Advanced indexing schema for imaging applications: three case studies

S. Battiato, G. Di Blasi and D. Reforgiato

Abstract: Imaging techniques and applications often require heavy computations for finding the *k*-nearest-neighbour of a given pattern. Texture synthesis, image colourisation and super-resolution are all affected by this issue. Advanced clustering-based indexing schemas over metric spaces speed-up efficiently both *k*-nearest-neighbour and range searches. By using them, we are able to save CPU time without losing quality which would be lost using approximate approaches. Moreover, with the proposed technique we are able to convert a batch process task into a real-time task and, more importantly, it might be run on a typical user-end PC desktop rather than powerful mainframes. It has been shown how the application of recently reported well-known indexing schemas improves the speed performance of the above problems.

1 Introduction

'The amount of data applications is growing fast and there is the need to deal with them'. For example, in bioinformatic, the growth of sequence data is increasing rapidly as well as in networks where the internet size is accelerating exponentially; this growth also affects imaging and computer vision where new methods require more and more image processing. Applications need to perform exact queries such as *k*-nearest-neighbour and range searches from these huge databases. Given their heavy computational cost, approximate searches, which avoid to compute all the distances, are hence preferred: they are usually performed using a binary tree which earns CPU time but, on the other hand, loses quality in the results (see [1]). A possible solution is to use database clustering-based indexing schemas over metric spaces. Many of them have recently appeared in literature (see, for instance, MVP-tree [2], M-tree [3], antipole-tree [4], SLIM-tree [5], FQ-tree [6], list of clusters [7], SAT [8], TSVQ [9], AESA [2–12], iAESA [11]; the reader is also referred to [12] for a survey on this subject). When the metric space is also Euclidean, one can see [13–15], X-tree [16] and CHILMA [17]. After their data preprocessing, they are able to perform exact searches by computing a small fraction of all the distances required by the classic full searches. Moreover, their approximate searches allow to save much more CPU time than the ones executed by simple binary trees.

A clustering indexing schema first groups objects into clusters. Then, for each cluster, it computes centroids and all the distances between the centroids and the objects of the relating clusters. During the searches, it visits only the necessary clusters, pruning those which do not satisfy the triangular inequality property.

Texture synthesis, image colourisation and super-resolution are three important imaging applications which are often difficult because of the big size of the input images. These problems have received a lot of interest as their use is spread across numerous industry groups (e.g. videogames, movies).

In all these problems, a *k*-nearest-neighbour over a metric space with large dimensions has to be computed. This makes them impratical in many cases. In this paper, we show how the introduction of efficient data structures improves the results and performances where the *k*-nearest-neighbour search is key to the algorithm performance. Further applications can be found in the field of photomosaic [18], [19].

We have conducted experiments on the application of five recent different data structures which outperform all the other methods for the efficient computation of the *k*-nearest-neighbour search. They are the antipole tree [4], the list of clusters [7], the TSVQ [9], the AESA [10] and iAESA [11]. We decided to make use each of the above algorithms to show that the time improvement does not depend on the particular property of the data structure but only on the current metric space.

The remainder of this paper is organised as follows: in Section 2, we introduce some basic definitions for clustering on metric spaces and the *k*-nearest-neighbour and range searches problems. In Section 3, we present the data structures we have used with the relative implementation of the *k*-nearest-neighbour search algorithms. Sections 4, 5 and 6, respectively, describe the texture synthesis, the image colourisation and super-resolution problems; timing and number of distance computations with the use of the antipole tree, list of clusters, TSVQ, AESA and iAESA for speeding-up the process of reconstruction for each imaging problem mentioned are shown as well. Section 7 ends the paper with conclusions and future directions.

2 Basic concepts

The aim of clustering is to organise data based on similarity. Basically, the data are split into disjoint clusters such that the items within any group are similar and the items in different groups are dissimilar. Data clustering algorithms can be hierarchical or partitional. The hierarchical algorithms which can be agglomerative (bottom-up) or divisive

doi:10.1049/iet-ipr:20060315

Paper first received 7th October 2006 and in revised form 3rd April 2007

The authors are with the Dipartimento di Matematica e Informatica, Università degli Studi di Catania, Catania, Italy

E-mail: diegoref@dmi.unict.it

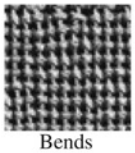
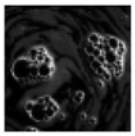
Images	Neighbour	Images	Neighbour
 Texture	{5×5,1} [D]={4096} [S]={75} (a)	 Pills	{5×5,1} [D]={4096} [S]={75} (a)
	{7×7,1} [D]={4096} [S]={147} (b)		{5×5,1} {7×7,1} [D]={1024,4096} [S]={75,147} (b)
	{5×5,1} {5×5,2} [D]={1024,4096} [S]={75,102} (c)		{9×9,1} {9×9,2} [D]={1024,4096} [S]={243,390} (c)
	{7×7,1} {7×7,2} [D]={1024,4096} [S]={147,222} (d)		{9×9,1} {9×9,2} {9×9,3} {9×9,4} [D]={64,256,1024,4096} [S]={243,390,465,492} (d)
 Flowers	{3×3,1} [D]={4096} [S]={27} (a)	 Strips	{5×5,1} [D]={4096} [S]={75} (a)
	{9×9,1} [D]={4096} [S]={243} (b)		{7×7,1} [D]={4096} [S]={147} (b)
	{5×5,1}{5×5,2} [D]={1024,4096} [S]={75,102} (c)		{7×7,1} {7×7,2} [D]={1024,4096} [S]={147,222} (c)
	{5×5,1}{7×7,2} [D]={1024,4096} [S]={75,222} (d)		{9×9,1} {9×9,2} {9×9,3} {9×9,4} [D]={64,256,1024,4096} [S]={243,390,465,492} (d)
 Money	{5×5,1} [D]={4096} [S]={75} (a)	 Bends	{5×5,1} [D]={4096} [S]={75} (a)
	{9×9,1} [D]={4096} [S]={243} (b)		{5×5,1} {5×5,2} [D]={1024,4096} [S]={75,102} (b)
	{5×5,1} {5×5,2} {5×5,1} {5×5,1} [D]={64,256,1024,4096} [S]={75,102,102,102} (c)		{5×5,1} {5×5,2} {5×5,2} [D]={256,1024,4096} [S]={75,102,102} (c)
	{3×3,1} {5×5,2} [D]={1024,4096} [S]={27,102} (d)		{7×7,1} [D]={4096} [S]={147} (d)
 Straw	{9×9,1} [D]={4096} [S]={243} (a)	 Bubbles	{9×9,1} [D]={4096} [S]={243} (a)
	{13×13,1} [D]={4096} [S]={507} (b)		{13×13,1} [D]={4096} [S]={507} (b)
	{5×5,1} {7×7,2} {9×9,2} [D]={256,1024,4096} [S]={75,222,390} (c)		{7×7,1}{9×9,2} [D]={1024,4096} [S]={147,390} (c)
	{9×9,1} {9×9,2} {9×9,3} {9×9,4} [D]={64,256,1024,4096} [S]={243,390,465,492} (d)		{7×7,1}{9×9,2} {11×11,2} [D]={256,1024,4096} [S]={147,390,606} (d)
 Fire	{5×5,1} [D]={4096} [S]={75} (a)		
	{7×7,1} [D]={4096} [S]={147} (b)		
	{7×7,1}{9×9,2} [D]={1024,4096} [S]={147,390} (c)		
	11×11,1}{11×11,2} [D]={1024,4096} [S]={363,606} (d)		

Fig. 1 *Texture synthesis input images and settings*

Number of distance computations are shown in Fig. 3 and timing is in Fig. 4

(top-down) find successive clusters using previously established clusters, whereas partitional algorithms determine all the clusters in one go.

The schemas we used for comparisons are both hierarchical (antipole, TSVQ) and partitional (list of clusters, AESA). Some of them (e.g. antipole) perform a clustering of the space in order to perform a more efficient k-nearest-neighbour.

When clustering is used with metric spaces, then some restrictions on the similarity measure are imposed. Formally, a metric space M is a set of points with an associated distance function $d{:}M \times M \rightarrow R$, where R is the set of real numbers. For all $x, y, z \in M$, d is required to satisfy the following conditions:

1. $d(x, y) \geq 0$ (positiveness);
2. $d(x, x) = 0$ (reflexivity);
3. if $x \neq y \rightarrow d(x, y) > 0$ (strict positiveness);
4. if $d(x, y) = 0$ then $x = y$(identity of indiscernibles);
5. $d(x, y) = d(y, x)$ (symmetry);
6. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

Well-known metric functions include Manhattan distance, Euclidean distance, string edit distance or shortest path distance through a graph. Different data structures which make use of clustering to index a metric space solve efficiently k-nearest-neighbour and range searches. Given a query object q, a database S, a threshold t and an integer $k > 0$, the range search problem is to find all objects $o \in S$ |dist$(o, q) \leq t$, whereas the k-nearest-neighbour search problem is to retrieve the k-closest elements to q from S. When the dimension of the metric space becomes high (say $\geq$50), the performances of all the existing data structures on range and k-nearest-neighbour searches decrease. This is because of the well-known problem of the curse of dimensionality [20]. How the search complexity exponentially grows with the space dimension has been shown in [21]. A successful way to alleviate the curse of dimensionality is to use approximate and probabilistic algorithms for k-nearest-neighbour search. Many interesting data structures have been proposed in literature [4, 7, 9–11, 22–24]. All of them are able to solve efficiently both k-nearest-neighbour and range searches.

3 Acceleration schemas

3.1 Antipole tree

The first data structure considered is the antipole tree [4]. It consists of a binary tree where each node contains clustering informations on two subsets of the input. The clustering is performed by a recursive top-down procedure starting from the given data set of elements and checking at each step if a given splitting condition is satisfied. If it is satisfied,

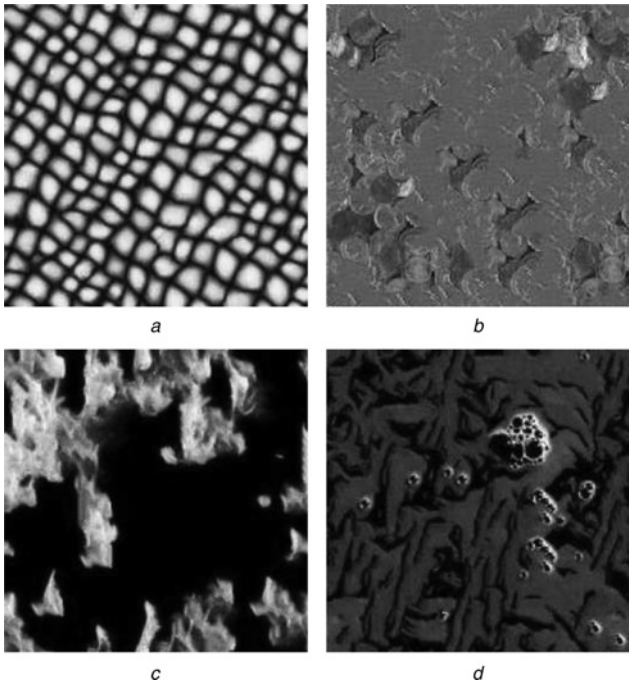


Fig. 2 *Texture synthesis results with exact searches*

Output textures are 256 × 256; input images and experiments settings are shown in Fig. 1
a Texture
b Money
c Fire
d Bubbles

then that pair of objects (A, B), called antipole pair, is generated and is used to split the data sets into two subsets S_A and S_B. Each subset is obtained by assigning each point p of the data set to the subset containing the endpoint closest to p of the antipole (A, B). If the splitting condition is not satisfied, then the given subset is a cluster and an approximate centroid is computed. The splitting condition states that the dist(A, B) is greater than the cluster diameter threshold which is based on a statistical analysis of the pairwise distances of the input set. All internal nodes are annotated with the antipole endpoints and the corresponding cluster radius; leaves contain the centroid of the corresponding final cluster. The building process details are discussed in [4] where it is also shown that its computational cost is at most $O(n^2)$.

3.1.1 *k*-nearest-neighbour search via antipole tree:

The antipole tree solves efficiently the problem of exact and approximate k-nearest-neighbour search.

The exact k-nearest-neighbour search algorithm takes as input the antipole tree T, the query object q and the parameter k indicating the number of elements requested and returns the k closest elements to q. Hjaltason and Samet [25] proposed the incremental nearest-neighbour to perform k-nearest-neighbour search in spatial databases. This approach has been applied to the antipole tree as shown in [4]. The algorithm uses two different priority queues. One is used to store the k objects which will be returned as output and the other stores the subtrees of the antipole which may be visited during the search. The incremental nearest-neighbour search starts by adding the root of the antipole tree in one priority queue. Then, recursively, it extracts the minimum from the priority queue. If the extracted node is a leaf, it visits it. Otherwise, according to the subtree's radius, the distance of the antipole endpoint from the query, a threshold t and the triangle inequality, it visits one of its subtrees.

As discussed in [4], the approximate k-nearest-neighbour search algorithm follows the best path in the tree from the root to the leaf and returns the centroid stored in the leaf node. This algorithm uses the same idea of TSVQ [9] to find quickly an approximate nearest-neighbour of the query object. As for the TSVQ, the timing and accuracy of the solution depends on the size of the tree.

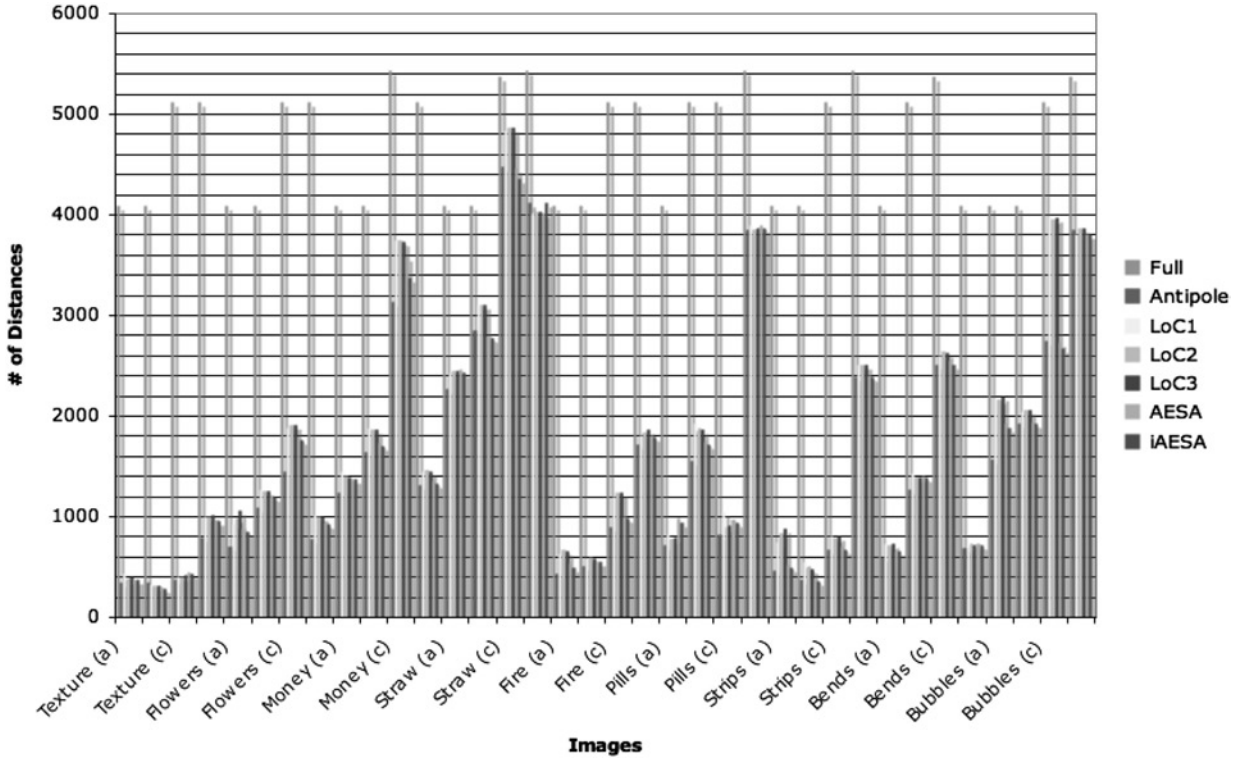


Fig. 3 *Number of distance computations for synthesis of textures in Fig. 1*

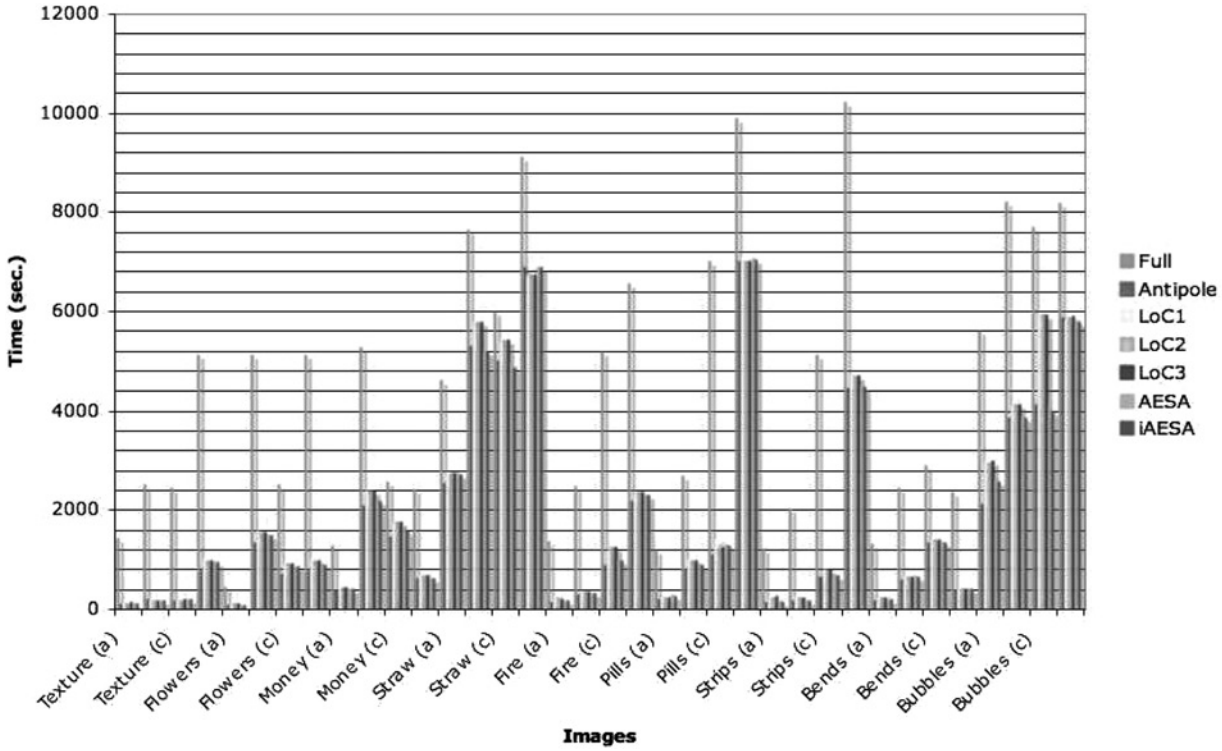


Fig. 4 *Synthesis reconstruction times for textures in Fig. 1*

Comparisons in [4] between the antipole and TSVQ approximate k-nearest-neighbour search show that the antipole tree improves on TSVQ.

As the antipole tree uses a randomised tournament technique to find the local centroids, all the experiments we present in this paper about the antipole tree exact and approximate searches have been averaged over 20 executions.

3.2 List of clusters

The list of clusters has been used in different applications for its effectiveness. As discussed in [7], given a centre c and a radius r, a set of all the objects which are at distance at most r from c is considered as the first cluster. Then the process is repeated recursively on the remaining elements. The building procedure returns a list of triples which would be a list of clusters. Different strategies for choosing a good radius and other algorithmical details can be found in [7] where it is also shown that it costs at most $O(n^{3/2})$ to build this data structure using optimal settings. We implemented the list of clusters as described in [26] with both strategies for the radius selection: partitions of fixed radius and partitions with fixed size. For the centre selection, we used the (p_4) strategy when using clusters of fixed radius; (p_3) and (p_5) strategies have been used in the presence of buckets of fixed size. Our choices have been made according [26] where it is shown that using the strategy above list of clusters computes a lower number of distances in the presence of clusters of fixed radius (p_4) or in the presence of buckets of fixed size (p_1) and (p_5). Throughout the paper, we will refer to LoC_1 as the list of clusters with clustering of fixed radius and (p_4) for centre selection, LoC_2 when using buckets of fixed size and (p_3) for centre selection, and LoC_3 for buckets of fixed size and (p_5) for centre selection. Details on how the heuristics of fixed radius and fixed bucket size affects the performance of the index are discussed in [26].

3.2.1 k-Nearest-neighbour search via list of clusters: The k-nearest-neighbour for the list of clusters works with one priority queue which maintains the current k-nearest objects to the query. First, the starting cluster is analysed and the queue is filled with k elements. Let $d_{\max}$ the distance of the farthest element to the query which lies in the queue. When the queue is full, the triangle inequality is applied in two ways: if a new cluster is being visited then if $d(c, q)-r > d_{\max}$ the cluster is pruned. If one cluster has not been pruned then it is visited and, for each element ei, if $d(c, e_i)-r > d_{\max}$ then the element is pruned, otherwise the distance $d(e_i, q)$ is computed. Comparisons between the antipole and list of clusters can be found in [4]. How the approximate incremental nearest-neighbour search algorithm is applied to the list of clusters is shown in [27].

3.3 Tree structured vector quantisation

One of the most successful data structure is the tree structure vector quantisation (TSVQ) [9]. It works as follows. It first takes as input a set of training vectors and generates a binary-tree-structured codebook; then it computes the centroid of this set and uses it as the root level codeword. The centroid and a perturbed centroid are chosen as children of the root, and locally optimal codewords for them are computed using a generalised Llyod algorithm. The training vectors are split into two groups based on these codewords and the algorithm then recurses in each of them. The algorithm recursively divides the set of training sequences into two different parts in order to get a balanced tree. The process ends when the number of codewords exceeds a chosen size or the average coding error is below a certain threshold. The final codebook is the collection of the leaf level codewords.

3.3.1 k-nearest-neighbour search via TSVQ: TSVQ can be used for fast approximate k-nearest-neighbour search. Starting from the root, the tree is traversed in a best-first ordering by comparing the query with the two

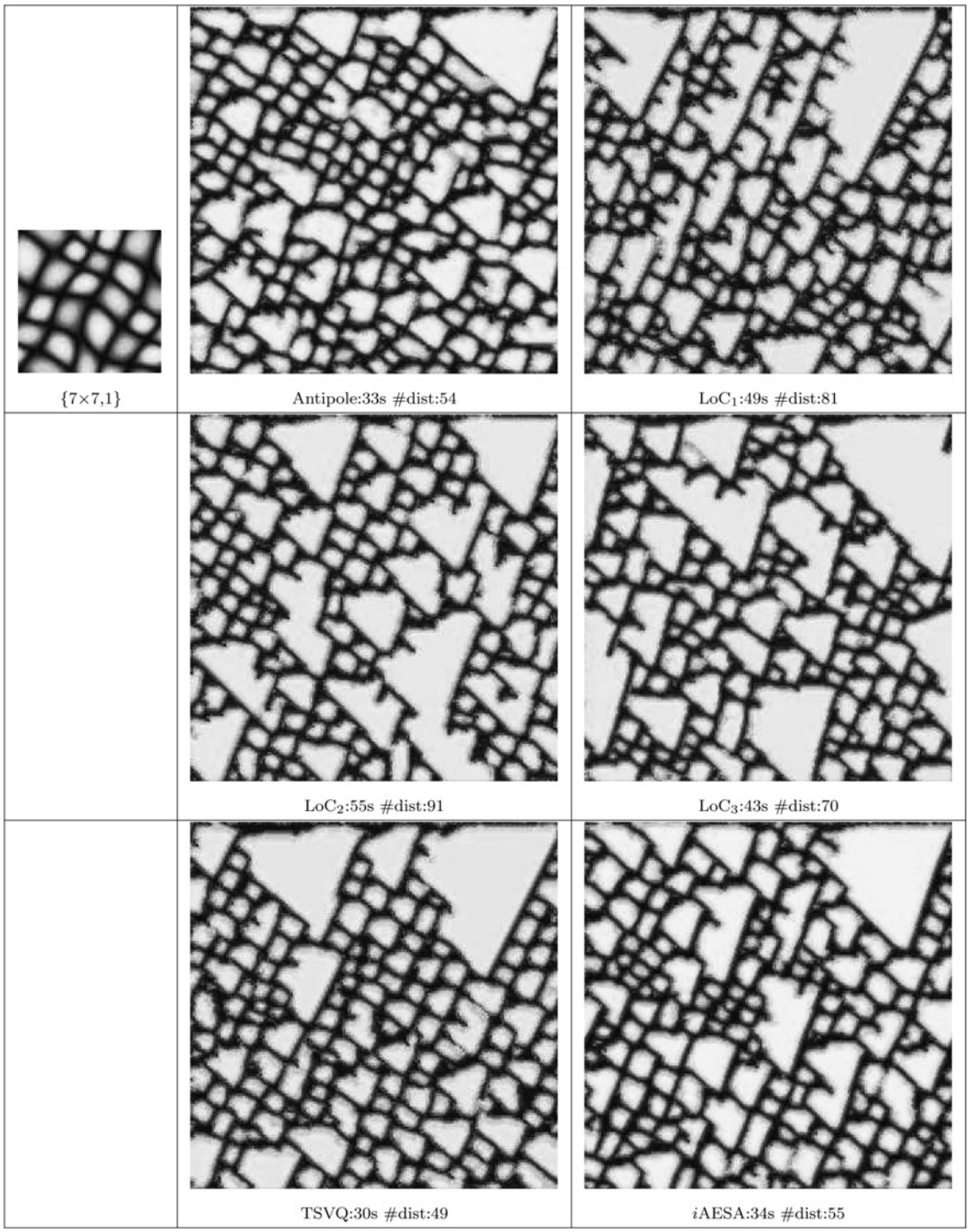


Fig. 5 *Approximate k-nearest-neighbour search comparisons between antipole, list of clusters (LoC_1, LoC_2, LoC_3), TSVQ and iAESA for 'Texture' image*

Input image is 65 × 65, whereas output images are 256 × 256
Under the input image, there is the neighbourhood used; under each produced texture, there is the running time in seconds and the number of distances computations
See Fig. 1 for input image dimensions

children codewords and following the one with a closer codeword. When a leaf node is reached, the search ends and the codeword of the leaf node is then returned. The result codeword is usually close to the optimal solution, and the computation is more efficient than full searching. When the tree is balanced, a single search with codebook size S can be achieved in time O(log S), which is much faster than exhaustive searching with linear time complexity. The TSVQ source code used for the experiments has been downloaded from [28]. Comparisons about the approximate and full searches between the antipole, TSVQ and classical sequential method are shown in [4, 29].

Both the antipole tree and TSVQ are binary trees. The key differences of the antipole tree with respect to the TSVQ are:

- The antipole is able to perform both exact and approximate k-nearest-neighbour searches, whereas TSVQ solves only approximate k-nearest-neighbour search.
- The splitting condition of the antipole tree uses a randomised tournament technique and does not make use of the Llyod algorithm as TSVQ does.

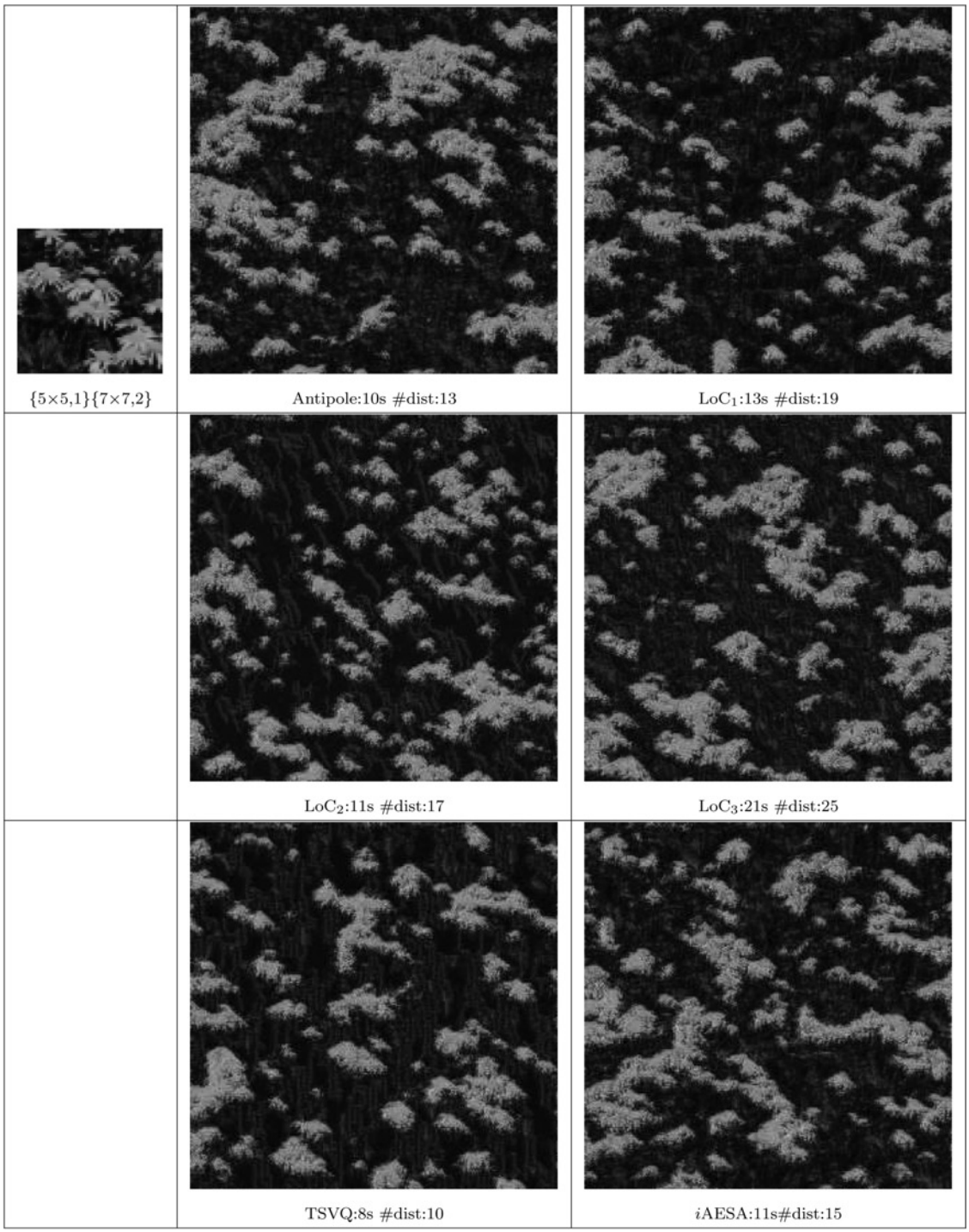


Fig. 6 *Approximate k-nearest-neighbour search comparisons between antipole, list of clusters (LoC_1, LoC_2, LoC_3), TSVQ and iAESA for 'Flowers' image*

Input image is 65 × 65, whereas output images are 256 × 256
Under the input image, there is the neighbourhood used; under each produced texture, there is the running time in seconds and the number of distances computations
See Fig. 1 for input image dimensions

- The antipole tree stores in each internal node N the distances from N to all the objects which previously were in the same subset. Hence, it is able solve efficiently the exact k-nearest-neighbour search.

3.4 AESA

The approximating and eliminating search algorithm (AESA) and AESA-based related techniques are among the fastest methods for k-nearest-neighbour search in general metric spaces. In our experiments, we have used both AESA and iAESA, an AESA improved version which makes use of a better prediction function.

3.4.1 k-nearest-neighbour search via AESA: Let M be a metric space, $P \subset M$ a set of points and $q \in M$ a query point. The AESA algorithm solves the k-nearest-neighbour search problem using an iterative procedure: it first chooses a pivot element which is compared against q. Then it tries to filter out from P as many candidates as possible until all the candidates are either compared

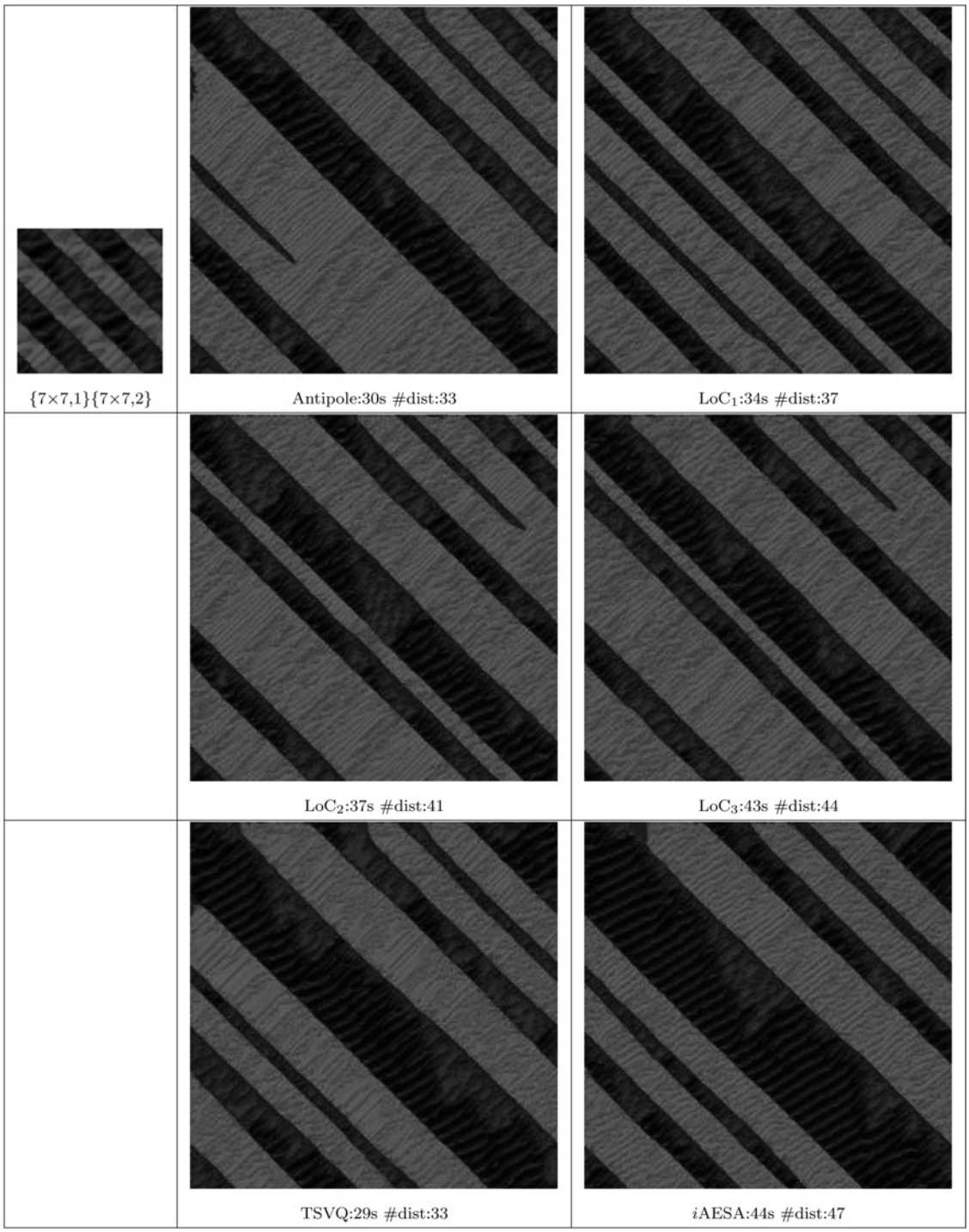


Fig. 7 *Approximate k-nearest-neighbour search comparisons between antipole, list of clusters (LoC_1, LoC_2, LoC_3), TSVQ and iAESA for 'Strips' image*

Input image is 65 × 65, whereas output images are 256 × 256
Under the input image, there is the neighbourhood used; under each produced texture, there is the running time in seconds and the number of distances computations
See Fig. 1 for input image dimensions

or discarded. Let P be the set of pivots already compared against q. The next pivot is then chosen as the candidate c for which

$$D(c) = \sum_{p \in P} |d(c, p) - d(p, q)| \quad (1)$$

is minimised. The triangular inequality is used in this process: given two candidates p, p', $d(p, q) \geq |d(p, p') - d(p', q)|$. Then the distance from any other candidate p to q can be easily bounded below by $B(p) = \max_{p' \in P}|d(p, p') - d(p', q)|$ with P being the set of previously selected candidates. The number of distance computations required by the AESA algorithm for the k-nearest-neighbour search is only a small fraction of the total number of candidates and this number is asymptotically independent of the total number of candidates.

iAESA is an improved version of AESA which makes use of a different technique to select the next pivot. Let $P \subset U$

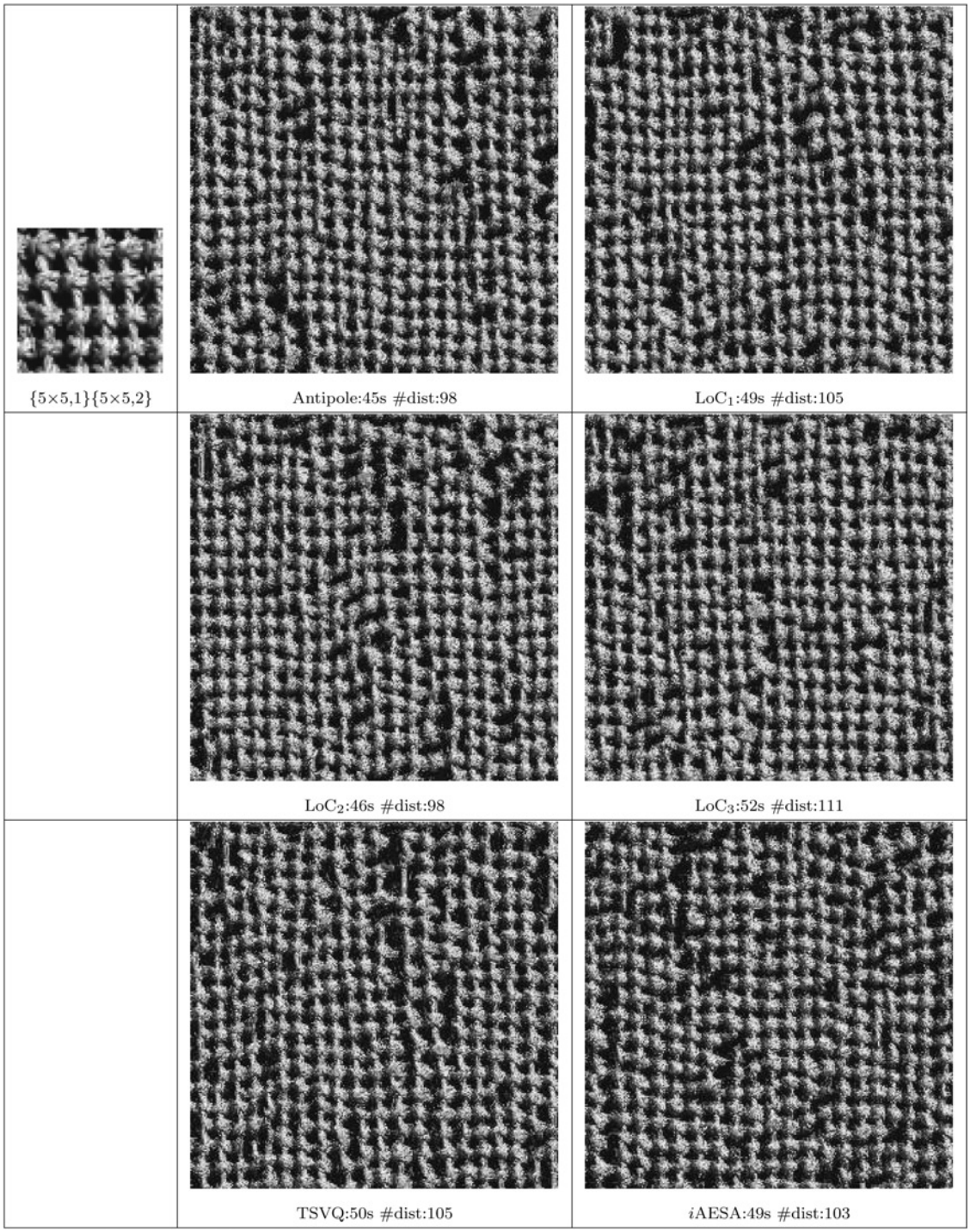


Fig. 8 *Approximate k-nearest-neighbour search comparisons between antipole, list of clusters (LoC_1, LoC_2, LoC_3), TSVQ and iAESA for 'Bends' image*

Input image is 65 × 65, whereas output images are 256 × 256
Under the input image, there is the neighbourhood used; under each produced texture, there is the running time in seconds and the number of distances computations
See Fig. 1 for input image dimensions

be the set of pivots already compared against q with the pre-order $\preceq_u$ defined as: for $y,\ z \in P$ and $u \in U,\ y \preceq_u z \Leftrightarrow d(u, y) \leq d(u, z)$. Every object u can compute its pre-order of P and associate it to a permutation. Let $\Pi_u = p_1, p_2, \ldots, p_{|P|}$ with $p_i \leq p_{i+1}$ (meaning that $d(p_i, q) \leq d(p_{i+1}, q)$), a permutation of u; let $\Pi_u^{-1}(p_i$ be the position of the element p_i in the permutation Π_u. Two equal elements will have the same permutation, whereas two similar elements will have similar permutation. Given P the set of pivots already compared against q, the next pivot is the one for which the Spearman Footrule predictive function

$$F(u) = F\left(\prod_u, \prod_q\right) = \sum_{i=1}^{|p|} \left| \prod_u^{-1}(p_i) - \prod_q^{-1}(p_i) \right| \quad (2)$$

is minimised.

The probabilistic iAESA has also been considered for the approximate k-nearest-neighbour problem. In this case, instead of checking each candidate, the algorithm would check a certain number, according to the desired precision.

Fig. 9 *Image colourisation results for exact nearest-neighbour search*

First column is the input image with its size; second column is the greyscale image; third column shows the original coloured image and its size; the fourth column shows the results

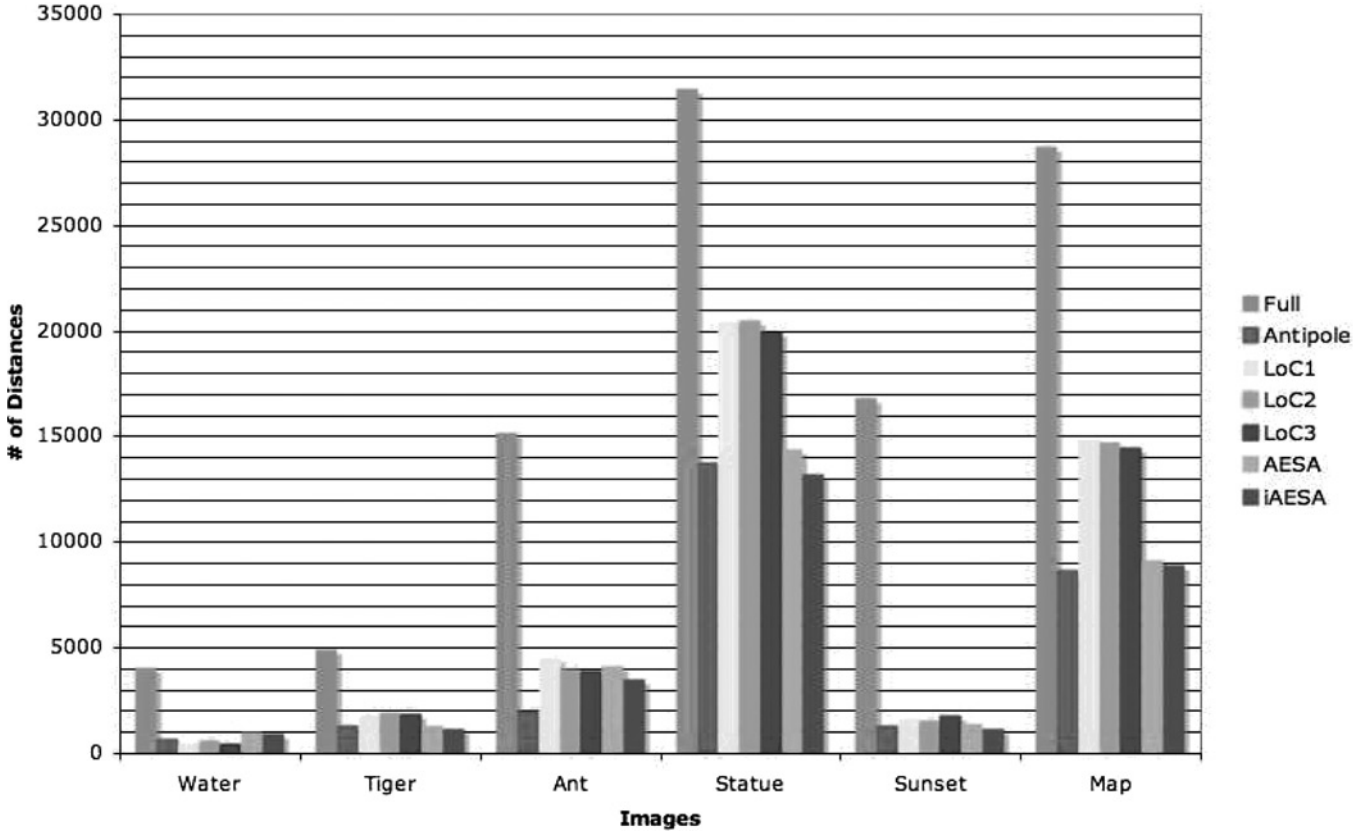


Fig. 10 *Number of computations for image colouring for images in Fig. 9*

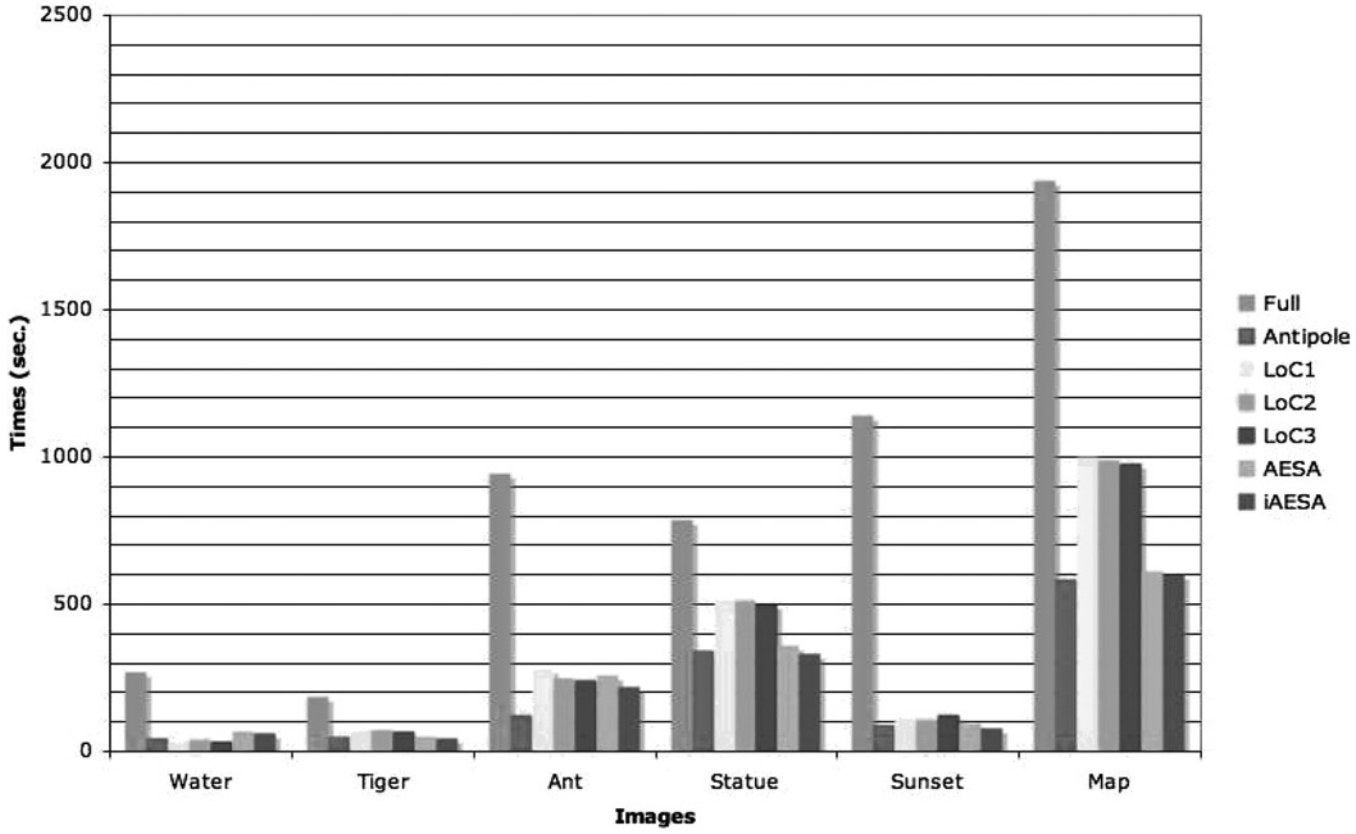


Fig. 11 *Image colouring reconstruction times for images in Fig. 9*

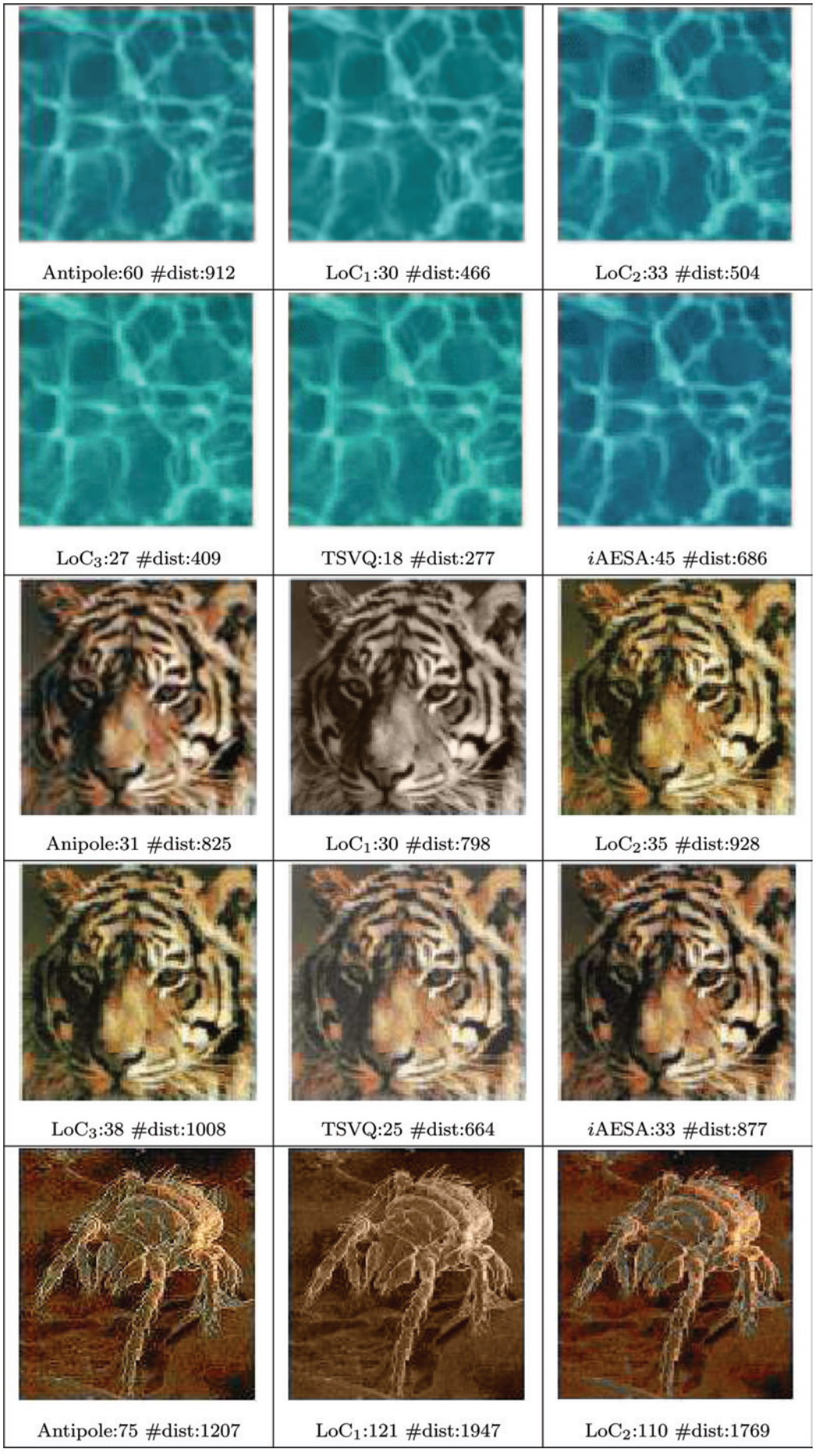


Fig. 12 *Image colourisation timing and number of distance computations for approximate nearest-neighbour search*

See Fig. 9 for input image and space dimension
Algorithms have been run by using a neighbourhood 5×5

Fig. 13 *Image colourisation timing and number of distance computations for approximate nearest-neighbour search*
See Fig. 9 for input image and space dimension
Algorithms have been run by using a neighbourhood 5 × 5

Even if the complexity of AESA is $O(|P| \cdot n)$ and iAESA is $O(|P|^2 \cdot n)$, experiments on iAESA [11] show improvements up to 75% of the number of distances (see [10, 30–33] for further details on AESA and iAESA).

4 Texture synthesis

Texture synthesis is an important issue in many applications in computer graphics. Textures are commonly

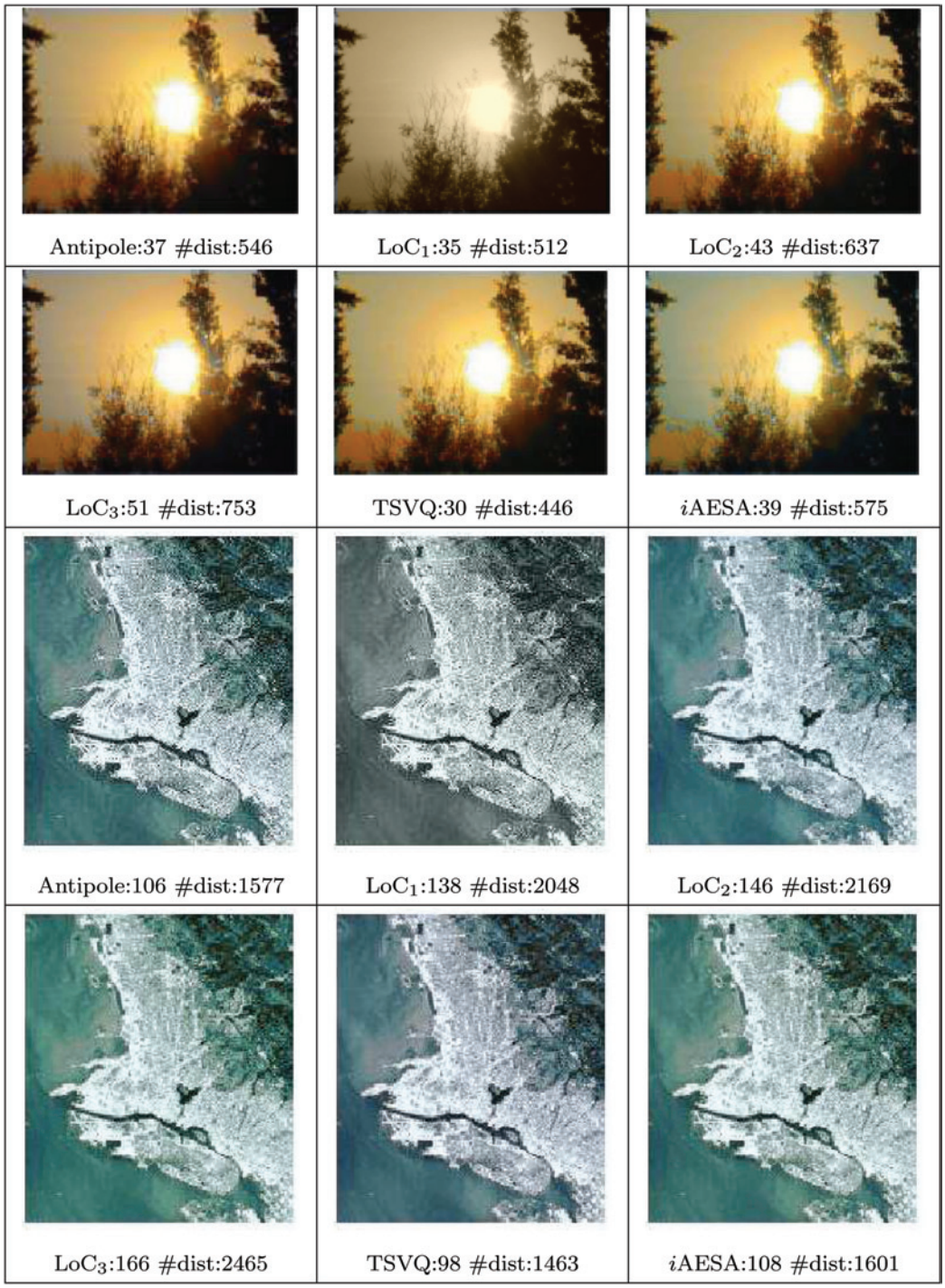


Fig. 14 *Image colourisation timing and number of distance computations for approximate nearest-neighbour search*
See Fig. 9 for input image and space dimension
Algorithms have been run by using a neighbourhood 5×5

employed when rendering synthetic images. Given a texture sample, the goal of texture synthesis is to synthesise a new texture that, when perceived by a human observer, appears to be generated by the same stochastic process. Hence, starting with a texture input, the major challenges are how to estimate the stochastic process and how to develop an efficient procedure to produce a new texture. Texture synthesis techniques can be explicit and implicit. Explicit methods (an extensive literature could be found in [34–41]) generate all the textures directly and the value of each texture pixel is related to other pixels. On the other hand, implicit methods (see [42–44]) answer a query about a given sample without computing the whole texture and they allow texels to be evaluated independently. In [45], the authors build a method with both advantages of explicit and implicit methods: their method synthesises new textures simply from given examples and it allows textures to be evaluated in any traversal order where different traversal orders always yield identical results starting from the same initial inputs. Another kind of texture synthesis methods are the ones which try to transfer texture patches from samples adjusting the borders (see [46, 47]).

All the explicit methods are computationally expensive as they have to load the input images database and then perform a nearest-neighbour search on the database. For this reason, usually, approximate searches are preferred, as in [41], with the subsequent losing of quality of results. We show

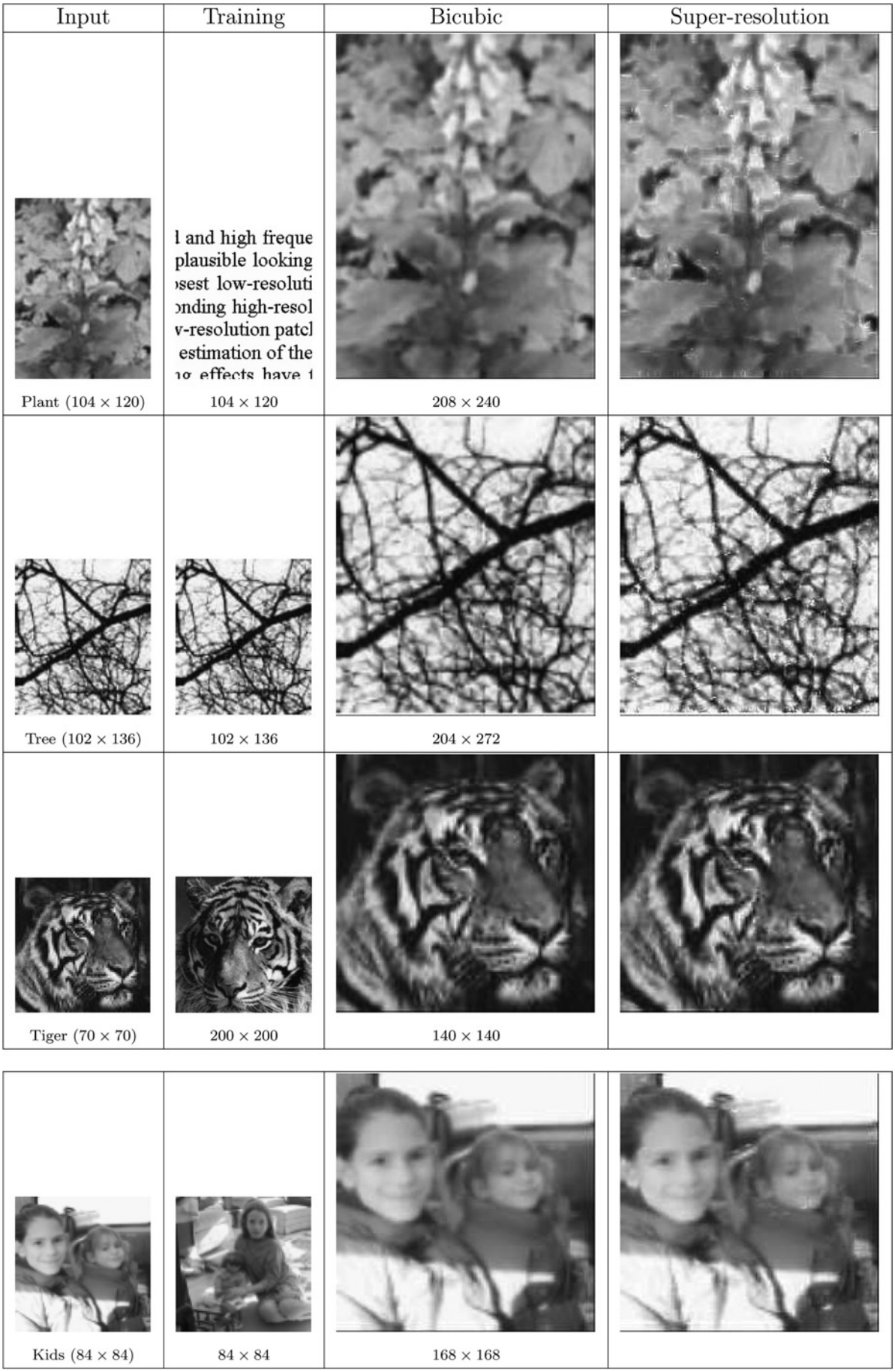


Fig. 15 *Full search results*

First two columns show, respectively, the input and the training image with their size
Third column shows the results obtained using the bicubic interpolation and the output size; fourth column shows the results obtained using the method we implemented

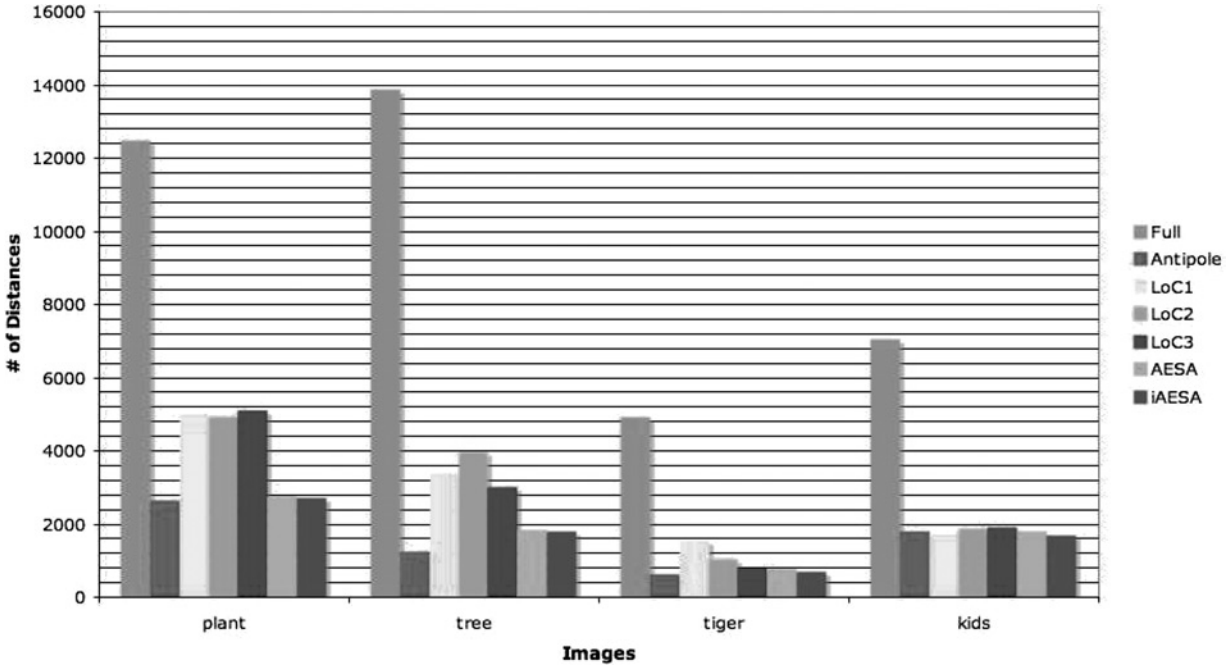


Fig. 16 *Number of distance computations for super-resolution of images in Fig. 15*

that using an efficient data structure for indexing the database, we are able to obtain the highest texture quality saving a lot of the time spent by the classic full-search technique. To give an example of the potential gain we get, we have implemented the texture synthesis technique discussed in [41] with the acceleration of different data structures above described. In this method, given an input texture patch together with a random noise having the desired size of the output image, the algorithm modifies the random noise image to make it looks like the given example. First, in raster scan order, for each pixel of the input texture, a causal neighbourhood is computed and stored in the database together with the RGB value of the current pixel. In the reconstruction process, the value of each output pixel is determined in raster scan order by comparing its spatial neighbourhood with all the neighbourhoods in the database. The input pixel with the most similar neighbourhood will be assigned to the current output pixel (see [41] for more details and [29] for the introduction of the antipole tree and list of clusters in the above procedure).

In Fig. 1, we show the time comparisons of the implemented texture synthesis technique with the exact

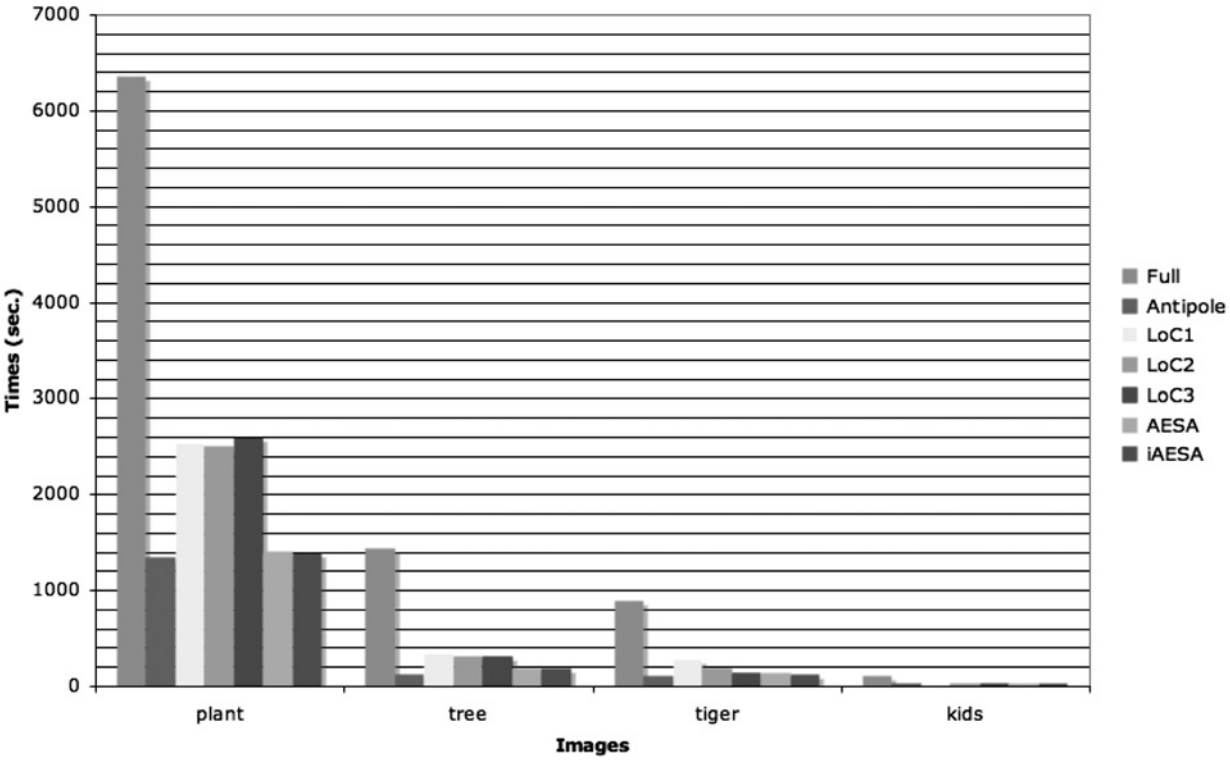


Fig. 17 *Super-resolution reconstruction times for images in Fig. 15*

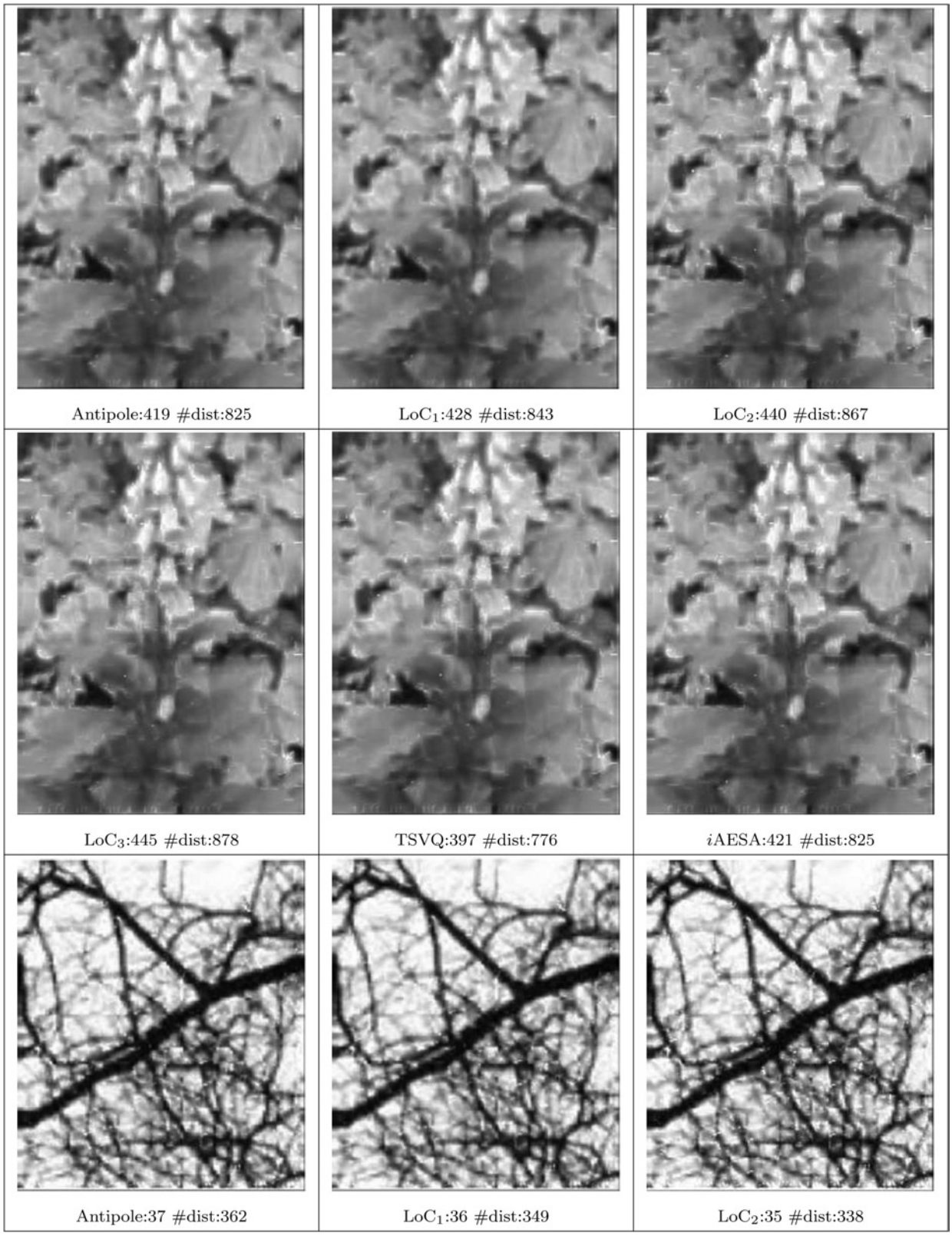


Fig. 18 *Super-resolution for the antipole, list of clusters (LoC_1, LoC_2, LoC_3), TSVQ and IAESA using approximate k-nearest-neighbour searches*

Timing in seconds and number of distance computations are displayed
Neighbourhood 7 × 7 has been used meaning that the space dimension is 186
See Fig. 15 for the data set size

nearest-neighbour search for different input textures. The first column reports the input textures. We have used input textures of 64 × 64 pixels to synthesise 256 × 256 output textures. The second column shows the size of the neighbour and the number of multiresolution levels used (see [29] for details about texture synthesis with multiresolution); $|D|$ and $|S|$ represent, respectively, the size and the space dimension of the data set for each level of multiresolution. Some synthesised textures are shown in Fig. 2. Figs. 3 and 4 report, respectively, number of distance

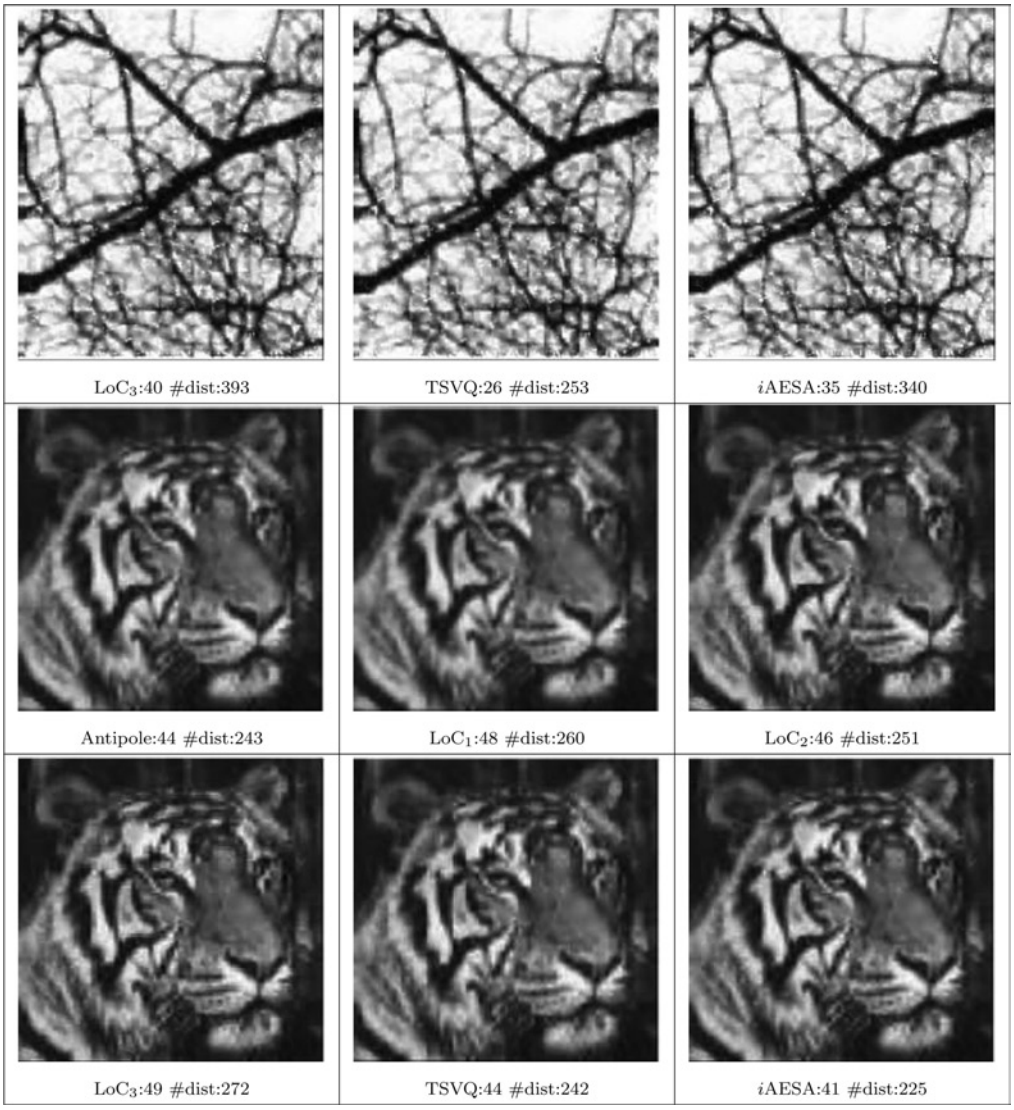


Fig. 19 *Super-resolution for the antipole, list of clusters (LoC_1, LoC_2, LoC_3), TSVQ and IAESA using approximate k-nearest-neighbour searches*

Timing in seconds and number of distance computations are displayed
Neighbourhood 7×7 has been used meaning that the space dimension is 186
See Fig. 15 for the data set size

computations and the times in seconds of the reconstruction process with the exact searches: sequential, antipole tree, list of clusters, AESA and iAESA.

The reader can observe two important considerations which hold also for the other imaging problems we have considered:

- The outputs for sequential, antipole, list of clusters and AESA full searches are always the same and are shown in Fig. 2.
- We have not included the TSVQ in Fig. 2 as it does not solve the exact *k*-nearest-neighbour search.
- Using one of the cited data structures, it is possible to speed-up enormously the texture synthesis reconstruction. The earned time is lower when the search space becomes bigger because of the curse of dimensionality. For texture synthesis, this happens when the size of the neighbourhood is high. For example, when the neighbourhood size is $\{9 \times 9,\ 1\}$ $\{9 \times 9,\ 2\}$ $\{9 \times 9,\ 3\}$ $\{9 \times 9,\ 4\}$ (see Straw, Pills, Strips), the gained performance is lower than the ones obtained with smaller neighbourhoods.

Figs. 5–8 display a comparison between the antipole, list of clusters, TSVQ and iAESA approximate searches (see [4] for a well-described performance comparison in function of the computed distances of approximate nearest-neighbour searches between the antipole and list of clusters). All the parameter settings (choice for radius, bucket size, number of iterations and so on) affect the results for all the data structures. For each of them, it is possible to find a reasonable trade-off between the quality and the performance. When the space dimension gets too high, the obtained gain is small. Note that for all the experiments, the required times for building the data structures have been included. We have noticed that the antipole tree produces reasonable output images keeping the number of distance computations low. We have not focused on a full comparison of all the cited data structures because that would go beyond the purpose of this paper. Here we want to emphasise that the introduction of indexing schemas speeds up numerous imaging problems and this does not depend on the particular property of the space.

Fig. 20 *Super-resolution for the antipole, list of clusters (LoC_1, LoC_2, LoC_3), TSVQ and IAESA using approximate k-nearest-neighbour searches*

Timing in seconds and number of distance computations are displayed
Neighbourhood 7 × 7 has been used meaning that the space dimension is 186
See Fig. 15 for the data set size

5 Image colourisation

Greyscale images such as old black-and-white photos, classic movies or scientific illustrations can be coloured in order to increase their visual appeal. The problem of colourisation consists in finding colour information for an image whose pixels are characterised only by the luminance. Different pixels may carry the same luminance but different hue and/or saturation values; the problem of colourising greyscaled images hence has not correct solution and human interaction plays a large role in the colourisation process. There are different techniques for image colourisation (see [48–52]), most of which use training set to learn how to colourise greyscaled images. The method we implemented is the one introduced by Welsh *et al.* [53]. In their work, colours are transferred from a source coloured image to a target greyscale image using a similar technique as the one seen for texture synthesis in [41]. First of all, both colour source and target greyscale images are converted to YUV colour space. This colour space provides three decorrelated channels, Y for the luminance which is a crucial datum for the procedure, U and V for the yellow–blue and red–green channels. Then, for each pixel of the source image, in raster scan order, we build a vector consisting of the luminance neighbourhood of the given pixel. We store in our data structures all the obtained vectors together with the UV components of the given pixel. Then we pass to the target greyscale image. In raster scan order, we build the luminance vector in the same way as we did for the source image. Then we search in the database the nearest luminance vector according to the L_2 distance metric and transfer its UV components to the target image. Details of the algorithm can be found in [53] and details of how an indexing schema is used in the whole procedure are in [54]. With such a technique, it is also possible to colourise video by transferring colour from a source colour image to a target frame. If the frame is successfully colourised, then all the similar frames consisting of the same objects will be colourised similarly.

Fig. 9 shows some graphical results for exact search obtained using a neighbourhood 5 × 5. Figs. 10 and 11 show the corresponding number of distance computation and seconds employed for the colourisation process, respectively.

As seen for the texture synthesis, it is possible to speed-up more the colourisation process using the approximate searches of the cited data structures. Figs. 12–14 show graphical and timing results for the images shown in Fig. 9 using approximated searches. As there is no multiresolution, the space dimension is generally lower than the one built for texture synthesis; this results in a better performance for each of the cited schemas with respect to the classical sequential search. For both Fig. 12 and 14, the preprocessing times for building the data structures have been included. Similar considerations made at the end of Section 4 apply here.

6 Super-resolution

Zooming is very important when dealing with images. Super-resolution is a technique that, in some way, enhances the resolution of an image. If we want to interpret an image, we are limited to the resolution of the image. If we zoom beyond its resolution, then the necessary interpolation will create a blurred image. Super-resolution techniques guess the missing data to make the zoomed image looks sharper. There are different methods which try to zoom an image. The cubic spline [55] is a common image interpolation function but it does not work properly in guessing image details. Other methods [1, 30–62] start with the cubic-spline interpolation and try to guess the missing details. Another method [63] uses as a reference the sharp

and blurred versions of the same image to create by analogy the sharp version of a given input image. We implemented the one-pass algorithm of Freeman introduced in [62] and discussed in [1]. The idea is learning how to sharpen a given input image from a training set of sharp images containing low-, mid- and high-frequency data. For each image of the training set, we consider its low- and high-frequency patches. Then, each $M \times M$ low-frequency patch is linked to the overlapping of borders of correspondent $N \times N$ high-frequency patch for the spatial effects constraint. A parameter, α is used to adjust the importance of matching the low-frequency patches against matching the neighbourhood high-frequency patches in the overlap pixels. As the algorithm works under the hypothesis that the predictive relationship between low- and high-resolution images is independent of local image contrast, patch pairs are normalised by the average absolute value of low-frequency patch across the RGB channels. The produced vectors are stored into a database together with the $N \times N$ high-frequency patches. Once the database is ready, the input image is scaled up using a cubic-spline interpolation. The process of building pixel vectors is similar to the one seen for training set generation; the high-frequency patch associated to the nearest vector to the current one according to the L_2 norm is added to the initial interpolation to obtain the output image (see [1, 62] for major details). In [1, 62], because of the high dimension of search space, the authors use approximated approaches for nearest-neighbour search. Given the efficiency of the discussed data structures, we used them to index the space and for nearest-neighbour searching. Fig. 15 shows the graphical results for exact k-nearest-neighbour searches. Note that the outputs are always the same. Figs. 16 and 17 report, respectively, the number of distance computation and timing in seconds, respectively. Similar to image colouring, as there is no multiresolution, the obtained metric space is lower than the one built for texture synthesis. This allows the indexing schemas to perform better in terms of both number of distance computations and times. The searches of all the mentioned schemas outperform the classical sequential search. Figs. 18–20 displays the outputs for the approximate searches for images in Fig. 15. Here is clearly visible how the approximate searches affects the quality of the outputs but it allows to save always more than 50% of the time. The times required for building the data structures have been included for both exact and approximate results. Same considerations made at the end of Section 4 hold here too.

7 Conclusions

In this paper, we have shown how the introduction of efficient data structures helps to solve efficiently different image problems (texture synthesis, image colourisation and super-resolution). We have considered the antipole tree, list of clusters, AESA, iAESA and TSVQ schemas because they are recent data structures which efficiently perform exact and approximate searches in metric spaces. All of them speed up the image reconstruction process. When the space dimension increases, the gain decreases for the curse of dimensionality. Even if the approximation search gives satisfactory results in almost all cases, it should be better to introduce *ad hoc* metrics devoted to better quantify the visual distortion for imaging applications. For all the experiments that we have conducted, we used a Mobile Pentium 4 with 2.30 GHz and 512 MB of DDR SDRAM equipped with Debian Linux 3.1. The antipole, list of clusters, AESA and iAESA have been implemented in ANSI C and C++.

8 Acknowledgment

We are grateful to the anonymous reviewers for useful suggestions and comments.

9 References

1 Régis Destobbeleire: 'Super-resolution', Technical report, 2002, **24**
2 Bozkaya, T., and Ozsoyoglu, M.: 'Indexing large metric spaces for similarity search queries', *ACM Trans. Database Syst.*, 1999, **24**, (3), pp. 361–404
3 Ciaccia, P., Patella, M., and Zezula, P.: 'M-tree: an efficient access method for similarity search in metric spaces'. Proc. 23th Int. Conf. on Very Large Databases, 1997, pp. 426–435
4 Cantone, D., Ferro, A., Pulvirenti, A., Reforgiato, D., and Shasha, D.: 'Antipole tree indexing to support range search and k-nearest-neighbor search in metric spaces', *IEEE Trans. Knowl. Data Eng.*, 2005, **17**, (4), pp. 535–550
5 Traina, C. Jr, Traina, A., Seeger, D., and Faloutsos, C.: 'Slim-trees: high performace metric trees minimizing overlap between nodes'. Proc. 7th Int. Conf. Extending Database Technology, 2000, (*Lect. Notes Comput. Sci.,* **1777**), pp. 51–56
6 Baeza-Yates, R., Cunto, W., Manber, U., and Wu, S.: 'Proximity matching using fixed-queries trees'. Proc. Combinatorial Pattern Matching, 5th Annual Symp., 1994, (*Lect. Notes Comput. Sci.,* **807**), pp. 198–212
7 Chávez, E., and Navarro, G.: 'An effective clustering algorithm to index high dimensional metric spaces'. Proc. SPIRE, 2000, pp. 75–86
8 Navarro, G.: 'Searching in metric spaces by spatial approximation', *VLDB J.*, 2002, **11**, pp. 28–46
9 Gersho, A., and Gray, R.: 'Vector quantization and signal compression' (Kluwer Academic Publishers, 1992)
10 Vidal, E.: 'An algorithm for finding nearest neighbours in (approximately) constant average time', *Pattern Recognit. Lett.*, 1986, **4**, (3), pp. 145–157
11 Figueroa, K., Chavez, E., Navarro, G., and Paredes, R.: 'On the least cost for proximity searching in metric spaces'. WEA 2006, 2006, pp. 279–290
12 Chávez, E., Navarro, G., BaezaYates, R., and Marroquin, J.L.: 'Searching in metric spaces', *ACM Comput. Surv.*, 2001, **33**, (3), pp. 273–321
13 Aggarwal, C., Wolf, J.L., Yu, P.S., and Epelman, M.: 'Using unbalanced trees for indexing multidimensional objects', *Knowl. Inf. Syst.*, 1999, **1**, (3), pp. 309–336
14 Gionis, A., Indyk, P., and Motwani, R.: 'Similarity search in high dimensions via hashing'. Proc 25th VLDB Conf., 1999, pp. 518–529
15 Beyer, K., Goldstein, J., Ramakrishnan, R., and Shaft, U.: 'When is nearest neighbor meaningful?'. Proc. 7th Int. Conf. Database Theory, 1999, **1540**, pp. 217–235
16 Berchtold, S., Keim, D.A., and Kriegel, H.P.: 'The x-tree: an index structure for high-dimensional data'. Proc. 22th Int. Conf. Very Large Database, 1996, pp. 28–39
17 Sumanasekara, S., and Ramakrishna, M.V.: 'Chilma: an efficient high dimensional indexing structure for image database'. Proc. First IEEE Pacific-Rim Conf. Multimedia, 2000, pp. 76–79
18 Di Blasi, G., Gallo, G., and Petralia, M.: 'Puzzle image mosaic'. Proc. IASTED/VIIP2005, 2005
19 Di Blasi, G., Gallo, G., and Petralia, M.: 'Smart ideas for photomosaic rendering'. Proc. Eurographics Italian Chapter 2006, 2006, pp. 267–271
20 Indyk, P., and Motwani, R.: 'Approximate nearest neighbors: towards removing the curse of dimensionality'. Proc. 30th Annual ACM Symp. Theory of Computing, 1998, pp. 604–613
21 Chazelle, B.: 'Computational geometry: a retrospective'. Proc. 26th Annual ACM Symp. Theory of Computing, 1994, pp. 75–94
22 Chávez, E., and Navarro, G.: 'A probabilistic spell for the curse of dimensionality'. Proc. 3rd Workshop on Algorithm Engineering and Experimentation (ALENENEX'01), 2001, (*Lect. Notes Comput. Sci.*, **2153**), pp. 147–160
23 Ciaccia, P., and Patella, M.: 'Pac nearest neighbor queries: approximate and controlled search in high-dimensional and metric spaces'. ICDE, 2000, pp. 244–255
24 Mitchell, T.M.: 'Machine learning' (McGraw-Hill, 1997)
25 Hjaltason, G.R., and Samet, H.: 'Distance browsing in spatial database', *ACM Trans. Inf. Syst.*, 1999, **24**, (2), pp. 265–318
26 Chávez, E., and Navarro, G.: 'A compact space decomposition for effective metric indexing', *Pattern Recognit. Lett.*, 2005, **26**, (9), pp. 1363–1376
27 Bustos, B., and Navarro, G.: 'Probabilistic proximity searching algorithms based on compact partitions'. Proc. SPIRE, 2002, pp. 284–297

28 University of Washington Data Compression Laboratory: 'TSVQ'. http://dcl.ee.washington.edu/
29 Battiato, S., Pulvirenti, A., and Reforgiato, D.: 'Antipole clustering for fast texture synthesis'. Proc. Winter School of Computer Graphics (WSCG), 2003
30 Vidal, E.: 'New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA)', *Pattern Recognit. Lett.*, 1994, **15**, (1), pp. 1–7
31 Moreno-Seco, F., Oncina, J., and Mico', L.: 'Improving the LAESA algorithm error rates'. Proc. VIII Symp. NdRdFyAdI, 1999, vol. 1, pp. 413–419
32 Moreno-Seco, F., Oncina, J., and Mico', L.: 'Improving the linear approximating and eliminating search algorithm (LAESA) error rates', *Pattern Recognit. Appl.*, 2000, **56**, pp. 37–43
33 Juan, A., and Vidal, E.: 'An optimized version of the approximating and eliminating search algorithm (AESA) for nearest neighbour classification', Technical report ITI-ITE-3/98, 1998
34 De Bonet, J.S.: 'Multiresolution sampling procedure for analysis and synthesis of texture images'. Proc. ACM SIGGRAPH, 1997, pp. 361–368
35 Efros, A., and Leung, T.: 'Texture synthesis by a non-parametric sampling'. Proc. IEEE Int. Conf. Comput. Vis., 1999, vol. 2, pp. 1033–1038
36 Zhu, S.C., Wu, Y.N., and Mumford, D.: 'Filters, random fields, and maximum entropy (frame) – towards a unified theory for texture modeling', *Int. J. Comput. Vis.*, 1998, **27**, (2), pp. 107–126
37 Popat, K., and Picard, R.: 'Novel cluster-based probability model for texture synthesis, classification, and compression', *Proc. SPIE*, 1993, **2094**, pp. 756–768
38 Heeger, D.J., and Bergen, J.R.: 'Pyramid-based texture analysis/synthesis'. Computer Graphics, ACM SIGGRAPH, 1995, pp. 229–238
39 Simoncelli, E., and Portilla, J.: 'Texture characterization via joint statistics of wavelet coefficient magnitudes'. Fifth Int. Conf. Image Processing, 1998, vol. 1, pp. 62–66
40 Hertzmann, A., Jacobs, C.E., Oliver, N., Curless, B., and Salesin, D.H.: 'Image analogies'. Proc. ACM SIGGRAPH 2001, 2001
41 Wei, L.Y., and Levoy, M.: 'Fast texture synthesis using tree-structured vector quantization'. Proc. ACM-SIGGRAPH, 2000, pp. 479–488
42 Ebert, D.S., Musgrave, F.K., Peachey, D., Perlin, K., and Worley, S.: 'Texturing and modeling: a procedural approach' (Morgan Kaufmann Publishers, 1998)
43 Peachey, D.: 'Solid texturing of complex surfaces'. Proc. SIGGRAPH' 85, 1985, pp. 279–286
44 Perlin, K.: 'An image synthesizer'. Proc. SIGGRAPH' 85, 1985, pp. 287–296
45 Wei, L.Y., and Levoy, M.: 'Order-independent texture synthesis', Technical report, 2002
46 Ashikhmin, M.: 'Synthesizing natural textures'. ACM Symp. Interactive 3D Graphics, 2001, pp. 217–226
47 Efros, A.A., and Freeman, W.T.: 'Image quilting for texture synthesis and transfer'. Proc. SIGGRAPH 2001, 2001
48 Gonzalez, R.C., and Wintz, P.: 'Digital image processing' (Addison-Wesley, Reading, MA, 1987)
49 Silberg J.: http://www.cinesite.com/core/press/articles/1998/10_00_98- team.html, 1998
50 Reinhard, E., Ashikhmin, M., Gooch, B., and Shirley, P.: 'Color transfer between images', *IEEE Comput. Graph. Appl.*, 2001, **21**, pp. 34–40
51 Rogowitz, B.E., and Kalvin, A.D.: 'The "Which Blair Project": a quick visual method for evaluating perceptual color maps'. Proc. IEEE Visualization 2001, 2001
52 Pratt, W.K.: 'Digital image processing' (Wiley, 1991)
53 Welsh, T., Ashikmin, M., and Mueller, K.: 'Transferring color to greyscale images'. Proc. ACM SIGGRAPH 2002, 2002
54 Di Blasi, G., and Reforgiato Recupero, D.: 'Fast colorization of gray images'. Eurographics Italian Chapter, 2004
55 Keys, R.: 'Cubic convolution interpolation for digital image processing', *IEEE Trans. Acoust. Speech Signal Process.*, 1981, **29**, (6), pp. 1153–1160
56 Fekri, F., Mersereau, R.M., and Schafer, R.W.: 'A generalized interpolative vq method for jointly optimal quantization and interpolation of images'. Proc. Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP), 1998, vol. 5, pp. 2657–2660
57 Schreiber, W.F.: 'Fundamentals of eletronic imaging systems' (Springer, New York, 1986)
58 Thurnhofer, S., and Mitra, S.: 'Edge-enhanced image zooming', *Opt. Eng.*, 1996, **35**, (7), pp. 1862–1870
59 Schultz, R.R., and Stevenson, R.L.: 'A bayesian approach to image expansion for improved definition', *IEEE Trans. Image Process.*, 1994, **3**, (3), pp. 233–242
60 Freeman, W.T., and Pasztor, E.C.: 'Learning to estimate scenes from images'. Adv. Neural Information Process. Syst., 11, 1999, pp. 775–781
61 Freeman, W.T., Pasztor, E.C., and Carmichael, O.T.: 'Learning low-level vision', *Int. J. Comput. Vis.*, 2000, **40**, (1), pp. 25–47
62 Freeman, W.T., Jones, T.R., and Pasztor, E.C.: 'Example-based superresolution', MERL TR-2001-340, 2001
63 Hertzmann, A., Jacobs, C.E., Olivier, N., Curless, B., and Salesin, D.H.: 'Image analogies'. ACM SIGGRAPH, 2001