PAPER *Special Issue on Machine Vision Applications*

Randomized Adaptive Algorithms for Mosaicing Systems*

Frank NIELSEN†, *Nonmember*

SUMMARY Given a set of still images taken from a hand-held camera, we present a fast method for *mosaicing* them into a single blended picture. We design time- and memory- efficient still image mosaicing algorithms based on geometric point feature matchings that can handle both arbitrary rotations and large zoom factors. We discuss extensions of the methodology to related problems like the recovering of the epipolar geometry for 3d reconstruction and object recognition tasks.

key words: image processing, registration, warping , mosaicing, point pattern matching, bucketting

1. Introduction

Mosaicing (also called *image stitching*) consists of taking a sequence of still image pictures and providing a collection of transformations to join/merge them into one blended picture (see Figure 1 and 9). Many software companies already propose stitchers for creating panoramas and browsers for navigating through them via environment maps (image-based rendering systems). Basically, those programs proceed as follows: (1) find or ask for camera parameters, (2) warp images according to these parameters (lines bend to quadratic curves) and (3) stitch images by means of a 1d- or 2d-translation and eventually small tilting, and (4) adjust and blend color intensities. Panoramic images do not preserve lines. We refer the reader to the course note [2] for an up-to-date survey on image registration and image warping techniques.

In this paper, we consider *perspective mosaicing* where straightness of lines is preserved. Our method is *automatic* and does not assume any user input nor any *a priori* knowledge of the camera parameters. Images can also be taken by several pinhole cameras having different intrinsic/extrinsic parameters. In the case of images taken by an ideal pinhole-model camera from (a) the same three-dimensional viewpoint or (b) a planar surface taken from two different viewpoints, the transformations related these images are known to be homographies (also called collineations [3]), linear transformations defined up to a scalar factor, in the projective plane $\mathfrak{P}^2$. A key difference from panoramic mosaicing is that we can interpret the composite stitching as the image that would have been taken by a bigger sensing device (e.g., CCD).

There are two widely used techniques that have been used so far in the past for mosaicing: the first one consists of *image registration* where the collineation is often restricted to similitudes on the plane (*id.* translation, rotation and scaling). Local image registration is usually either performed by gradient descent or Levenberg-Marquadt optimization procedure. Global image registration is reached (sometimes) by hierarchical matching using pyramidal image representation. The second method is based on Fourier principles and is called the *phase correlation* method. This technique estimates a global 2d translation by computing the phase differences of the respective image frequency signals but loses the perspective information as it goes to the frequency domain. Moreover for large rotations or different zoom scales, this method fails. Szeliski and Shum presented an elegant and robust method for creating full mosaics and texturing them onto a polyhedron [4], given rough image matchings.

Our proposed fully automatic method handles large zoom ($\cong \times 3$) and arbitrary rotation values while keeping the running time attractive for responsive applications. Its current limitations, as discussed in section 6, are mostly due to the detection of *reliable* features (sometimes called "repeatable" or "stable" features) in images having large different scalings rather than the matching process in itself. Indeed, our method is based on planar point set pattern matching having a given precision tolerance. As the zoom range grows, say linearly, the tolerance window needs more than linear growth because of the detected feature imprecisions. Our method is based on Monte-Carlo/Las Vegas algorithms that combine both randomization and geometric feature selection.

Mosaicing is a core technology used in composite scanning, image-based rendering (e.g., new view generation without having fine 3d model description), high-definition pictures, image compression, image stabilization, etc. Still image mosaicing techniques largely differ from video mosaicing techniques [5] because no video

Manuscript received 1st October 1999
Manuscript revised 11th January 2000

correspondences efficiently.

2. Feature-based mosaicing

Let $\mathbf{I}_1$ and $\mathbf{I}_2$ be two pictures taken from the same optical center. We first start by detecting geometric feature (points, edges, triple junctions, etc.) sets $\mathcal{S}_1 = \{L_1, ..., L_{n_1}\}$ ($n_1 = |\mathcal{S}_1|$) and $\mathcal{S}_2 = \{R_1, ..., R_{n_2}\}$ ($n_2 = |\mathcal{S}_2|$). For example, points are extracted by a Harris-Stephens corner detector or even more reliably (at the expense of a small increase of the running time) by a Kanade-Lucas-Tomasi's intensity gradient approach. A collineation or homography is defined by the pairing $L_i \leftrightarrow R_i$ of 4 points $\{L_i\}_i$ of $\mathcal{S}_1$ with 4 other points $\{R_i\}_i$ of $\mathcal{S}_2$ in general position. Let $\mathbf{H}$ be such a transformation. Let L_i and Q_i be the pixel points in image $\mathbf{I}_1$ and $\mathbf{I}_2$ that are the respective perspective projection of the same physical three-dimensional points M_i. Using the homogeneous 2d coordinates for L_i and R_i, we get:

$$R_i = \begin{pmatrix} x'_i \\ y'_i \\ z'_i \end{pmatrix} = \mathbf{H}L_i = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$$

with $L_i = (\frac{x_i}{z_i} \frac{y_i}{z_i})$ and $R_i = (\frac{x'_i}{z'_i} \frac{y'_i}{z'_i})$. Let $\mathbf{P} = (L_1^T L_2^T L_3^T L_4^T)$ and $\mathbf{Q} = (R_1^T R_2^T R_3^T R_4^T)$ be 3x4 matrices. Then, we get $\mathbf{H} = \mathbf{Q}\mathbf{P}^T(\mathbf{P}\mathbf{P}^T)^{-1}$.

correspondences, we have as many as $4!\binom{n_1}{4}\binom{n_2}{4}$ possible transformations (Example: for $n_1 = n_2 = 40$ we have more than 200 billion induced homographies. In comparison, we have only about 1.2 millions panoramic transformations). The naive algorithm which consists in scoring one-by-one each homography and reporting one which has the best score is therefore not scaleable ($O(n^8)$ homographies, where $n = \max\{n_1, n_2\}$).

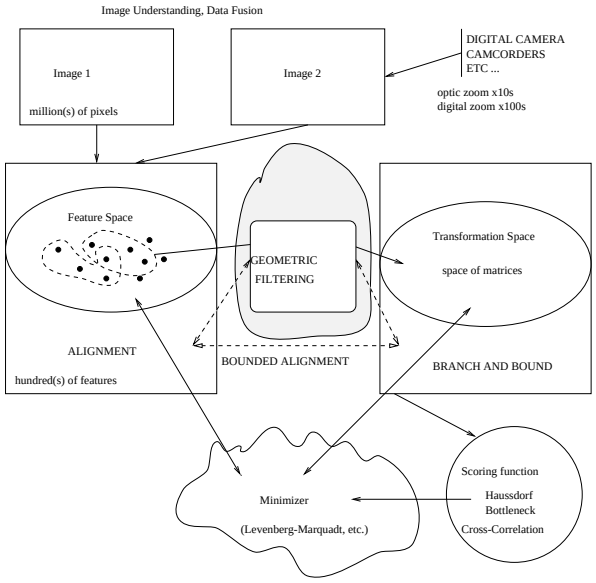


Fig. 2 Registration process synopsis: Geometric filtering selects potential feature candidates (feature space) that generate plausible transformations, i.e. reasonable zoom values, not too large rotations, etc. (transformation space)

If we consider anchored corners (a corner point and two half-line segments emanating from it) as features [6], the homography can be found by pairing only two anchored corners. Although the running time increases for detecting these elaborated templated features by non-linear minimization techniques, the core bottleneck is still the $O(n^4)$ combinatorics of the *all-pairing approach*. In this paper, we present an efficient scheme to select only a few (homographies) of them that are plausible to give birth to potential solutions.

The outline of the proposed algorithm is as follows (see also Figure 2):

- Extract features from images (e.g. corners).
- Give a set $\mathcal{T}$ of homographies satisfying *geometric constraints* and matching at least a given fraction of the point sets.
- Score each homography of $\mathcal{T}$ and choose one with highest quality (eg., score).

- Perform local optimization (subpixel analysis) on features. This step is required because we deal with pixels and not points! We can use also a gradient descent or Levenberg-Marquadt method on pixel intensities.
- Perform local optimization by perturbating coefficients of the homography matrix (compensate for lens aberration).
- Warp images (deghosting techniques and pixel amplification).
- Correct intensity and blend images.

3. Scoring transformations

Once a transformation $\mathbf{H}$ has been chosen as a potential candidate ($\mathbf{H}$ is induced by 4 correspondence pairs), we have to evaluate its score. We initially attach to each detected feature a vector of characteristics describing the neighborhood where the feature has been extracted (e.g., intensity values, qualitative measures). We say that a point L_1 in $\mathcal{S}_1$ matches a point R_1 in $\mathcal{S}_2$ for a collineation $\mathbf{H}$ if $d_2(R_1, \mathbf{H}L_1) \leqq \epsilon$ (for some prescribed $\epsilon \geqq 0$) and if their corresponding qualitative features correlate satisfactorily. We call this match an ϵ-match. Let $\mathbf{H}\mathcal{S}_1$ be the set of points $\{\mathbf{H}L | L \in \mathcal{S}_1\}$. The common feature points are called *inliers* and features present in only one of the two images are called *outliers*.

We distinguish between four families of scorings that exhibit trade-offs between running time and reliability of their scores:

(1) Pixel Cross-Corellation. We use as a scoring function for $\mathbf{H}$ the zero mean cross-correlation on subwindows centered at extracted points. On RGB pictures, a pixel p with color triple (p_R, p_G, p_B) has intensity $I(p) = 0.3p_R + 0.59p_G + 0.11p_B$. The quality is defined as follows:

$$Q(\mathbf{H}) = \frac{\sum_{p \in \mathcal{W}} (I(\mathbf{H}p) - I(p))^2}{\sqrt{\sum_{p \in \mathcal{W}} I(\mathbf{H}p)} \sqrt{\sum_{p \in \mathcal{W}} I(p)}},$$

where $\mathcal{W}$ is the correlation window.

(2) The Haussdorf matching. Each point $R \in \mathcal{S}_2$ is associated to its closest neighbor of $\mathbf{H}\mathcal{S}_1$ provided that its distance is less than a prescribed ϵ (see Figure 3). Eventually *several points* of $\mathcal{S}_2$ may be attached to the same point of $\mathbf{H}\mathcal{S}_1$ (especially when zoom factors differ). Each associated pair of points defines an edge of the graph whose nodes are the vertex set $\mathbf{H}\mathcal{S}_1 \cup \mathcal{S}_2$. (See Figure 3, top).

distance that takes into account inliers/outliers.) This measure, however, is not suitable for different image scalings.

(3) The bottleneck matching. The bottleneck measure [7], [8] reflects in a better way the $1 \leftrightarrow 1$ matching of point sets. We consider the complete bipartite graph $\mathcal{G} = (\mathbf{H}\mathcal{S}_1, \mathcal{S}_2, \mathcal{E})$, where $\mathcal{E}$ is the set of all weighted edges $e = (L, R)$ where $w(e) = d_2(L, R)$. Let $\mathcal{G}_\epsilon$ be the restricted bipartite graph of $\mathcal{G}$ containing all edges of weights less than ϵ: $\mathcal{G}_\epsilon = (\mathbf{H}\mathcal{S}_1, \mathcal{S}_2, \mathcal{E}_\epsilon)$, st. $\mathcal{E}_\epsilon = \{(\mathbf{H}L_i, R_j) | L_i \in \mathcal{S}_1, R_j \in \mathcal{S}_2, d_2(\mathbf{H}L_i, R_j) \leqq \epsilon\}$. The bottleneck matching is a maximum matching†, often not perfect matching, of $\mathcal{G}_\epsilon$ (see Figure 3, bottom).

Fig. 3 Haussdorf matching (top). Bottlebeck matching (down).

(4) Discrete approximate matching. Since our point sets are dense and of bounded diameter (namely the image diameter), we can use bucketting techniques. For each point $L \in \mathbf{H}\mathcal{S}_1$ we check whether there is a point $R \in \mathcal{S}_2$ in its neighborhood (see Figure 4).

We report the number of matching points approximately. (Buckets introduce an inherent $\sqrt{2}$-approximation factor). Note that the motif of the boolean bucket can be computed beforehand and does not depend on $\mathbf{H}$ but on $\mathcal{S}_2$.

Running times.

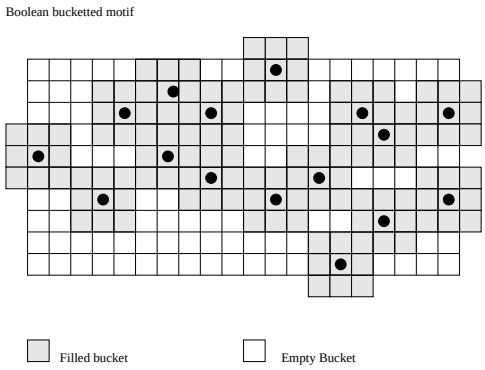


Fig. 4 Boolean bucket on set $\mathcal{S}_2$: grey buckets are marked to contain a point of $\mathcal{S}_2$ in their neighborhood.

to the number of features). (2) can be computed efficiently in $O(n \log n)$-time using Voronoi diagrams but is not appropriate for different scalings. (We may use the hardware graphics pipeline as suggested in [9] to accelerate this computation.) (3) requires more processing time; Efrat and Itai [8] using an implicit form of the geometric graph reported nearly $O(n^{\frac{3}{2}})$-time algorithm for computing a longuest matching that minimizes the maximum edge length among all matched edges. More precisely, let $m = |\mathcal{E}_\epsilon|$ be the number of edges of $\mathcal{G}_\epsilon$ and $n = n_1 + n_2$ be the number of vertices. Perfect/maximum matchings in general weighted graph, where one wants to minimize the sum of matched edges, require $O(n^3)$ time [10] using the so-called Hungarian method. On the other hand, on unweighted bipartite graphs, a maximum matching can be found whenever it exists in time $O(m\sqrt{n})$ [11]. When considering geometric graphs, i.e., graphs obtained from a geometric scene, Vaidya [12] gave an $O(n^{\frac{5}{2}})$-time algorithm for matching two points sets so that the sum of the matched edges is minimized. Considering the L_∞ distance instead of the L_2 distance, Vaidya obtained an $O(n^2 \log^3 n)$-time algorithm. Later on, those results where improved by Agarwal et al. [13] to $O(n^{2+\gamma})$ for any arbitrary small positive $\gamma > 0$. Very recently, Efrat and Itai [14] using an implicit form of the geometric graph reported nearly $O(n^{\frac{3}{2}})$-time algorithm for computing a longuest matching that minimizes the maximum edge length among all matched edges. Further refinements of their algorithm has been achieved by using dynamic data-structures for fat objects [15]. (See also the work of Heffernan and Schirra [16] for approximation schemes.)

(4) is evaluated in linear time but only gives an approximation of the size of the common point set.

Another classic approach to point set pattern matching, first developed by Huttenlocher et al. [17], is to perform a *branch and bound strategy* on some search

rithms start with a given $4d$ box containing an optimal solution, splits the current box into sub-boxes, kill some of them (those where the current best solution is better than any solution provided by them) and branch on the remaining active sub-boxes. The process stops whenever an "acceptable" solution is found (depends on the required precision). Very recently, those methods have been extended using alignment as in Mount et al. [18] and, Hagedoorn and Veltkamp [19].

To sum up, when scoring transformation $\mathbf{H}$, we first check that we have a large common point set using (4). If so, we refine the score by using either (2) or (3) depending on the zoom factor of $\mathbf{H}$. Finally, we spend more time computing the score by applying (1) for the remaining transformations.

In the following section, we focus on how to report a pool of candidate transformations in which our solution is likely to be.

4. Geometric filtering

The basic idea of *geometric filtering* is to constrain the properties of feature matchings. We can input geometric constraints like zoom value (e.g., in range $[\frac{1}{4}, 4]$) and rotations (e.g., between $[-60\,\text{deg}, 45\,\text{deg}]$), and a precision tolerance ϵ (each feature can be moved into a ball centered at it of radius ϵ). Loosely speaking, we are interested in finding *large common point sets*, that is a set of transformations that match at least a number of points greater than a given threshold. (For example if we want to recover the epipolar geometry, we may ask for at least 7 matching pairs of points). We do not choose a largest common point set because of matching artefacts[†] but it is very likely that our transformation lies in the ones matching large point sets.

If no value of the threshold of the size of a large common point set is given, the system can estimate it in order to run into given time bounds. For any planar transformation T, we associate a characteristic vector $v(T) = (zoom(T), angle(T))$, where $zoom(T)$ is the zoom value of T and $angle(T)$ is the rotation angle from the x-axis. The algorithm can report lexicographically the transformations $T_1, T_2, \ldots$ (that is such that $v(T_1) \leqq v(T_2) \leqq \ldots$) so that two job processes can run simultaneously: (1) reporting potential transformations (2) scoring the transformations and determining if a plausible transformation has been found so far.

Let α be the percentage of points required to match up to an absolute error ϵ (that is $\max\{\lceil\alpha|\mathcal{S}_1|\rceil, \lceil\alpha|\mathcal{S}_2|\rceil\}$ points at least). Parameter α is useful in practice since it reflects the perspective distribution of common feature points, *inliers*, and possibly occluding parts (hidden features or *outliers*). For example, a 50% overlap

α/k	1	2	3	4	5
0.15	6.1	55	327.9	1199.3	13238.3
0.20	7.2	26.4	77.1	689.6	18586.1
0.35	2.9	8.4	19	59.5	321.4
0.5	2.1	3.7	7.8	25.6	30.7
0.60	1.7	2.7	5	12.2	14
0.75	1.4	2	2.2	3.5	4.2

Table 1 Number of times, l, that the program enters the `while loop` before finding a good tuple ($1 \leqq k \leqq 5$) which defines an (α, ϵ)-match. We used the pseudo-random generator `drand48()`.

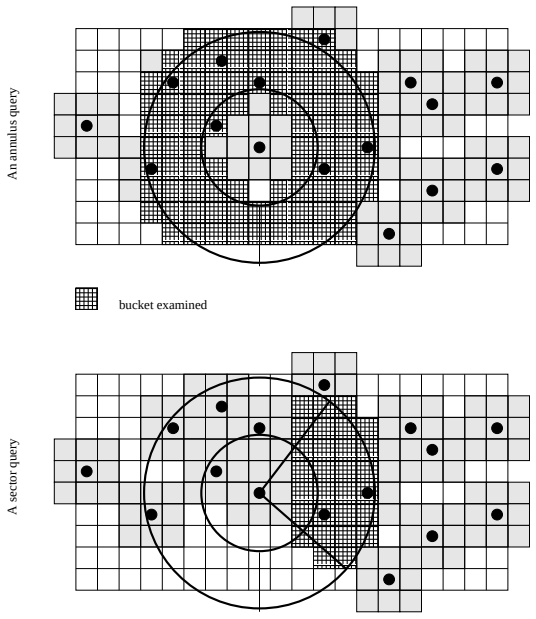
at constant zoom with 50% of outliers of $n = 40$ point features, will match around 10 features (*ie.* $\alpha = 0.25$). We are looking for a (α, ϵ)-match, i.e. a transformation $\mathbf{H}$ that matches at least αn points up to some error tolerance ϵ.

Algorithm 1 The core selection algorithm

```
1: H = Id
2: while not found a (α, ε)-matching homography H do
3:    Choose r points S'_1 from S_1
4:    Draw at random a k-tuple P_1 from S'_1
5:    while not STOP do
6:       Draw at random a k-tuple P_2: k = |P_2| points of S_2
7:       (* We use geometric FILTERING *)
8:       for all permutations P'_2 or P_2 do
9:          Compute the homography H that perfectly matches tuple P_1 to tuple P'_2
10:         if S'_1 is a (λ, ε)-match in S_2 then
11:            if S_1 is a (α, ε)-match in S_2 then
12:               if the characteristics of matched points correlate satisfactorily then
13:                  STOP
14:               end if
15:            end if
16:         end if
17:      end for
18:   end while
19: end while
```

Let k be the number of features required in $\mathcal{S}_1$ and in $\mathcal{S}_2$ for computing a basic transformation that perfectly matches pair by pair these $2k$ features (edges, corners, triple junctions, etc.). Using corners, we have k set to 4. Since we know that a significant proportion of points in $\mathcal{S}_1$ will ϵ-match, choosing at random a k-tuple $\mathcal{P}_1$ and computing all induced homographies with all other k-tuples of $\mathcal{S}_2$ will lead to the more efficient (by a square root factor) Monte-Carlo algorithm. Table 1 shows experimentally the number of times we loop before finding a good k-tuple (independent of n

straints imposed by the selection of $\mathcal{P}_1$. We call it *geometric filtering* and it allows both in *practice* and *theory* to speed up the algorithm significantly. For sake of simplicity, let us assume that we look for a translation and a rotation matching features of $\mathbf{I}_1$ into $\mathbf{I}_2$. Each detected feature point p lies in a ball $\mathcal{B}(p, \mu)$. We set $\epsilon = 4\mu$ in order to take into account the fuzziness of our features. Let p^* denote the "visual" feature point that our feature extraction algorithm have approached by p ($p^* \in \mathcal{B}(p, \mu)$). Given any two feature points, we have $|d_2(p_1^*, p_2^*) - d_2(p_1, p_2)| \leqq 2\mu$. Let $L_1, L_2 \in \mathcal{S}_1$ be corner points in image $\mathbf{I}_1$ (resp. image $\mathbf{I}_2$) that have been drawn randomly. Assuming uniform scaling factor, instead of comparing (L_1, L_2) to all pairs (R_i, R_j), we choose for *every point* $R_i \in \mathcal{S}_2$ as candidate for the second point R_j, all the points inside the ring whose center is R_i with minimum circle $\mathcal{B}(R_i, d_2(L_1, L_2) - 2\mu)$ and width 4μ (see Figure 5). This can be done easily using buckets (as depicted in Figure 5) and extend naturally to zoom ranges and angle sectors. We balance the preprocessing time of building the buckets with the processing time of querying it according to the intrinsic difficulty of the point set (see [1] for details). The idea depicted in Figure 8 is that we can count possibly faster than reporting points inside the annuli query. Therefore we can adapt the size of the buckets in order to accomodate the batched ring queries faster. Indeed, loosely speaking, having a too fine bucket costs much prepocesing time but answer queries quickly, while having a coarse-sized bucket is fast to build but queries take more time as illustrated in Fig 8.



DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.