International Journal of Image and Graphics
Vol. 9, No. 2 (2009) 287–298
© World Scientific Publishing Company

A GPU-BASED ALGORITHM FOR APPROXIMATELY FINDING THE LARGEST COMMON POINT SET IN THE PLANE UNDER SIMILARITY TRANSFORMATION*

DROR AIGER

Department of Computer Science
Ben Gurion University, Be'er Sheva, Israel
aiger@cs.bgu.ac.il
and
Orbotech LTD, Yavne, Israel
dror-ai@orbotech.com

KLARA KEDEM

Department of Computer Science
Ben Gurion University, Be'er Sheva, Israel
klara@cs.bgu.ac.il
and
Computer Science Department
Cornell University, Ithaca
kedem@cs.cornell.edu

Received 17 April 2008
Revised 27 September 2008
Accepted 3 March 2009

We consider the following geometric pattern matching problem: Given two sets of points in the plane, P and Q, and some (arbitrary) $\delta > 0$, find the largest subset $B \subset P$ and a similarity transformation T (translation, rotation and scale) such that $h(T(B), Q) < \delta$, where $h(.,.)$ is the directional Hausdorff distance. This problem stems from real world applications, where δ is determined by the practical uncertainty in the position of the points (pixels). We reduce the problem to finding the depth (maximally covered point) of an arrangement of polytopes in transformation space. The depth is the cardinality of B, and the polytopes that cover the deepest point correspond to the points in B. We present an algorithm that approximates the maximum depth with high probability, thus getting a large enough common point set in P and Q.

The algorithm is implemented in the GPU framework, thus it is very fast in practice. We present experimental results and compare their runtime with those of an algorithm running on the CPU.

Keywords: Approximation algorithms; randomized algorithms; GPU; computational geometry; geometric pattern matching; Hausdorff distance.

*This work was partly supported by the MAGNET program of the Israel Ministry of Industry and Trade (IMG4 consortium).

1. Introduction

Pattern matching is pervasive in many areas of computer science. It is of special importance in computer vision, where it directly corresponds to fundamental vision problems like object registration and recognition. The first step in any machine vision application, and the one that usually determines whether the application succeeds or fails, involves locating the object within the camera's field of view, a process known as pattern matching. It is also of importance in computational drug design and computational biology, where it has been successfully used for identification of drug molecules with similar shapes (and presumably similar chemical properties). Shape matching is an important ingredient in shape retrieval, recognition, classification, alignment, registration, approximation and simplification.

A central problem in these applications, is the question of whether two point sets P and Q in the plane *resemble* each other under a given family of transformations. A common measure of resemblance is the minimum Hausdorff distance. The Hausdorff distance (HD) between two point sets P and Q is defined as

$$H(P, Q) = \max(h(P, Q), h(Q, P)),$$

where $h(P, Q)$ is the directional Hausdorff distance from P to Q:

$$h(P, Q) = \max_{p \in P} \min_{q \in Q} d(p, q).$$

Here, $d(\cdot, \cdot)$ represents a standard metric (L_2 or L_∞). The minimum Hausdorff distance $D(P, Q)$ is defined as $H(T(P), Q)$, where T is a transformation from the given family that gets $H(T(P), Q)$ to the minimum.

In some applications a constant δ is given, determined by the uncertainty in the position of the input points (P and Q), and one is then interested in the *decision problem*, namely, whether a transformation T that brings P to Hausdorff distance smaller than δ, from the points of Q, exists; and if so, one wishes to find this transformation. A slightly more general problem allows outliers — the question is similarly posed but now we want to find a transformation T that brings the largest subset $B \subset P$ to Hausdorff distance smaller than δ, and, report also the points in B. This problem is called *the largest common point set* (LCP).

Our paper is motivated by practical pattern matching problems under similarity transformations. The data in our experiments comes from the rasterized (pixel) world, thus we are interested just in the decision problem. Moreover, in our kind of data we must take into account outliers, therefore we solve here the approximate LCP problem.

Geometric algorithms that solve pattern matching problems (including LCP) suffer from the *curse of dimensionality* in the dimension of transformation space. Huttenlocher *et al.*[1] find the minimum Hausdorff distance between P and Q under translation in time $O(n^3 \log n)$, where $n = max\{|P|, |Q|\}$. Chew *et al.*[2] solve exactly the minimum HD under rigid transformation requiring $O(n^5 \log^2 mn)$ time. Choi and Goyal[3] solve the LCP problem (under rigid transformation) approximately, the runtime of their algorithms is $O(n^6 \log n)$. To get over this hurdle we provide a GPU-based algorithm and compute approximately the LCP.

As has been done before (see, e.g. Refs. 4 to 8) our solution is based on reducing the LCP problem to that of finding the deepest point in an arrangement of polytopes in transformation space which is 4-dimensional in our case. We present in this paper an approximation scheme that allows speeding up the computation in the GPU framework, improving, on the fly, the results in Aiger and Kedem[4] for a similarly stated problem, but in 3-dimensional transformation space (rigid transformation, or translation and scale). We provide few examples of the implementation, and compare our runtime with an implementation of the algorithm of Irani *et al.*[9] on the CPU.

Finding the deepest point in an arrangement of objects in $3D$ shares common ideas with the computation of shadow volumes in computer graphics (see a survey by Crow[10]). The problem we solve is more challenging since the objects we deal with may have many intersections (details in the paper). Modern graphics hardware have built-in capabilities that allow fast computation of a point of maximum depth.

Throughout the paper we assume that the scaling parameter is constrained to be above a minimum value (to avoid the trivial case that all points in the pattern collapse into a single point).

2. The GPU as a Stream Computer

A graphics processing unit or GPU is a dedicated graphics rendering device for a personal computer, workstation, or game console. Modern GPUs are very efficient at manipulating and displaying computer graphics, and their highly parallel structure makes them more effective than typical CPUs for a range of complex algorithms. Recently, many GPU-based algorithms for geometry, image processing and other problems have been considered by researchers (see, e.g. Refs. 11 and 12 for more details). In particular, the GPU as a stream computer for geometric optimization was considered by Ref. 13. Today's GPUs have a large number of processors (about 256), thus as in the application we present here, performing a computation on the GPU is far faster than performing it on the CPU. The GPU provides parallelism where each pixel on the screen can be viewed as a stream processor, enabling an application to be computed in highly parallel mode. Modern graphics hardware come now with a specially designed software, e.g., the NVIDIA's CUDA, which enables programmers and developers to write their software for the GPU as easily as coding in C.

3. Approximating the Depth of Grid Points on a Plane in $3D$ in one Rendering Pass

We start with some definitions as in Ref. 5.

Definition 1. We define the δ-neighborhood of a point q in the plane to be all the points in R^2 that have distance $\leq \delta$ from q.

Note. Throughout this work we use L_∞ as the underlying metric, as we did in Ref. 4. However, the GPU method allows us to work with the L_2 metric without any changes to our methods and algorithm. In the plane the δ-neighborhood of a point is an axis parallel square, of side-length 2δ, under L_∞, and a circle of radius δ for L_2.

Definition 2. Given a set $\mathcal{S}$ of convex polytopes in R^d and a point $q \in R^d$, we define the *depth* of q in $\mathcal{S}$, $depth(q, \mathcal{S})$, to be the number of polytopes of $\mathcal{S}$ containing q. The depth of $\mathcal{S}$, $depth(\mathcal{S})$, is defined as $max_q depth(q, \mathcal{S})$ over all $q \in R^d$.

We represent the similarity transformation T by its linear parametrization as follows. Let $(x', y') = T(x, y)$, then

$$x' = ax + by + c, \tag{1}$$

$$y' = -bx + ay + d, \tag{2}$$

where $a = s\cos(\theta)$, $b = s\sin(\theta)$, s is the scale, θ is the rotation, and (c, d) is the translation vector. We describe the similarity transformation as the point (a, b, c, d) in R^4.

Consider the set of similarity transformations that map a point $p \in P$ exactly to a point $q \in Q$. The transformations that correspond to Eq. (1) describe a hyperplane in R^4 parallel to the d-axis. The transformations that correspond to Eq. (2) describe a hyperplane in R^4 parallel to the c-axis. The set of transformations that map p to q is thus a 2-flat which is the intersection of these two hyperplanes. Based on this parametrization we define a δ-stick in the transformation space:

Definition 3. Let R_δ be an axis parallel square, of side-length 2δ, in the (c, d) plane centered at $(c, d) = (0, 0)$. A δ-stick is the polytope in R^4 determined by the Minkowski sum of R_δ with the 2-flat described above.

Claim 1. The set of all similarity transformations that map p to the δ-neighborhood of q is a δ-stick in transformation space.

In Ref. 4 we describe a GPU-based algorithm for the approximation of point set matching in 3-parameter transformation space (i.e. rigid transformation or scale + translation). It computes an approximate LCP of P and Q under these transformations, for a given δ. The algorithm is based on counting levels in arrangements of polytopes in R^3, and is implemented on the GPU by depth peeling.[14] In this section we improve the results of Aiger and Kedem,[4] and present a faster and simpler GPU-based algorithm for approximating a point of maximum depth in an arrangement of polytopes in $3D$. This result is one of the building blocks of the $4D$ algorithm discussed in Sec. 5.

Let $\mathcal{S}$ be any set of convex polytopes in R^3 bounded by a cube in R^3, and let W be a given plane in this cube. We are given an approximation parameter α that determines a grid on R^3 and on W. Let U be the set of α-spaced grid points on W.

April 18, 2009 20:3 WSPC/164-IJIG 00345

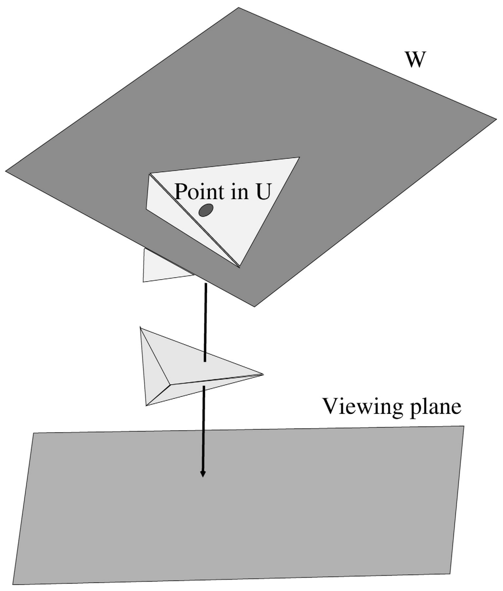

Fig. 1. A point on W belonging to the grid U, the ray coming out of it, the viewing plane, and two polytopes.

Our goal is to approximate the depth of each point in U in one rendering pass over $\mathcal{S}$ and W.

The term *rendering* refers to the process of drawing a $3D$ object on the frame buffer, given a viewing point and the object's geometry. While objects are rendered to the frame buffer, the Z-buffer holds the updated distance of any rendered pixel from the viewer (or when parallel projection is used, from the viewing plane). This is essentially the (rasterized) lower envelope of the arrangement of the $3D$ objects. The frame buffer contains the color of the object thus it can be used to store a pointer to the object, on the envelope, at each pixel. The stencil buffer allows us to increment or decrement a counter for each pixel. We count in this buffer the number of *front* faces and *back* faces (definition below) placed between W and the viewer. (See Ref. 14 for a detailed description of the computer graphics terms above.)

Our algorithm works as follows. We set the viewing plane to be parallel to the plane W and use parallel projection rendering. We initialize the Z-buffer to a minimum value. Then we render all the points in U (by simply drawing W using a resolution of α and updating the depth buffer). We refer to a convex object as an intersection of halfspaces and we say that a face is a *front* face if the viewing point is outside the halfspace that defines the face, and that a face is a *back* face if the viewing point is inside the halfspace containing the face. We render the objects in $\mathcal{S}$ in two stages. First, the front faces are rendered while incrementing the stencil buffer for each pixel that is closer than the depth buffer value. This counts the number of front faces in front of every point in U. In the second stage, we do the same with the back faces, but now decrementing the stencil for each pixel closer than the value in the depth buffer. We thus count the back faces in front of the points in U. When done, the stencil buffer holds, for each point of U, the number of objects (front faces minus back faces) that contain this point. We thus compute the depth of all the points of U in one pass over the data of polytopes in $\mathcal{S}$ and the points in U.

In our previous work[4] $\mathcal{S}$ consisted of δ-sticks in R^3. By using the above method, instead of the *peeling* method applied in Ref. 4, we managed to make that algorithm much simpler and faster.

In the following section we construct the set of polytopes $\mathcal{S}$ for our approximate LCP problem for similarity transformations. As we will demonstrate, the set $\mathcal{S}$ is a collection of slabs in R^3.

4. Using Random Projection

Cardoze and Schulman[15] show how to solve the geometric pattern matching problem of finding a rigid transformation that brings P approximately to the δ-neighborhood of Q (allowing a small error, ε, in the distance). They discretize the plane into a γ-size grid, G, for $\gamma = \frac{3\delta\varepsilon}{5}$, and convert the coordinates of the input points into integer grid coordinates. Let us denote by P', respectively Q', the converted points of P, respectively Q. Then they add to Q' the $O(n\delta\gamma^{-2})$ grid points of G, which are in the δ-neighborhood of each point in Q. Finding a translation that brings the maximum number of points of P' to coincide with the points in Q' is an approximation of the LCP for P and Q.

They further show, that for the integer case, the question of finding the best translation in 2-dimensional space can be reduced to finding the best translation on a random line in the plane, and that the latter translation, when carried back to $2D$, will be the best translation in $2D$ with high probability.

Their algorithm is as follows. They pick a random line u in the plane and project the input points (P' and Q') on that line. They define a fine grid on u, so that the probability that the best translation on the line, when carried back to $2D$, will not be the best translation $2D$ is very small. Then, they use FFT to find all best translations on this line from the projection of P' to the projection of Q'. Thus they get a randomized algorithm that succeeds with high probability to find the translation in $2D$. (See Ref. 15 for more details.)

Following the ideas in Ref. 15, we get rid of one translation parameter, thus converting our $4D$ problem into a $3D$ approximation problem. Now, instead of looking for the approximately optimal $4D$ similarity vector (a, b, c, d), we look for an approximately optimal $3D$ vector (a, b, t). For simplicity we first describe the idea in the continuous space. For a given point $p \in P$ and a point $q \in Q$, let q_t be the projection of q on u (see Fig. 2). Let p_1 denote the transformation of p by scale and rotation (parameters a, b), and let p_t be the projection of p_1 on u. We have (applying scale and rotation):

$$p_1 = (ap_x + bp_y, -bp_x + ap_y),$$

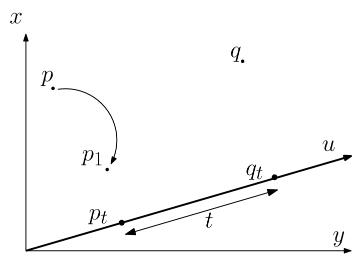

Fig. 2. Reducing $2D$ translation to $1D$ by projection onto a random line.

where p_x and p_y are the x and y coordinates of p respectively. Then (applying scale, rotation and dot product with u):

$$|p_t| = ap_xu_x + bp_yu_x - bp_xu_y + ap_yu_y,$$
$$|q_t| = q_xu_x + q_yu_y,$$

where $|\cdot|$ is the length of the vector. The mapping from p to q_t on the vector u satisfies:

$$ap_xu_x + bp_yu_x - bp_xu_y + ap_yu_y + t = q_xu_x + q_yu_y, \tag{3}$$

where t is the shift on the line u. This is a plane in R^3 (for fixed p, q, u) and with parameters a, b (from Eqs. (1) and (2)) and t on the line u. We denote this plane by N.

Recall that the fine grid size on u was defined so that, with high probability, only one point of Q' will be projected to one grid interval on u. For a fixed pair of points $p \in P'$ and $q \in Q'$ as above, there is a transformation (a', b', t') that maps p to q_t (and with high probability to q). Allowing t to vary from one end of the grid cell of t' on u to the other, the plane N sweeps a *slab* in transformation space (a, b, t). Our set of polytopes $\mathcal{S}$ consists of the slabs assigned to all pairs (p, q), $p \in P'$ and $q \in Q'$.

Consequently, it follows from Ref. 15 that the similarity transformation that brings the largest number of points in P' close to points of Q' can be found by finding the point (a_0, b_0, t_0) that is covered by the maximum number of slabs in R^3. This is the second building block in the algorithm which we present below.

Note that in the exact setting in Ref. 5 we required the δ-neighborhood of the points in Q to be pairwise disjoint in their interiors, in order for the depth to correspond to the cardinality of the LCP of P and Q. Here this restriction is relaxed by adding $O(\delta\gamma^{-2})$ grid points around each point of Q, and by refining the cell size on u.

5. Algorithm, Implementation, and Experiments

The algorithm. We are given two point sets P and Q in the plane, $\delta > 0$, and $\varepsilon > 0$. We wish to find a similarity transformation that brings the maximum number of points in P to the δ-neighborhood of points in Q. We first give a brief overview of our algorithms: we round all points in P and Q to a planar grid determined by ε. Next we project the $4D$ transformation space to a $3D$ transformation space, creating the objects in this space that correspond to transformations which map each $p \in P$ to every $q \in Q$. We then find a point in the arrangement of these objects in 3-space that is covered by the maximum number of objects (maximum depth). To this end we apply our GPU algorithm. Returning back to $4D$, the point we thus found corresponds to a similarity transformation as required.

In more detail we do:

- Draw the grid G of cell-size γ on the plane.
- Convert the coordinates of each point of P into its closest grid's integer coordinates, creating P'.
- Convert the coordinates of each point of Q into its closest grid's integer coordinates, initiating Q', and add to Q' all the $\delta\gamma^{-2}$ grid points in the δ-neighborhood of each point of Q.
- Pick a random line u on the plane and convert our $2D$ translation into $1D$ translation, and the $4D$ transformation space (a, b, c, d) into a $3D$ transformation space (a, b, t).
- Construct the slabs, one for each pair (p, q), for all $p \in P'$ and $q \in Q'$.
- Determine a grid in transformation space so that for every p in P' and every two centers, T_1, T_2, of neighboring cells $|T_1(p) - T_2(p)| \leq \frac{\gamma}{2}$. Let α from Sec. 3 be that grid size.
- For each slab, we set its lower plane to be the plane W as in Sec. 3 and set U to be the grid points on W. We compute the depth of the arrangement of all slabs for all points of U. (Clearly, the deepest point in the arrangement is in one of the slabs, and hence can be computed, without loss of generality, for the lowest plane in the slab.)
- We pick the slab, and grid point of maximum depth on it, as our result.

Complexity. In total we need one rendering pass for each plane, meaning $O(mn)$ passes. (If $m = |P|$ and $n = |Q|$ then there are $O(mn)$ such planes, one for each $p \in P$ mapped to each $q \in Q$). Naively, we have to render m^2n^2 polygons in total. However, by applying a randomization scheme from Aronov and Har-Peled,[16] we can reduce the number of objects in our arrangement and with it the number of passes, so the number of rendered polygons in total gets down to $O(n^2)$. According to Ref. 16, we sample our set of objects by taking every object with probability $O(\frac{1}{m} \log mn)$ (with constants that depend on the desired approximation). We compute the point of maximum depth in the arrangement of this sample set and we get a "good enough" approximation for the maximum depth in the entire set, with high probability. This means that instead of finding the point of maximum depth, D, we approximate it to find a point of depth $(1 - \varepsilon)D$ with high probability where ε determines the approximation. For details on this approximation see Refs. 5 and 16.

Note (see below for detailed implementation) that the actual number of rendered polygons can be lower and their size is very small since for each pass we compute the depth of all (discrete) points that lie on a specific plane which is the lower plane of the object in turn (we go over all objects), thus we are only interested in the polygons that intersect this object. To conclude, we render $O(n^2)$ small polygons in total but the practical number can be smaller.

The implementation. We implemented the GPU algorithm on a PC Pentium 3.3 Ghz with Nvidia GEforce7800 using OpenGL. We should note that modern GPUs now have advanced capabilities that can be used by modern languages like Nvidia CUDA, but at the time we wrote this code it was not available. Clearly, using such modern hardware and software would have resulted in much better time performance for the same algorithm.

Since all objects are slabs with limited width, we can limit our rendering to a small slab around each plane. We use OpenGL function *glClipPlane()* to set a clipping region which is a small slab around the plane in process. For each plane among the lower planes of all objects we apply the algorithm described in Sec. 3 in one rendering pass. Following each pass, we have to get the point of maximum value from the stencil buffer. In CUDA, fast stream reduction is possible but our implementation does not use CUDA. Still, this can be done without reading back the buffer to main memory by parallel reduction techniques[17] implemented as a fragment program.[11] We use the simplest implementation that runs in $O(\log N)$ steps and overall $O(N)$ time where N is the number of pixels.

Since the stencil buffer has only 8 bits in most hardware, there is theoretically a potential overflow if there are more than 255 objects counted during a pass for a single point in the stencil buffer. To prevent this, we apply a process of adding the stencil buffer values to 24 bits buffer for every chunk of 255 polygons rendered and then reset the stencil. This again is done without reading back the buffer. Modern hardware eliminates this need. Once all passes are finished and the point $P = (x, y)$ in the stencil buffer, corresponding to the point of maximum depth in $3D$, is computed, P corresponds to a ray in $3D$ where the point of maximum depth (in $3D$) must lie. We apply one more pass to compute the actual $3D$ point by OpenGL selection mode and get the desired output.

We give experimental results for a number of input sizes and error parameters. In all tables, $|P| = |Q| = n$, ε is the error (pixel size). For finding the depth in the arrangement of these planes in $3D$, we use the method described in Sec. 5 on the GPU to get $O(n^2)$ time GPU algorithm. In all tests the range for the parameters a and b (Eqs. (1) and (2)) was $[0.2, 2]$. The input sets are points taken randomly from $[-1, 1] \times [-1, 1]$. We used $\delta = 0.01$ for all tests. An example of the point sets can be seen in Fig. 3 for $n = 100$. The time is given in seconds. For comparison, we implemented on the CPU the algorithm of Irany and Raghavan[9] which is a randomized faster version of the practical algorithm of Goodrich *et al.*[18]

The experiments are summarized in Tables 1 and 2, taking $m = n$.

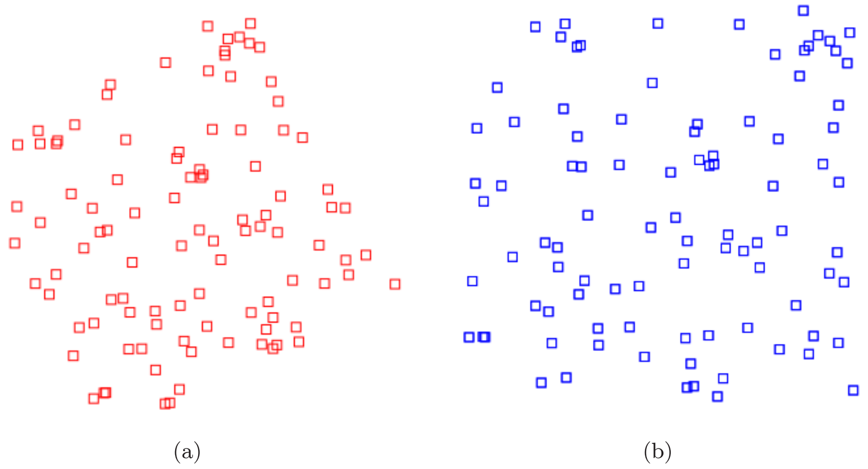

Fig. 3. The point sets P and Q for $n = 100$. P contains the points of Q, rotated by 0.5 radians and scaled by a factor of 0.8, with added position noise.

Table 1. Running time (in seconds) for number of input points. In practice we get quadratic time in the number of points in Q.

n	ε	Our GPU algorithm time	Randomized alignment time[9]
50	0.02	0.008	0.078
100	0.02	0.031	0.485
200	0.02	0.079	2.937
400	0.02	0.269	37.469
800	0.02	1.231	—
1600	0.02	3.943	—

Table 2. Running time (in seconds) for various ε.

n	ε	time
100	0.02	0.031
100	0.01	0.081
100	0.005	0.245
100	0.0025	0.714

Limitations. While being fast in many practical cases, our GPU algorithm has some limitations. The input should be bounded so that it can be rendered onto the GPU frame buffer that has bounded memory. The approximation error, ε, should not be too small. Our algorithm can become expensive both in runtime and in the required memory if ε is very small.

References

1. D. P. Huttenlocher, K. Kedem and M. Sharir, "The upper envelope of Voronoi surfaces and its applications," *Discrete and Computational Geometry* **9**, 267–291 (1993).

2. P. Chew, M. Goodrich, D. Huttenlocher, K. Kedem, J. Kleinberg and D. Kravets, "Geometric pattern matching under Euclidean motion," *Computational Geometry: Theory and Applications* **7**, 113–124 (1997).
3. V. Choi and N. Goyal, "An efficient approximation algorithm for point pattern matching under noise," *LATIN*, 298–310 (2006).
4. D. Aiger and K. Kedem, "Applying graphics hardware to achieve extremely fast geometric pattern matching in two- and three-dimensional transformation space," *Inf. Process. Lett.* **105**(6), 224–230 (2008).
5. D. Aiger and K. Kedem, *Geometric Pattern Matching for Point Sets in the Plane under Similarity Transformations*, manuscript.
6. H. S. Baird, *Model Based Image Matching Using Location*, MIT Press, Cambridge, MA (1985).
7. T. M. Breuel, "Fast recognition using adaptive subdivision of transformation space," *CVPR* **92**, 445–451.
8. L. P. Chew and K. Kedem, "Getting around a lower bound for the minimum Hausdorff distance," *Comput. Geom.* **10**(3), 197–202 (1998).
9. S. Irani and P. Raghavan, "Combinatorial and experimental results for randomized point matching algorithms," *Comput. Geom.* **12**(1–2), 17–31 (1999).
10. F. C. Crow, "Shadow algorithms for computer graphics," *ACM SIGGRAPH*, 242–248 (1977).
11. J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krager, A. E. Lefohn and T. Purcell, "A survey of general-purpose computation on graphics hardware," *Eurographics 2005*, State of the Art Reports, pp. 21–51 (2005).
12. K. E. Hoff III, A. Zaferakis, M. C. Lin and D. Manocha, "Fast and simple 2*D* geometric proximity queries using graphics hardware," *SI3D 2001*: 145–148.
13. P. K. Agarwal, S. Krishnan, N. H. Mustafa and S. Venkatasubramanian, "Streaming geometric optimization using graphics hardware," *ESA 2003*: 544–555.
14. C. Everitt, "Interactive order-independent transparency," *Technical Report, NVIDIA Corporation* (May 2001).
15. D. E. Cardoze and L. J. Schulman, "Pattern matching for spatial point sets," *FOCS*, 156–165 (1998).
16. B. Aronov and S. Har-Peled, "On approximating the depth and related problems," *SIAM J. Comput.* **38**(3), 899–921 (2008).
17. D. Roger, U. Assarsson and N. Holzschuch, "Efficient stream reduction on the GPU," *Workshop on General Purpose Processing on Graphics Processing Units* (2007).
18. M. T. Goodrich, J. S. B. Mitchell and M. W. Orletsky, "Approximate geometric pattern matching using rigid motions," *IEEE Trans. Pattern Anal. Match. Intell.* **21**(4), 371–379 (1999).

Dror Aiger is a PhD student at the Computer Science Department of the Ben-Gurion University of the Negev, Israel. He received his BSc and MSc degrees from the Computer Science Department of the Tel-Aviv University, Israel in 1992 and 1997, respectively. He is currently a researcher in Orbotech Ltd. His main research interests are in the area of computer vision, computational geometry and image processing.

Klara Kedem is a Professor at the Department of Computer Science at the Ben-Gurion University of the Negev. Her main research interests are computational geometry and bioinformatics.