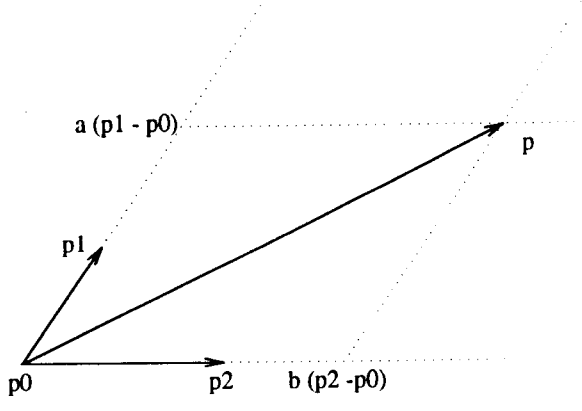

Figure 1. Basis for affine transformation.

each point a hash element containing (*model, basis triplet*) is inserted at the appropriate entry. In the **actual recognition** stage three points are chosen from the scene and again the coordinates of all the other scene points are computed with respect to this basis. For each point, a vote is given to all hash elements at the entry indexed by its coordinates. If a model with m points lies unoccluded in the scene, and the same basis triplet was chosen in the preprocessing stage as in the recognition phase, the combination (*model, basis triplet*) will get the maximum of $m-3$ votes. If the model was partially occluded there will be k missing points and the combination (*model, basis triplet*) will get as many as $m-k-3$ votes, and may be still detectable. If there are combinations that have collected more votes than a certain threshold the process of Geometric Hashing will deliver candidate solutions. The chosen scene basis triplet, together with a combination (*model, basis triplet*) gives three point-to-point matches for an object which determines the affine transformation. On the other hand, if no combinations receive more votes than a certain threshold we choose another basis triplet in the scene and repeat the voting process. If all basis triplets in the scene have been tried and no candidate solution has come up, the recognition process ends with no recognition. To ensure that we do not miss an object because one or more of his basis points used in the preprocessing phase is occluded in the scene, we repeat the preprocessing stage for *each* basis triplet of a model. During the recognition stage, it is now sufficient to choose *any* three scene points belonging to some object to ensure recognition.

The Geometric Hashing technique was illustrated for the 2D affine transformation, but the concept applies to a wide range of transformations in N-dimensional space. It can be used for the 2D case, when no depth information of an object is available, but it equally applies for the 3D case, when range or stereo data is available.

Of particular importance is also the similarity transform, which we will use later. In this case we have translation, rotation and scale in 2D. A two point basis (p_0,p_1) is enough to form a basis which spans the 2D space. The first basis vector is chosen between the two points and the second lies counterclockwise orthogonal to the first.

The advantage of choosing the middle of p_0 and p_1 as origin of the new coordinate frame (Figure 2b), instead of placing the origin at one of the basis points (Figure 2a) is that the hash table becomes more balanced after the preprocessing. To see this, observe that if the coordinates of a third point with respect to basis (p_0,p_1) is (x,y), it is $(-x,-y)$ with respect to (p_1,p_0). Therefore, if both orderings are used as a basis, the hash table will be symmetrical.

2.2. *The general scheme*

The general outline of the algorithm is as follows (see Figure 3, from [6]).

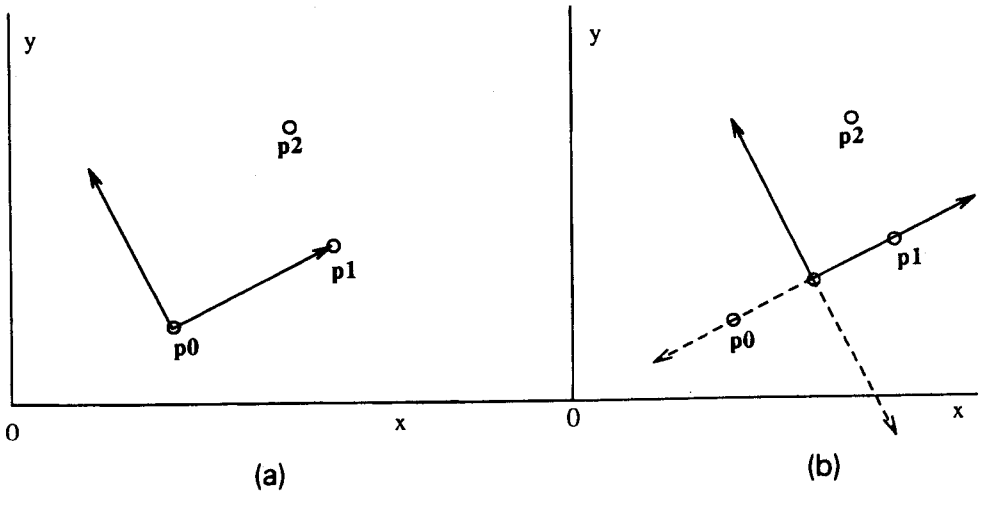

Figure 2. Basis for similarity transformation.

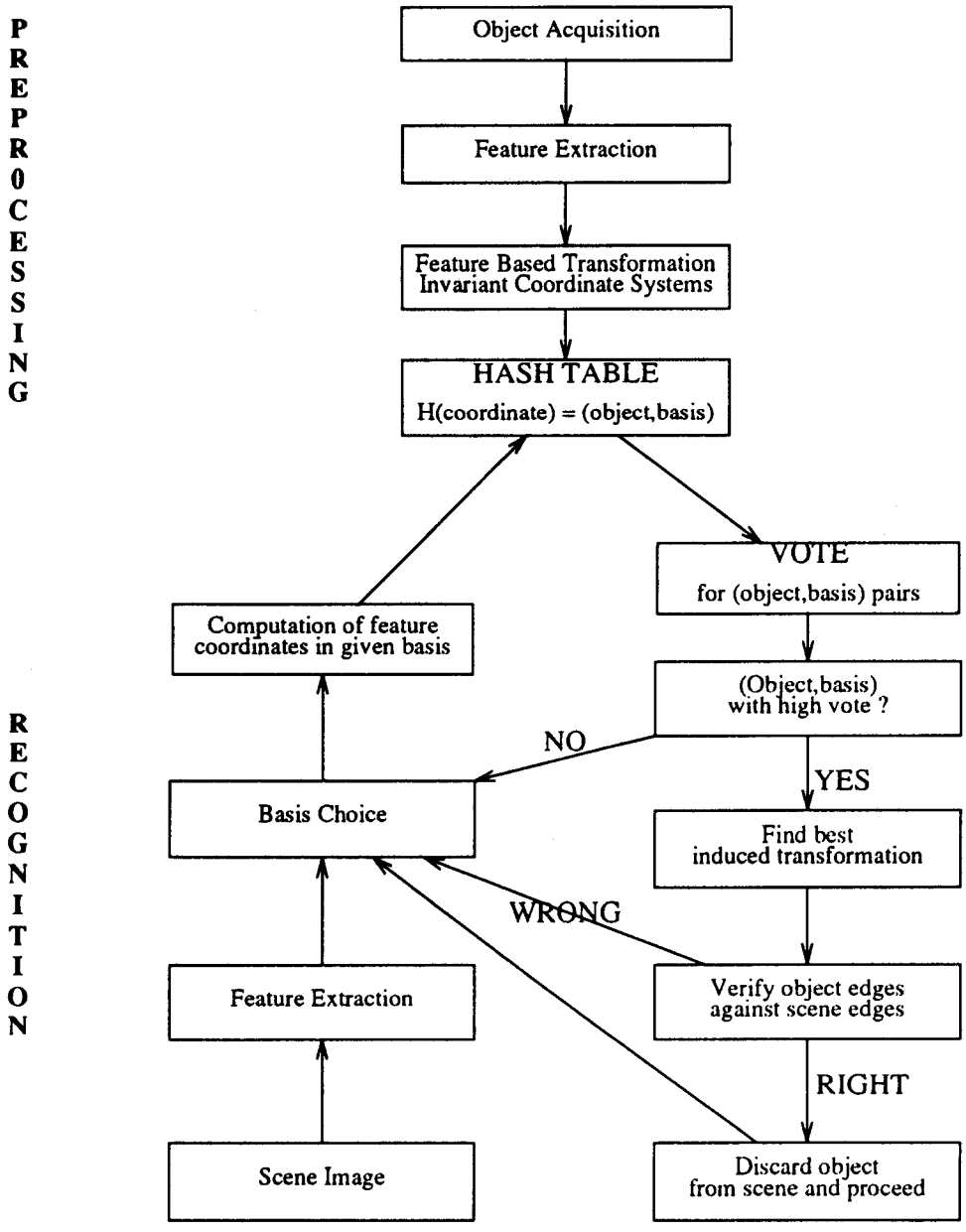

Figure 3. The general scheme of Geometric Hashing.

(1) **Preprocessing stage** (object representation)

For each model do:

(a) Extract a set of interest points (let their number be m).

(b) Determine the minimum number k of basis points needed to define a basis in which all the other $m-k$ points have transformation invariant coordinates. This number depends on the dimension of the space and the transformation involved.

(c) For each such k-tuple (a focus of attention) do: compute the coordinates of the other $m-k$ points and for each point store at the hash entries indexed by the coordinates of the hash element (*model*, *k-tuple*).

(2) **Recognition stage** (matching)

(a) Extract a set of interest points (let their number be n).

(b) Choose a k-tuple of non-collinear points as a basis (focus of attention), and compute the coordinates of the rest of the scene points with respect to this basis. If there is no k-tuple left, the recognition process stops with no recognition.

(c) For each coordinate access the hash table

and vote for all hash elements (*model, k-tuple*) appearing there. Each such vote suggests the presence of a certain model together with the transformation given by the correspondence of the model k-tuple and the scene k-tuple under investigation. The vote accumulator has $\sum_{i=1}^{i=N} m_i^k$ entries where N is the number of models and m_i the number of interest points of model i.

(d) Step through the vote accumulator and look for (*model, k-tuple*) pairs that received more votes than a certain threshold. The maximum number of votes possible for an object with m_i points is $m_i - k$. If no pairs have received more votes than the threshold, go back to step (b), else continue.

(e) Track all points that voted for a candidate transformation and induce additional point correspondences. Use a least square method to find the best transformation for the points involved, this is supposed to be more accurate than the transformation based solely on the correspondence of the k-tuple basis points.

(f) Verify the candidate model and transformation by transforming *all* edges (not only interest points) in the scene accordingly and comparing them with the model edges. If this step confirms the presence and orientation of an object, then recognition is complete (if more objects can be expected in the scene, the search continues, after removing all interest points in the scene belonging to the recognized object). If this step rejects a candidate solution, another candidate solution offered by step (e) is analyzed. If all candidate solutions are rejected, we are back to step (b).

Some important points can be made here:

- Geometric Hashing should *not* necessarily be taken as a conclusive recognition algorithm. Step (d) will generally produce more than one solution. Geometric Hashing is basically a fast filtering procedure where promising candidate solutions are selected from the vast amount of possible transformations and then offered to a verification method.

- The algorithm described above is independent of the choice of interest points. For example, a database with polyhedral objects would suggest to take vertices as interest points, but also more complex geometric features, or even non-geometrical features like high variance in intensity values could be used [6]. Moreover, one can use the above scheme using different types of interest points at the same time. The idea is that corresponding points in the model and scene must be of the same type. The advantage of this is that, in the recognition stage, voting takes place only in the relevant part of the hash table and is thus more efficient and reliable. In general, the more specific the features represented by the interest points, the more effective Geometric Hashing will be. The actual extraction of image features however is done by a layer that lies *under* Geometric Hashing [2, 8].

- Because models may vary in the number of interest points they have, it makes sense to define the threshold in step (d) of the recognition process relative to this number. That means that we may introduce a global threshold which specifies the fraction of point correspondences out of the maximum which are needed to accept the solution as a candidate. So if t_r is the relative threshold and object i has m_i points, the number of votes v model i needs is:

$$v \geqslant t_r \cdot m_i - k.$$

Doing this, threshold t_r is a direct measure of the amount of occlusion we tolerate. At least $t_r \cdot 100\%$ of an object must lie unoccluded in the scene. At the same time one could still have a global absolute threshold to eliminate candidate solutions of models which have relative few interest points and are likely to have arisen by accident. t_a gives the minimum number of votes v a model i needs:

$$v \geqslant t_a.$$

2.3. Complexity analysis

Let us examine the complexity of the presented algorithm. In [6] Lamdan and Wolfson argue that for a single model with m points and a scene with n points the worst time complexity is $O(m^{k+1})$ in the preprocessing stage, and $O(n^{k+1})$ in the recognition stage. They notice that in the preprocessing stage, there is a maximum of m^k different basis k-tuples, each requiring $m-k$ hash element insertion operations. In the recognition stage, there is a maximum of n^k different basis k-tuples, each requiring $n-k$ coordinates to be

computed. This is certainly true, but having a large hash table, it is not the computation of the coordinates that requires the effort at this stage, it is the actual voting process. Therefore the quantity that determines the worst time complexity in the recognition stage should be the amount of votes given, which has in the general case the complexity of:

$$O(\#hash\ elements \times \#hash\ accesses(rec.)) \quad (1)$$

if there are N models each with m modelpoints, this gives:

$$O(N \times \#basis\_prep \times m \times \#basis\_rec \times n) \quad (2)$$

which gives:

$$O(N \times m^{k+1} \times n^{k+1}) \quad (3)$$

if all possible k-tuples are chosen as basis. Equation (1) is based on the observation that the amount of votes given in the recognition phase is proportional to the number of hash elements and hash accesses, for a certain distribution of hash accesses and elements. It is *this* complexity that one should use while comparing, as in [6], the Geometric Hashing with other techniques like alignment and Hough transform. This complexity was confirmed in the experiments described in Section 4.

It can be seen that the number of interest points in the preprocessing and recognition stage (m and n) has a great impact on the complexity of the algorithm. Choosing interest points such, that they represent salient features, has the advantage that it reduces m and n. The time complexity can be reduced even if both m and n are large by using both in the preprocessing and recognition stage only a subset of all possible k-tuples as basis. This subset could be based upon a relation R between k interest points that is invariant with respect to the transformation performed and to use in the preprocessing and recognition stage only those k-tuples as basis which have relation R.

2.4. Error analysis

We will give an error analysis for the 2D affine transformation, although much applies also for the 3D case. There are basically two types of errors involved, errors that occur due to the quantization of the hash table and measurement errors in the location of the interest points.

Due to the hash quantization in the preprocessing stage two hash entries can be stored in the same hash bucket though the indexing coordinates were different (see p_1, p_2 and p_3 in Figure 4). That means that in the recognition stage we will vote not only for the hash elements associated with a particular coordinate, but also for those associated with the neighbouring coordinates (for example, p_1 would vote for p_2 and p_3 too in Figure 4). This quantization error increases with the cell size of the hash table. (We will prevent that the coordinate of a point falls beyond the coordinate range of the hash table (like p_4 in Figure 4) by placing restrictions on the allowable basis choices).

In the recognition phase, we have an error in the measurement of the coordinates of an interest point. The coordinates can be represented as a vector in 2D space with an appropriate norm, usually the Euclidean L_2 or the maximum coordinate norm L_∞. Assume that the measurement of interest points in the scene introduces a maximum error ε in a certain norm.

For affine transformations we had to compute in the recognition stage the coordinates x of a fourth point p with respect to basis triplet (b_0, b_1, b_2). This can be formulated as finding the solution of the linear system [7]:

$$Ax = c$$

where, if b_0 is the new origin, the columns of A are the vectors $b_1 - b_0$ and $b_2 - b_0$, and c is the

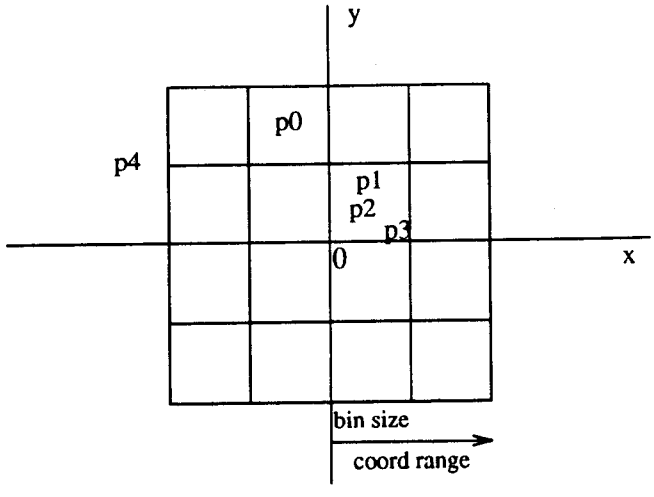

Figure 4. Quantization errors.

vector $p - b_0$. Taking noise in consideration, the question now is:

Given the maximum error ε on the triplet of basis points (b_0, b_1, b_2) and on the fourth point p, what is the error on the vector q which gives the coordinate of p relative to this basis?

As was noted in [7], this problem can be stated as finding the solution of the linear system:

$$(A + \delta A)(x + \delta x) = c + \delta c$$

where δA represents the error on matrix A, and δx, δc the errors on vectors x and c. In this case, the absolute entries of δA and δx are less than 2ε.

In Golub [4] a treatment of the above linear system, expanding x in its Taylor serie, leads to the following results:

$$x + \delta x = x + A^{-1}(\delta c - \delta A\, x) + O(e^2),$$

$$|\delta x| \leqslant |A^{-1}(\delta c - \delta A\, x)| + O(e^2) \quad (4)$$

$$\leqslant |A^{-1}|(|\delta c| + |\delta A| \cdot |x|) + O(e^2) \quad (5)$$

$$\leqslant |x| \cdot k(A) \cdot \left(\frac{|\delta A|}{|A|} + \frac{|\delta c|}{|c|} \right) + O(e^2) \quad (6)$$

where $k(A) = |A| \cdot |A^{-1}|$ is the condition number of the matrix A. Inequalities (4), (5) and (6) give a stepwise rougher first-order estimate of the maximum error $|\delta x|$ which can be introduced when computing the coordinate of a fourth point with respect to a basis triplet. But note that δA and δc, as well as their norms, are unknown in our case.

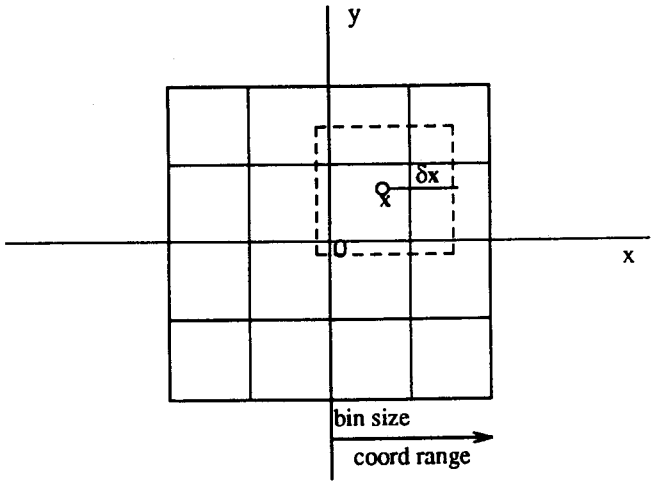

Figure 5. Voting for a neighbourhood.

The thing to do is to use an upper bound of these norms in (5) or (6), which depends on the maximum error ε. For example in L_∞ norm this will be 4ε for δA and 2ε for δc.

The difference with the error free algorithm presented so far, is that in the recognition stage, we do not vote solely for the hash bin indexed by the coordinate of x but we also vote for all hash bins which lie in a neighbourhood of $|\delta x|$. See Figure 5 for the L_∞ case. Note that due to the hash table quantization error, we vote for more than a neighbourhood of $|\delta x|$. If the coordinate of x falls out of the hash table range, we vote for the intersection of the neighbourhood $|\delta x|$ and the hash table range.

Equations (4), (5) and (6) are independent of the norm used. In the case of similarity transformation the column vectors of A are orthogonal (Section 2.1) and $k(A) = 1$ in L_2 case and at most 2 in L_∞ norm. In the case of affine transformation, $k(A)$ is not necessary bounded and that can introduce a great uncertainty $|\delta x|$. Another source of large errors arises when the length of the basis vectors are small with respect to the error ε and if $|x|$ is big. Since it is possible to estimate whether a basis will introduce a relative large amount of error (by checking $k(A)$ or checking whether the length of basis vectors lies under a threshold) on forehand, it makes sense to skip these triplets without entering the voting proces. The presence of a certain model can still be recovered from other triplets in the image.

In the case of L_∞, we can obtain a sharper estimate on $|\delta x|$ than (5), by using that

$$|\delta A\, x| \leqslant 2\varepsilon(|x_1| + |x_2| + \cdots + |x_k|),$$

rather than the general $|\delta A\, x| \leqslant |\delta A|\, |x|$.

This new concept of 'voting for a neighbourhood' has some important implications for the voting scheme. Consider in a hash table those hash elements belonging to a particular model and a particular basis triplet. See Figure 6.

Four hash elements were stored in the hash table at the bins corresponding with points $p_1, \ldots, p_4$. Assume that during recognition the same bins were accessed, with tolerance δx as shown in Figure 6. In this situation point p_1, for example, would vote 3 times for the *same* (*model, basis*) combination, at

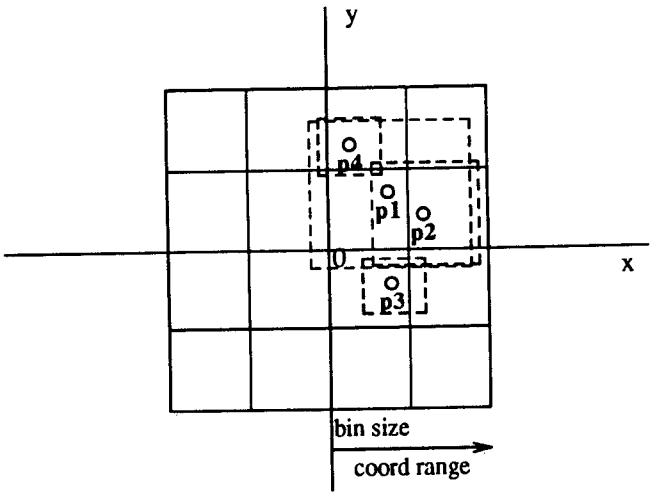

Figure 6. Vote count in a neighbourhood.

p_1, p_2 and p_4. In addition, p_2 would vote for the second time for the same (*model*, *basis*) combination, at location p_1. The problem is that one coordinate may vote more than once for the same (*model*, *basis*) combination and that several coordinates may repeatedly vote at the same bin for a (*model*, *basis*) combination. That is, a scene point may be associated with more than one model point and, conversely, a model point may be associated with more than one scene point. See Figures 7a and 7b.

This makes the vote count scheme, as discussed so far, an unreliable measure for the similarity between model and scene. What is needed is a criterium to select candidate solutions, which takes into account that *one* scene point (s.p.) can correspond only to *one* model point (m.p.). So candidate solutions should be based on the *maximum number of distinct point correspondences* between the model and scene point set. If the maximum number of different correspondences is higher than a certain threshold accept, else reject. This is illustrated in Figures 7a and 7b. The maximum number of distinct point correspondences in Figure 7a is 3 (m.p. 1 with s.p. 1, 2 or 3, m.p. 2 with s.p. 4, and m.p. 3 or 4 with s.p. 5) while in Figure 7b it is 4 (m.p. 1 with s.p. 2, m.p. 2 with s.p. 1, m.p. 3 with s.p. 3 or 5, and m.p. 4 with s.p. 4). Note that the 'vote count', as discussed before, which is the number of links, is higher in Figure 7a than in Figure 7b.

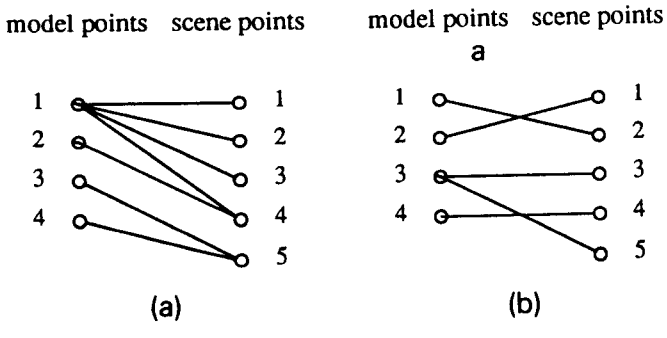

Figure 7. Distinct correspondences.

The notion of the maximum number of distinct correspondences between two disjunct point sets appears also in Discrete Mathematics as the size of the 'maximum match'. It involves computing all possible subsets S of a point set P. A more efficient way is to determine the number of different scene points and different model points involved in the correspondences and to use the minimum of these two as criterium:

$$\min(\#mpoints, \#spoints).$$

In some cases this criterium will give a count that is higher than when using the maximum number of distinct correspondences criterium, but this would be compensated by the straightforward computation of this criterium.

To be able to calculate these correspondences, we include in a hash element (*model*, *triplet*) the model point which was responsible for this entry. In the recognition stage, voting by incrementing a counter of a (*model*, *basis*) combination is replaced by adding a model-scene point correspondence to the combination.

2.5. From 3D to 2D

So far, we have only considered the case where the object and scene representation have the same dimension in space. When recognizing a 3D object from a single 2D image, there is an additional problem of the reduced dimension of the image space, compared with the model space. This problem can be solved by representing a 3D model by its 2D projections. When an orthogonal projection on the viewing plane is assumed, the two projections of an object taken from the same viewing angle are in similarity correspondence. The set of viewpoints with equal distance to the center of the model, at different viewing angles, represents a sphere, the

viewing sphere. In the model representation stage we tessellate the viewing sphere and represent a 3D model by its 2D (model) projections at the different discrete viewing angles.

For each of the model projections we have to compute coordinates of visible model interest points with respect to a chosen basis. For each coordinate the quadruplet (*model, viewing angle, basis, point*) is stored in the hash table.

In the recognition stage we are looking for a 2D similarity transform between a scene and a certain 2D model projection. Votes will be given for triplets (*model, basis, viewing angle*). A triplet with high score indicates the presence of a model with the specified orientation. Since the number of model projections is finite, the projections which lie between the model projections will be seen as having a tessellation error ε_T compared with the nearby model projections. In other words, their interest points are within an absolute distance of ε_T to the corresponding points of a nearby model projection, if any. In the recognition stage we will make use of the results of Section 2.4 and supply the algorithm with an error ε that is high enough to account for both the tessellation error ε_T and the measurement error ε_M.

Let us examine the *model representation* stage in more detail. Its aim is to represent a 3D model by such a set of 2D projections that *any* projection of the model will be recognized as one of its nearby model projections, using a value for ε as discussed in Section 2.4 based on the tessellation error ε_T. There exists a strong relationship between the tessellation error ε_T and the tessellation constant. The smaller the tessellation constant is (finer tessellation grid), the smaller ε_T can be expected to be and vice versa. Therefore, one way to guarantee the successful recognition of any projection of a 3D model could be to use an error ε based on an upper bound of ε_T, given a certain tessellation constant of the viewing sphere. It turns out that although it is straightforward to derive such an upper bound for a 3D model of radius R, this results in a very rough and inpractical upper bound. The algorithm will be able to work with a much lower ε, since:

- The recognition of a projection is not necessary for *all* its bases with the *maximum* number of point correspondences (votes). It is sufficient that the projection obtains enough votes for certain basis choices to distinguish it from random solutions.
- It is not the absolute deviation of individual points, compared with their corresponding points of a nearby model projection, that is significant and should be less than $\varepsilon = \varepsilon_T$. Important is the relative position of the shifted points with respect to the shifted basis.
- The error formula (5) results itself in a $|\delta x|$ that corresponds most of the time to a higher error than the ε actually provided.

In absence of a useful theoretical relation between the ε and the tessellation constant, a reasonable approach to assure that a certain tessellation is fine enough for a value of ε, is to check this explicitly by experiments. In particular, when using a triangular tessellation of the viewing sphere, one could check for all projections that correspond to the midpoint of the triangles, whether they are recognized, using the chosen ε. A projection is considered recognized when it is recognized for a certain fraction of its bases with a certain fraction of point correspondences. The requirement that a projection is recognized for different basis choices enhances robustness, in view of occlusion and noise.

Observing that the structure of a 2D projection varies in certain areas of the viewing sphere more than in others, it would be wasteful to have an overall tessellation based on that needed for the areas of the viewing sphere where the structure of a 2D projection varies rapidly. Therefore the idea of a uniform tessellation is abandoned in favor of the following more efficient approach. We start as before with a rather coarse triangular tessellation of the viewing sphere (say at intervals of 15 degrees), choose a certain ε_R to work with, and check for each triangle whether the projection corresponding with the midpoint is recognized using that ε_R. If this is the case, we proceed with the next triangle. If not, split the triangle in four smaller triangles and continue recursively the process on each of the smaller triangles. Each time a triangle is split, three new model projections are added to the model base, corresponding to the midpoints of the edges of the original triangle. The idea is that new model projections are only added

to the model basis whenever the chosen ε_R proves insufficient to map non-model projections to existent model projections. The final result of this process is that the whole viewing sphere is covered. Quite understandable, the error ε used in the recognition stage will have to satisfy $\varepsilon \geqslant \varepsilon_R$, in order to maintain the integrity of the recognition algorithm.

Observe that all actions described above take place in the model representation stage, which is done off-line. The result is an efficient representation of a 3D model by its 2D projections.

If the object under consideration is symmetrical the number of 2D projections N can be reduced by a factor. Also, if the viewpoint is constrained to some region, like in some industrial applications where the position of the camera with respect to the object is known, N can be further reduced. The advantage of representing a 3D model by its 2D projections is that constraints on viewpoint are easily incorporated.

3. Implementation

3.1. GEOHASH

To recognize 3D objects from 2D images, a sequential version of the Geometric Hashing concept has been implemented for the 2D similarity transformation. The program is called GEOHASH and was written in C. The model representation stage consists of an initial uniform tessellation, followed by a recursive split-process. The hash table is multi-dimensional, which facilitates the hashing of different types of interest points. The output consists of candidate point correspondences between various models and scene. The last step of Geometric Hashing, finding the best transformation in least square (Figure 3), was not included at this point. All error analysis, discussed before, is included. In addition GEOHASH allows the user to specify all the noise related parameters, such as the noise ε, hash bin size, and the thresholds.

3.2. Memory requirements

The key operation during recognition is accessing the hash table and voting. To get high speeds the hash table and voting accumulator should be in the main memory. So we will examine the memory requirements of the three data structures: (1) the vote accumulator, (2) the initial hash table, and (3) the hash elements.

(1) The vote accumulator keeps track of the vote count for a certain model-basis combination. It needs storage for $\#models \times \#bases\_prep$ entries. Noticing that a range of 0 to 255 is quite sufficient for the vote count, it is enough to use one byte. Therefore:

$$\text{memory requirements} = \#models \times \#bases\_prep \text{ bytes.}$$

For a typical 20 3D models, having a number of 2D model projections that corresponds to a uniform tessellation with patches of 10 degrees each, we have $\#models = 20 \times 36 \times 18 = 12960$. Taking the number of allowable basis choices per model to be 15 (which is quite reasonable), this gives:

$$\text{memory requirements} = 12960 \times 15 = 194 \text{ Kb}$$

which is not a major problem.

(2) The initial hash table needs storage for

$$nhash\_tables \times 4 \times \left(\frac{hash\_coord\_range}{hash\_bin\_size} \right)^2$$

hash bins. In this expression, *nhash_tables* denotes the number of 2D hash tables needed which depends on the dimensionality of the hash table. Notice the quadratic dependence on the bin size. Each hash bin requires a pointer to the set of hash elements stored at the bin and space to store the scene points which mapped onto this bin at recognition time, in order to track the point correspondences of the solutions. Assume a mean of n bytes is sufficient for the latter purpose. Taking four bytes for a pointer, this gives:

$$\text{memory requirements} = nhash\_tables \times 4 \times \left(\frac{hash\_coord\_range}{hash\_bin\_size} \right)^2 \times (4 + n) \text{ bytes.}$$

For $nhash\_tables = 5$, a typical range of 0 to 10, a hash bin size of 0.1 and a mean $n = 5$, this gives:

$$\text{memory requirements} = 5 \times 4 \times 100^2 \times 9 = 1.8 \text{ Mb}$$

which should be taken in consideration.

(3) There are $\#models \times \#bases\_prep \times (\#modelpoints - k)$ hash elements stored in the hash table, where k denotes the number of interest points of a basis. Each hash element specifies a model, a basis point pair, and the model point responsible for this entry. Taking a short, two characters and a character respectively for these entries, and taking two bytes for a short, one byte for a character, we get:

$$\text{memory requirements} = \#models \times \#bases\_prep \times (\#modelpoints - 2) \times 5 \text{ bytes.}$$

Taking again $\#models = 12960$, mean $\#bases\_prep = 15$ per model, and mean $\#modelpoints = 20$ per model, this gives:

$$\text{memory requirements} = 12960 \times 15 \times 18 \times 5 = 17.5 \text{ Mb}$$

which is quite a lot to have in main memory, even for these days.

As could be expected, the storage of the hash elements is the determining factor. The example presented seems to represent a limit case: a total memory requirement of just below 20 Mb could be supplied in main memory of a SPARC-compute server. Note that an object database of 20 3D objects is quite sufficient for most industrial applications. If needed, one may lower the mean $\#bases\_prep$ per model which would allow a larger number of 3D models.

4. Experiments

4.1. Random experiments

Geometric Hashing is a filtering method which selects from the vast amount of possibilities the promising solutions. The filtering ratio, between all possible solutions and the ones Geometric Hashing presents, should be *significant enough* to make Geometric Hashing applicable. As Lamdan and Wolfson note in [7], the filtering ratio of Geometric Hashing is equal to the probability that a certain random solution has received 'by accident' more votes (based on a certain criterium) than the threshold T.

Another interesting issue is the complexity of the Geometric Hashing scheme, which is determined by the number of votes given in the recognition stage. How is this influenced by parameters like the number of models, the number of interest points, the hash bin size and the error? We would also want to examine the typical distribution of the hash elements in the hash table.

In order to investigate these problems for the *similarity transformation*, random experiments have been performed with large 2D object databases of $N = 50$, $N = 100$ and $N = 250$. The models consisted each of 15 points and the scene had 25 points. All interest points were considered of the *same* type. The models and scene consisted of random dots with the coordinates (x, y) uniformly distributed in the range of 0–511. All possible pairs of points were chosen as basis, in the preprocessing stage as well as in the recognition stage. In the preprocessing stage one pair of points accounted only for one basis choice (one of the two orderings), in the recognition stage it accounted for two basis choices (both orderings). This means that in the case of $N = 250$ there are approximately $250 \times (15 \times 14/2) \times 13 = 3.4 \times 10^5$ hash elements in the hash table, and in the recognition stage there are approximately $25 \times 24 \times 23 = 13800$ hash accesses. These figures are taken so large to get a good estimate of the natural distribution of hash elements in the hash table and the probabilities involved.

The following parameters of GEOHASH were used in all random experiments, unless stated otherwise:

abs. threshold = 5,

rel. threshold = 0.0,

hash coord. range = 10.0

$(-10 \leqslant x \leqslant 10$ and $-10 \leqslant y \leqslant 10)$,

min. dist. basis = 50.0,

hash bin size = 0.1.

The min. dist. basis sets a lower bound on the tolerated distance between basis points to avoid introducing a large error in the coordinates of the third point, see formula (5).

4.1.1. Hash table distribution

The hash table distribution in the case of $N=250$ was calculated. Examining this hash table distribution, we noticed a bell shaped (Gaussian-like) distribution, highly centered around the origin. The hash elements in the coordinate range $|x|$, $|y| \leqslant 2$ account for 89% of all hash elements. Conversely, the distribution of hash elements not only tells something about the preprocessing stage, but it can also be interpreted as a probability distribution for hash accesses in the recognition stage. Therefore it can be expected that for 89% of all hash accesses in the recognition stage we will have a $|x|<2$ in L_∞ and that is interesting to know because it influences the error $|\delta x|$ in the error formula (5). Further, if we want to distribute the hash elements uniformly over the hash bins, we can use the standard error function (i.e., the integral of the Gaussian distribution) to convert the coordinates of the hash elements, before accessing the hash table.

4.1.2. Hash bin size

Increasing the hash bin size will increase the hash table quantization error. This is illustrated for the $N=50$ case with error $\varepsilon=2.0$.Table 1 lists the number of solutions that received 'by accident' k votes for different bin sizes.

It can be seen that a larger bin size substantially increases the number of random solutions at any level. It seems that it would be a good idea to take the bin size as small as possible. Indeed, this would make the voting scheme as accurate as possible, but the cost for this is increased memory requirement of the hash table, especially if the latter is multi-dimensional. Also, making the bin size small means that in the voting part there will be many more bin accesses needed for the same voting neighbourhood. This slows down recognition. On the other hand, the recognition time would also increase if a large bin size was taken; more solutions are needed to be processed in the retrieval part, which is relatively time consuming. What is needed is a compromise on the value of the bin size, a size in the range of 0.05–0.1 (corresponding to a hash table of 400×400 to 200×200 bins) seems most suitable. In the 3D object recognition experiments of the next section, a bin size of 0.1 was chosen.

Table 1
Effect of bin size on vote distribution

#votes	bin size = 0.05	bin size = 0.1	bin size = 0.2
5	1730	23271	175451
6	265	5816	103494
7	47	1280	55713
8	14	215	26831
9	4	27	11125
10	1	7	3687
11	0	0	900
12	0	0	127
13	0	0	9

4.1.3. Significance of solutions

To examine the probability that a certain random solution receives more than k votes, we examine the fraction of *all* possible solutions that has received more than k votes. In general, if we have N models with m points and a scene with s points, the number of all possible solutions is $N\times m(m-1)\times s(s-1)/2$ if each point pair is chosen once as a basis in the preprocessing stage and twice in the recognition stage. Therefore, we have to divide by this number. If we take k to be our absolute threshold, then the computed probability represents the filtering factor of Geometric Hashing. The smaller this factor is, the more effective Geometric Hashing is.

Experiments have been performed to compute the probability that a certain random solution receives k votes for various error levels ($\varepsilon=0.0$, 1.0, 2.0 and 3.0). See Table 2. It was found that this probability is *not* affected by the number of model basis choices in the preprocessing stage, apart from a statistical error. One may assume that

Table 2
Probabilities of random solutions

#votes	$\varepsilon=0.0$	$\varepsilon=1.0$	$\varepsilon=2.0$	$\varepsilon=3.0$
5	6.1e-4	2.4e-3	7.8e-3	2.1e-2
6	8.9e-5	4.7e-4	2.0e-3	7.0e-3
7	9.7e-6	7.5e-5	4.1e-4	1.9e-3
8	8.1e-7	9.0e-6	7.1e-5	4.4e-4
9	6.7e-8	8.1e-7	9.1e-6	8.4e-5
10	0.0	0.0	8.1e-7	1.5e-5
11	0.0	0.0	0.0	2.6e-6
12	0.0	0.0	0.0	2.7e-7
13	0.0	0.0	0.0	6.7e-8

the probability distribution is neither dependent on the number of scene bases chosen. Yet the number of model and scene points does affect the probability distribution. The probabilities given in Table 2 are based on the $N=250$ case.

Using these figures, we can get an impression of the number of false solutions with k votes that will occur, given there are 15 model points and 25 scene points, all of the same type, and given the other fixed parameters (table bin size, rel. threshold, min. distance basis etc). For example, consider the case of $\varepsilon=2.0$. It is expected that 4.1×10^{-4} of all possible solutions will have 7 votes at this error level. Assume we have an object database of 12960 2D models (which represent 20 3D models at a tessellation of the viewing sphere of 10 degrees) with a mean number of 15 basis choices per model and 20 basis choices in the scene. In this case, the number of all possible solutions is $12960\times15\times20=3.9\times10^6$. It is expected that approximately $3.9\times10^6\times4.1\times10^{-4}=1599$ random solutions will receive 7 votes. In order to avoid these random solutions, the relative threshold must be set higher than 60%. In general, the higher error level, the higher relative threshold must be set to allow a good filtering ratio, and the less occlusion can be tolerated.

4.2. 3D object recognition

In order to examine the performance of Geometric Hashing in the case of 3D object recognition, experiments have been performed on a database of six polyhedral 3D objects. The objects under consideration were the six letters M, A, R, L, E and N in 3D (see Figure 8).

The objects were represented with a wire frame model. The actual database consisted of the 2D projections of these objects at discrete viewing angles and was built using the method described in Section 2.5. Since the 3D models M, A, E and N have an axis of symmetry, only half of their viewing sphere was tessellated. A 2D projection of a 3D model at a certain angle was obtained using the program HIDLINPX [1], which also did hidden line removal. HIDLINPX was also used to generate scenes for a specific angle. Simulating a camera of 512×512 pixels looking at a scene, HIDLINPX scales the 2D projection to a size of 400×400, taking into account that a camera is never fully zoomed in.

In these experiments, the interest points are the *vertices* and the bases are the *vertices connected by an edge*. Connectivity is indeed a transformation invariant property. Since the choice of vertices as interest points is hardly sufficient to capture the essence of a scene, the vertices were labeled by the number of outgoing edges. So the interest point can be of different type. The case of a T-vertex, the intersection of one edge occluded by another, will be considered of type 1. For a correspondence between a model and a scene point, we require both to be of the same type. This means that for a scene point to vote for a certain hash element, the scene point must have the same type as the model point of the hash element, and both model and scene bases must match. This is implemented efficiently by using a multi-dimensional hash table, where, in addition to the coordinates of a point, the type of the point and the type of its basis are part of the index.

In addition, we require that all outgoing edges of corresponding points match as well. In order to match the edges, their relative orientation with respect to the basis under consideration is computed. This outcome is invariant under the similarity transformation. Two edges are considered to correspond whenever their difference in relative orientation is less than 30 degrees. Important is to observe that with these extensions, matching is still done

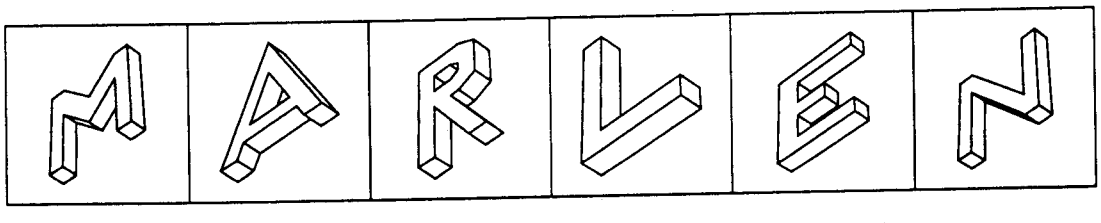

Figure 8. The letters M, A, R, L, E and N at ($\phi=45°$, $\psi=135°$).

Table 3
Summary of model representation stage

3D Letter	#models ($s=15$)		#models ($s=7.5$)		#models ($s=3.75$)	rest
M (1/2)	140	+	86 (42/278)	+	10 (4/168)	0
A (1/2)	140	+	52 (24/278)	+	0 (0/96)	0
R (1/1)	278	+	59 (24/552)	+	10 (4/96)	(4/16)
L (1/1)	278	+	246 (117/552)	+	87 (35/468)	(11/140)
E (1/2)	140	+	73 (28/278)	+	13 (5/112)	(5/20)
N (1/2)	140	+	81 (42/278)	+	30 (12/168)	(1/48)

on basis of information *locally* available at interest points: vertices and orientation of outgoing edges.

In order to save memory, a pair of basis points accounted only for one basis choice in the preprocessing stage. Consequently, the recognition stage needed to consider both basis orderings, whenever they were indistinguishable by type.

4.2.1. Model representation

Table 3 summarizes the results of the model representation stage for the 3D letters M, A, R, L, E and N. The following parameters were used:

abs. threshold = 3,
rel. threshold = 0.75,
hash coord. range = 10.0
($-10 \leqslant x \leqslant 10$ and $-10 \leqslant y \leqslant 10$),
min. dist. basis = 40.0,
hash bin size = 0.1,
error level $\varepsilon = 2.75$.

The method used here was an initial uniform triangular tessellation of the viewing sphere with tessellation constant of 15 degrees. This was followed by a recursive SPLIT process where triangles were split and new model projections were added, whenever needed, using the method described in Section 2.5. A test projection was considered recognized when more than 20% of the edges resulted in a successful basis, which identified a nearby model projection with more than 75% of the possible point correspondences. A model projection was considered nearby if it was within 15 degrees, in terms of both spherical angles ϕ and ψ. An error level ε in the range 2.0–3.5 proved the most suitable in terms of the trade-off between the number of 2D projections needed and the number of random solutions that came up.

See Table 3. The column under $s=15$ lists the initial number of model projections, 140 or 278, depending on whether the half (1/2) or the whole (1/1) of the viewing sphere was tessellated. The number of model projections added at the first split of triangles is listed under the column $s=7.5$ (the 'size' s of the smallest triangles is now 7.5). The ratio between brackets denotes the ratio of unsuccessful recognitions of midpoints of triangles to the total number of midpoints checked, at this step. In general, most of the projections that were indeed recognized, were recognized for a much higher percentage of their total scene bases than the 20% required, typically for more than 50%. The $s=3.75$ column lists the results of the second recursive step of the split-algorithm, resulting in some triangles of size 3.75. For each unsuccessful recognition at the $s=7.5$ step (nominator fractions between brackets) we have now four new tests (column $s=3.75$, denominator fraction), corresponding to the midpoints of the 4 smaller triangles. The results of the third step are listed under the last column 'rest', no more model projections have been added. The unsuccessful recognitions at this point deal with projections with highly unstable 2D structure.

4.2.2. Scene recognition

The presented algorithm has been tested for several scenes. Figure 9 shows three of them. None of the projections in the scenes are model projections. The relative threshold, i.e., the fraction of occlusion tolerated, was set to $t_r=0.75$, $t_r=0.7$ and $t_r=0.5$, for the scenes of Figure 9 from left to right. Other parameters, such as the error level ε, remained the same as in the preprocessing stage. *All* allowable bases choices were computed, in

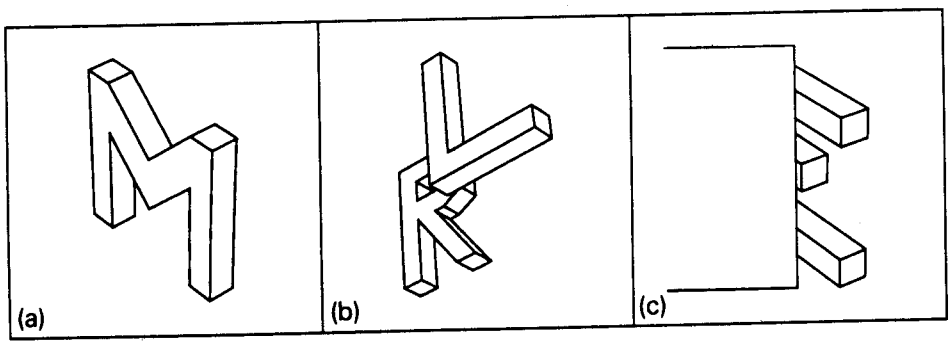

Figure 9. (a) Scene M, (b) scene RL, (c) scene E.

random order. What remains is an examination of the reliability of Geometric Hashing in the basic case. See Table 4 for the resulting vote distribution. The numbers between brackets denote the number of correct solutions within 15 degrees plus (+) the number of correct solutions within 15–30 degrees, in terms of both spherical angles ϕ and ψ.

In such a distribution, the correct solutions, if any, should be among the solutions that have come up multiple times, with high vote counts and high percentage of votes out of the maximum. In the case of the unoccluded M at $\phi = 51$ and $\psi = 53$, all the solutions that have acquired the highest three vote counts, represent indeed the correct solution.

Table 4
Vote distribution at recognition

#votes	scene M	scene RL	scene E
3	0	0	21
4	0	0	87
5	30	0	0
6	0	26 (10+6)	0
7	10	3	20
8	0	4 (1)	3
9	0	6 (5)	1
10	4	0	20 (17+3)
11	4	0	7 (7)
12	8	0	0
13	14	0	0
14	4	0	0
15	11 (9+2)	0	0
16	4 (4)	0	0
17	13 (11+2)	0	0
18	0	0	0
19	0	5 (4+1)	0
20	0	8 (8)	0

For the RL scene, the vote distribution is split in two clusters. The cluster at the higher vote counts consists entirely of the correct solution R, while the one at the lower vote counts strongly advocates the second correct solution L. Finally, the scene of the occluded E is also correctly recognized by all of the solutions with the highest vote counts. This last scene illustrates one of the strengths of Geometric Hashing, the local matching of interest points, independently of the others. The occluding surface has split the visible part of the E in three separate, not connected segments. It is also interesting to examine what other solutions are suggested by the algorithm, at the lower vote counts. In this case, the vote counts at 7, 8 and 9 all suggest M or N lying on the side and mark the lower and upper visible E part as the legs of M or N.

As can be seen, using Geometric Hashing in this way achieves such a high filtering ratio that it implicitly provides the correct solutions. It also provided the correct point correspondences between model and scene points. Moreover, the recognition is robust; the correct solutions came from bases at different edges (Table 5). A verification procedure has only a light task left.

The runtime performance of Geometric Hashing was encouraging. Using a SPARC workstation, the average CPU time needed per basis considered in the scene was 0.8 s, 0.6 s and 1.0 s. Taking into account only the successful bases, these figures were 1.5 s, 0.7 s and 2.1 s. It is expected that the use of a massive parallel machine like the Connection Machine will reduce the above figures dramatically, since separate processors can be assigned to parts of the hash table and vote accumulators.

Table 5
Number of successful bases

	#edges visible	#edges allowed as bases	#edges succesful as bases
M	23	20	5
R	22	8	7
L	13	6	6
E	12	12	11

5. Conclusions

In this paper the Geometric Hashing technique was discussed for model-based recognition. The focus has been 3D object recognition from 2D images. Several results from random experiments and scenes containing 3D objects were presented. An efficient model representation scheme was proposed, based on the representation of a 3D model by a non-uniform set of 2D model projections. The underlying idea was to require that any 2D projection of a model to be within an error distance of ε to nearby model projections in order to be recognizable. It has been found that distinguishing between different types of interest points results in a very efficient implementation of Geometric Hashing, with a multi-dimensional hash table. The filtering ratio of Geometric Hashing turned out to be high enough to identify the correct solution(s), with the correct point correspondences. The algorithm was also rather successful in dealing with occlusion, and performed at speeds in the order of one second per basis on a SPARC station.

The Geometric Hashing approach, as any other structural matching method, seems somewhat less suited for applications with high noise levels and for applications with loosely defined models. Having an intensive preprocessing stage, Geometric Hashing places heavy memory demands which can limit the number of models which fit in the database. On the bright side, actual recognition is very efficient and at high speeds. Together with the inherent high parallelism and the ability to deal conceptually simple with occluded scenes, it makes Geometric Hashing a promising model-based recognition method for the future.

Acknowledgements

The authors thank Prof. R. Hummel for bringing the Geometric Hashing under their attention, providing several useful ideas.

References

[1] Ammeraal, L. (1986). *Programming Principles in Computer Graphics*. Wiley, New York.

[2] Ballard, D.H. and C.M. Brown (1982). *Computer Vision*. Prentice-Hall, Englewood Cliffs, NJ.

[3] Chin, R.T. and C.H. Dyer (1986). Model-based recognition in robot vision. *ACM Computing Surveys* 18(1), 67–108.

[4] Golub, G.H. and C.F. van Loan (1983). *Matrix Computations*. J. Hopkins Press, Baltimore, MD.

[5] Kalvin, A., E. Schonberg, J.T. Schwartz and M. Sharir (1986). Two-dimensional, model-based, boundary matching using footprints. *Int. J. Robotics Research* 5 (4).

[6] Lamdan, Y. and H.J. Wolfson (1988). Geometric Hashing: a general and efficient model-based recognition scheme. *Proc. Second Int. Conf. on Computer Vision*, Tampa, FL, 238–249.

[7] Lamdan, Y. and H.J. Wolfson (1989). On the error analysis of 'Geometric Hashing'. NYU Robotics Research Report No. 213.

[8] Lowe, D.G. (1985). *Perceptual Organization and Visual Recognition*. Kluwer Academic Publ., Dordrecht.