Efficiently Constructing Mosaics from Video Collections

Frank Liu
Stanford University
liuf@stanford.edu

Rob Hess
Flickr Vision Group
hess@yahoo-inc.com

Alan Fern
Oregon State University
afern@eecs.orst.edu

Abstract

In this paper, we describe an efficient method for creating mosaics from collections *of videos. Our method is based on a utility maximization formulation which we optimize greedily. We employ a function for quickly estimating mosaics* without *computing image features that allows us to efficiently take greedy steps while still achieving user-definable goals for mosaic quality. Indeed, we demonstrate using a number of single- and multi-video experiments that our approach can construct high-quality mosaics in only a fraction of the time required to perform the operations undertaken by existing video mosaicing algorithms. While we focus in this work on the application of panorama construction, our method has a wide range of applications, such as super-resolution, summary, and indexing.*

1. Introduction

In this paper, we investigate the problem of creating a mosaic from a *collection* of videos. More specifically, given a collection of videos, the problem in which we are interested is how to select a set of representative frames from those videos (where criteria for representativeness may be user-defined) and to register those frames with one another in a common coordinate system. This is a difficult problem with both end-user applications, such as panorama construction [1], as well as applications in other areas, such as super-resolution [11, 9], summary and indexing [7], compression [8], etc. [12, 13].

The primary challenge of the multi-video mosaicing problem is computational. In particular, good methods already exist for generating accurate mosaics from an input set of still images (e.g. [1]). However, because these methods are typically quadratic in the number of input images, applying them out of the box to a collection of many videos, each potentially with hundreds or thousands of frames, is simply infeasible.

To our knowledge, the multi-video mosaicing problem has never been investigated. Specifically, all research efforts on video mosaicing of which we are aware have focused on the single-video problem, which, even for videos of reasonable length, is still computationally challenging. Early approaches to this problem simply computed image-to-image transformations between contiguous video frames to register the video with the coordinate system defined by the first video frame [7, 8]. Unfortunately, when transformations are computed locally between frames, as in these approaches, instead of being optimized globally (e.g. using bundle adjustment, as in [1]), small errors can compound and lead to severe inaccuracy.

In more recent single-video work, Steedly *et al.* [11] attempt to integrate bundle adjustment in a tractable way by using an image feature-based method to select keyframes in the video based on degree of overlap. Under this method, each intermediate frame is compared only to the first keyframes before and after it in the video. Specifically, Steedly *et al.*'s bundle adjuster has an overall time complexity of $O(K^2 + n - K)$, where n is the total number of frames in a video and K is the number of keyframes selected, whereas the naive approach has an overall complexity of $O(n^2)$.

Be that as it may, Steedly *et al.*'s method still involves detecting and matching image features in every video frame. In our experience, these operations are prohibitively expensive, even for a single video. For example, just computing SIFT features [10] in a single 500-frame, 720×480 video can take more than 20 minutes on a typical desktop machine. For a single video, this order of feature computation time dominates the bundle adjustment times reported in [11], and it is clear that, for even a moderately large video collection, computing image features for every video frame poses a huge computational burden.

In this paper, we present a novel approach for constructing single- or multi-video mosaics that is designed to overcome the limitations of previous single-video approaches. In particular, our method takes advantage of the fact that it is rarely necessary to include every video frame (in a single video or in a collection) in the mosaic. Instead, our method attempts to select a minimal subset of frames that achieve user-defined objectives for mosaic quality (e.g. a specific degree of redundancy, anytime maximal scene area, com-

978-1-4799-6683-7/15 $31.00 © 2015 IEEE
DOI 10.1109/WACV.2015.37

plete connectedness of selected frames, etc.).

Once frames are selected by our method, their mosaic-registration transforms are optimized globally, using a bundle adjustment procedure similar to the one employed in [1]. Because we take a greedy approach to frame selection, this style of full bundle adjustment is typically still tractable, since at any given iteration of our procedure, the subset of selected video frames minimally meets the user-defined quality objectives.

In what follows, we describe in detail our greedy utility maximization approach for frame selection (Section 2), we discuss possible forms of the utility function (Section 3), and we outline a function for quickly estimating mosaics without computing image features (Section 4). In Section 5, we present a full set of qualitative and quantitative experiments to demonstrate our approach's effectiveness, and we conclude by summarizing our work in Section 6.

2. Utility Maximization Framework

We model multi-video mosaicing as a utility maximization problem. In particular, let $V_1, \ldots, V_N$ be a collection of N videos, where each video V_i, $i = 1, \ldots, N$, is itself a set of n_i video frames, i.e. $V_i = \{v_1^{(i)}, \ldots, v_{n_i}^{(i)}\}$, and let $\mathcal{V} = V_1 \cup \cdots \cup V_N$, denote the set of all video frames. Let $\mathcal{M}$ denote the space of possible mosaics, and let $M : 2^{\mathcal{V}} \rightarrow \mathcal{M}$ be a function that computes a mosaic from a subset of the video frames in the collection $\mathcal{V}$ (we discuss the form of $M(\cdot)$ we use in Section 5.1). Finally, let $U : \mathcal{M} \rightarrow \mathbb{R}$ be a function that assigns utility values to mosaics. (In general, $U(\cdot)$ may be designed to capture arbitrary properties of mosaic quality as desired by the user. We discuss possible forms of $U(\cdot)$ in Section 3.) Then, our goal is to find the subset of video frames $S^* \subseteq \mathcal{V}$ that yields the highest mosaic utility:

$$S^* = \arg\max_{S \subseteq \mathcal{V}} U(M(S)). \tag{1}$$

It may be possible to maximize some utility functions $U(\cdot)$ by simply selecting $S^* = \mathcal{V}$. However, because the state of the art in mosaic construction (including the mosaic builder we use, described later in Section 5.1) involves computing image features in all of the input images—an extremely costly operation, as stated above—we wish to avoid including all of $\mathcal{V}$ in the final mosaic.

And, while it is possible to embed preferences about the number of frames selected into most forms of utility function, additional computational issues arise, since, even given the temporal structure of each video, performing the maximization in (1) exactly is extremely difficult. In fact, this problem is closely related to the set cover problem, which is known to be $\mathcal{NP}$-complete [3]. On the other hand, it is well known that the greedy approximation to set cover is highly effective [2]. Inspired by this fact, we use a greedy approach to approximately optimize (1).

Specifically, our method greedily builds an approximation to S^* one video frame at a time by repeatedly selecting the frame that, when added to the current mosaic, yields the new mosaic with the greatest utility. In other words, given the approximation S_t from the current iteration of our greedy procedure, we select a video frame $v_t^* \in \mathcal{V} \setminus S_t$ such that

$$v_t^* = \arg\max_{v \in \mathcal{V} \setminus S_t} U\big(M(S_t \cup \{v\})\big), \tag{2}$$

and we use v_t^* to form a new approximation $S_{t+1} = S_t \cup \{v_t^*\}$. We repeat this step until some stopping criteria are met.

Again though, an exact search for v_t^* at each iteration of our greedy procedure (2) would require computing image features in every video frame $v \in \mathcal{V}$ in order to compute mosaics $M(S_t \cup \{v\})$. To avoid this, we further define a deterministic function $\widetilde{M}(v, S_t)$ that, without computing image features, estimates the mosaic that would result from adding frame v to S_t (we discuss our specific design for this function in Section 4).

3. The Utility Function

The form of the utility function $U(\cdot)$ can vary depending on the user's specific goals. For example, the utility function can be defined to attain a desired level of redundancy across the space of the generated mosaic, e.g. to enable super-resolution [11, 9] of the mosaic image, or it could be defined to attempt to select a disjoint set of frames that summarizes the video collection.

Because this work is the first to examine the the design space of utility functions for the multi-video mosaicing problem, we concentrate on a single class of utility functions that we believe is broadly applicable. In particular, we focus on the class of utility functions that attempt to cover as much of the area of the scene depicted in the video collection $\mathcal{V}$ as possible, while keeping the number of frames selected to a minimum.

Perhaps the most straightforward utility function in this class is

$$U(M(S)) = \frac{A(M(S))}{|S|}, \tag{3}$$

where $A(M(S))$ denotes the total pixel area of the mosaic generated from image set S. This utility function corresponds loosely to the set-cover interpretation of multi-video mosaicing, encouraging the creation of mosaics that cover the maximum area using the least number of video frames.

It is important to note, however, that the utility function in (3) does not consider the number of connected components present in the mosaic. In particular, because area is maximized when there is zero overlap between images in

the mosaic, this utility function will cause our greedy procedure to prefer to select video frames that are not connected with each other and to fill in gaps only when necessary. This behavior may be useful for some applications such as video collection summarization, but in many others, such as panorama construction or compression, the user may desire a fully-connected mosaic instead. For these applications, a utility function that encourages the selection of overlapping video frames is required.

One way to accomplish this objective is by penalizing the selection of frames that do not connect to the current mosaic. For example, if we let N_S denote the number of connected components in the mosaic $M(S)$ and $\{S^{(k)}\}_{k=1,\ldots,N_S}$ denote the collection of the sets of frames in those connected components, where $S = \bigcup_k S^{(k)}$ and $S^{(i)} \cap S^{(j)} = \emptyset, \forall i \neq j$, we can craft a utility function that penalizes the creation of new mosaic components:

$$U(M(S)) \;=\; \sum_{k=1}^{N_S} A\big(M\big(S^{(k)}\big)\big) + \lambda N_S. \qquad (4)$$

Here, λ is a parameter that determines how much to penalize the creation of each new mosaic component. Intuitively, λ helps determine a threshold on the amount of additional area a video frame must contribute, at each iteration of our greedy procedure, to an already existing mosaic component. Below this threshold, the greedy procedure will choose to begin a new, unconnected component at the given iteration rather than add to an existing component.

Because it may be difficult to characterize areas within the transformed mosaic space in order to determine λ, we can replace the absolute component area $A(M(S^{(k)}))$ in (4) with some form of normalized area $\widehat{A}(M(S^{(k)}))$. There are several ways in which we can choose to define the normalized area $\widehat{A}(M(S^{(k)}))$. A natural first choice might be to simply normalize the absolute area of the mosaic component by the sum the individual areas of its constituent video frames, or, in other words, to compute the fraction of "effective area" contained within a mosaic component. Unfortunately, under this definition of normalized area, in order to grow a mosaic component, we must be able to continually add frames that overlap less and less with the component. This can be difficult to achieve.

We can obtain more reasonable behavior by instead normalizing each component's absolute area by the *average* video frame area in that component, i.e. by defining

$$\widehat{A}\big(M\big(S^{(k)}\big)\big) \;=\; \frac{A\big(M\big(S^{(k)}\big)\big)}{\frac{1}{|S^{(k)}|}\sum_{v \in S^{(k)}} A\big(v, M\big(S^{(k)}\big)\big)}. \qquad (5)$$

Intuitively, this form of normalized area corresponds to the number of average-sized video frames required to comprise the total absolute area of mosaic component $S^{(k)}$ without overlap.

If we replace the absolute area in (4) with the normalized area from (5), our greedy procedure will choose to add to an existing mosaic component whenever a video frame v and a component $S^{(k)}$ can be found such that

$$\widehat{A}\big(M\big(S^{(k)} \cup \{v\}\big)\big) \;\geq\; \widehat{A}\big(M\big(S^{(k)}\big)\big) + 1 + \lambda. \qquad (6)$$

That is, a video frame will be added to an existing component as long as a frame v can be found such that the number of average-sized frames comprising (without overlap) the absolute area of mosaic component $S^{(k)} \cup \{v\}$ is at least $1-(-\lambda)$ more than the number comprising component $S^{(k)}$.

This condition leads to an intuitively appealing interpretation of λ. Specifically, λ acts as a threshold on overlap for video frames with average area. For example, if we set $\lambda = -0.5$, then a frame with average area can overlap with a component by at most 50% to be added to that component. In other words, by setting $\lambda = -0.5$, we encode a preference that each selected frame should increase the area of some existing component by an amount greater than or equal half the area of an average-sized frame. If no such frame and component can be found, our greedy procedure will begin a new component.

In the experiments reported in Section 5, we use a utility function that takes this last form, with normalized areas as in (5).

4. Efficient Mosaic Estimation

In order to avoid computing image features in every video frame in a collection, we use a deterministic function $\widetilde{M}(v, S)$ that quickly estimates the mosaic that would result from adding video frame v to set S. By using this function instead of the full mosaic builder $M(\cdot)$ (which computes image features in all input frames), we can rapidly assess the utility that results from adding each individual frame in the video collection to the current mosaic.

4.1. Velocity and Location

The key component of our mosaic estimator $\widetilde{M}(\cdot)$ is an estimate l of the location within the current mosaic of each video frame $v \in \mathcal{V}$. In order to compute these location estimates, we model each video's velocity relative to the current mosaic. Specifically, for each video frame $v_j^{(i)} \in \mathcal{V}$, we maintain an estimate $\boldsymbol{\delta}_j^{(i)}$ of that frame's velocity by estimating the speed and direction of each of the frame's four corners relative to the current mosaic (modeling the velocities of frames' corners allows us to account for effects such as zoom, rotation, etc.).

The velocity estimates $\boldsymbol{\delta}_j^{(i)}$ are calculated based on the actual mosaic locations of frames already selected for inclusion in the mosaic in previous iterations of our procedure. Specifically, before our procedure's first iteration all

frames' velocity estimates are set to an empirically determined initial value. Once two frames $v_{j_1}^{(i)}$ and $v_{j_2}^{(i)}$, $j_1 < j_2$, from the same video are selected and assigned to the same mosaic component with no other intermediate frames also assigned to the same component (i.e. $\exists\ k$ s.t. $v_{j_1}^{(i)}, v_{j_2}^{(i)} \in S_t^{(k)}$ and $\nexists\ j$ s.t. $j_1 < j < j_2$ and $v_j^{(i)} \in S_t^{(k)}$), their velocities are computed using their exact known mosaic locations as $\mathbf{l}_{j_2}^{(i)} - \mathbf{l}_{j_1}^{(i)}/(j_2 - j_1)$ (where we assume $j_2 > j_1$). The velocities of all other frames are set equal to the velocity of the closest frame in the same video that is already included in the mosaic. In other words, for each frame $v_j^{(i)} \notin S_t$, if we let

$$v_{j^*}^{(i)} = \arg \min_{v_k \in S_t \cap V_i} |k - j|, \tag{7}$$

then we set $\boldsymbol{\delta}_j^{(i)} = \boldsymbol{\delta}_{j^*}^{(i)}$.

Once velocity estimates are computed, we can estimate each frame's location relative to the current mosaic. Specifically, if, for each frame $v_j^{(i)} \notin S_t$, $v_{j^*}^{(i)}$ is defined as in (7), then we set

$$\mathbf{l}_j^{(i)} = \mathbf{l}_{j^*}^{(i)} + \sum_{k=j^*}^{j} \boldsymbol{\delta}_k^{(i)} \tag{8}$$

(with the appropriate adjustments made if $j < j^*$), where we assume unit time between contiguous frames.

4.2. Matchability from Mosaic Overlap

In addition to just estimating the location of each frame within the current mosaic, we also need to determine whether the final mosaic builder will be able to match each frame to an existing component in the current mosaic instead of being forced to use that frame to begin a new mosaic component. In the end, a frame's final matchability to an existing mosaic component is determined by the mosaic builder's ability to find and match distinctive image features between that frame and the existing mosaic. However, since our goal here is to be able to estimate a mosaic without computing image features, we use two separate heuristics to determine matchability.

The first of these heuristics is based on the frame's estimated overlap with the existing mosaic. In particular, this heuristic measures matchability based on the assumption that frames with a greater degree of overlap with the existing mosaic are more likely to contain image features that can be matched with the mosaic. Specifically, we define

$$o_j^{(i)} = \begin{cases} 1 & \exists\, k \text{ s.t. } \widetilde{O}\big(v_j^{(i)}, M\big(S_t^{(k)}\big)\big) > \theta \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

where $\widetilde{O}(\cdot)$ estimates the proportion of overlap between a video frame and a mosaic component using the frame's location estimate $\mathbf{l}_j^{(i)}$, and θ is a threshold on overlap below which it is likely that a frame will be unmatchable to the current mosaic due to a lack of matchable image features (we empirically set $\theta = 0.35$ in our experiments).

4.3. Matchability from Proximity to Unmatchable Frames

Our second matchability heuristic is based on the premise that it is likely possible to match frames from the same video into a single mosaic component, and the reason a frame may not be matchable to the rest of its video is because it does not contain enough distinctive image features to match with other frames in the video. These types of frames will not be matchable to the existing mosaic no matter how much they overlap with it.

To accomplish this for a given frame $v_j^{(i)}$, we define $v_{j^-}^{(i)}$ as the closest unmatchable frame from the same video, and we set a threshold τ on $v_j^{(i)}$'s temporal proximity to $v_{j^-}^{(i)}$, below which $v_j^{(i)}$ is also to be considered unmatchable. Specifically, we set

$$u_j^{(i)} = \begin{cases} 1 & |\, j - j^-| > \tau \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

In practice, we vary τ based on the distance between $v_{j^-}^{(i)}$ and the nearest frame in the same video that *is* matched to the current mosaic, since the presence of nearby matchable frames also indicates the presence of matchable image features.

4.4. Final Mosaic Estimate

Once the variables $\mathbf{l}$, o, and u have been computed for a frame v using (8), (9), and (10), respectively, the final mosaic estimate $\widetilde{M}(v, S_t)$ can be computed for that frame and the current mosaic. If $o = u = 1$ (i.e. we believe the frame can be matched to the current mosaic), we compute $\widetilde{M}(v, S_t)$ by simply transforming v to its estimated mosaic location $\mathbf{l}$ and adding it onto the appropriate mosaic component $M(S_t^{(k)})$. Otherwise, if either $o = 0$ or $u = 0$ (i.e. we believe the frame *cannot* be matched to the current mosaic), we compute $\widetilde{M}(v, S_t)$ by adding v to $M(S_t)$ as a new, separate component.

5. Results

In this section, we demonstrate our approach's effectiveness by presenting qualitative and quantitative results from a number of single- and multi-video experiments. We compare against the baseline methods of random frame selection or selection of every k^{th} frame. However, as our experiments show, these methods exhibit major shortcomings. For example, random frame selection can yield highly variable results, sometimes producing a reasonable mosaic, but often missing portions of the scene. Similarly, attempting to set

k to select every k^{th} frame can be very difficult, since setting k too large can result in missing portions of the scene, and setting k too small can lead to the selection of far more frames than necessary, thus eliminating any potential efficiency gains.

5.1. Implementation Details

In all of the experiments described below, our method uses a utility function that takes the form of (4) but uses the normalized component area from (5) instead of absolute component area. The final mosaic builder $M(S_t)$ we use is similar to the one described by Brown and Lowe in [1]. Specifically, it computes SIFT features in each video frame in S_t and matches these between all pairs of frames in S_t (these computations are cached to avoid duplicating them for S_{t+1}). Based on the number of feature matches between each pair of frames, frame-frame matches are formed to construct the set of connected mosaic components $\{S_t^{(k)}\}$, and homographies relating the frames in each component to each other are optimized jointly using a Levenberg-Marquardt-based bundle adjuster. Note that, because final mosaic construction is not our focus, we use a basic mosaic builder that assumes planar mosaics and does not perform any sophisticated blending. As a result, some of our visual results exhibit parallax and ghosting effects.

5.2. Single-Video Experiments

Each video in our single-video experiments was shot on one of two university campuses with a handheld digital camera. In order to test our approach's consistency under various conditions, each scene was shot using several different camera motion patterns (e.g. left-right, right-left, center-up-down, etc.). Since the results for each pattern were similar, we present here representative results for one video of each scene. The final mosaics and timing results for these videos are summarized in Figure 3. In each case, a high quality mosaic is constructed after only 4-5 iterations of our greedy procedure.

Since Steedly *et al.*'s algorithm [11] requires computing image features in every video frame, the time required to do so for each of our single-video examples is also reported in Figure 3 for reference. By comparison, our procedure's total running time is a fraction of this time alone.

5.3. Multi-Video Experiments

We assessed our procedure's ability to perform multi-video mosaicing on a data set containing 15 videos, each with between 400 and 600 720×480 frames, from the American football domain. Mosaicing in this domain is motivated by the real application of sports video analysis, where the generated mosaics are useful for initializing video registration [4] and for computing background models for player tracking and other tasks [6, 5]. This data set poses a challenging test for our procedure, since it was shot with a camera that rapidly panned, tilted, and zoomed to follow the game's action. Nonetheless, our procedure was quite successful on this collection. As depicted in Figure 4, our greedy procedure was able to select frames that covered the football field almost completely within 25 iterations, taking only about 41 minutes to do so. By comparison, simply computing image features in every frame of each of the 15 videos in the data set, as required by Steedly *et al.*'s algorithm [11], took over 4 hours.

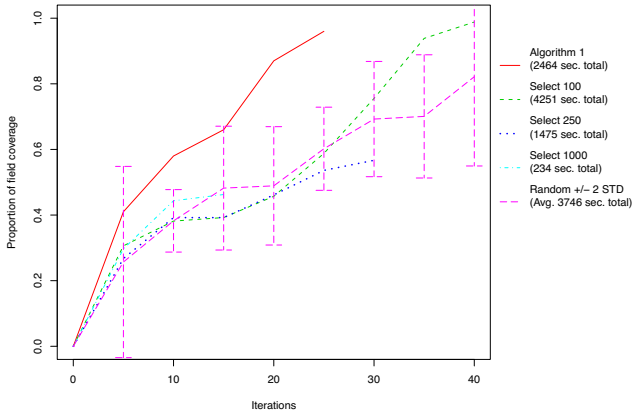


Figure 1. Proportion of the in-bounds football field covered by the mosaic computed after each iteration of our greedy procedure on the 15-video American football data set. Our approach exhibits strong anytime properties, with the connected scene coverage increasing at every iteration. Nearly the entire football field is covered by the mosaic after only 25 iterations of our algorithm. For comparison, the coverage attained from selection of every 100, 250, and 1000 frames is presented along with the average coverage attained over 5 runs of random frame selection (with error bars indicating 2 standard deviations). Though selection of every 100 frames eventually achieves similar coverage, our method still clearly outperforms each of the baselines by a comfortable margin.

Figure 1 details our procedure's progress per iteration in covering the football field depicted in the video collection by plotting the proportion of the in-bounds football field contained within the mosaic at each iteration. As a comparison, baseline results for the same data set are given for random frame selection and selection of every 100, 250, and 1000 frames. In the figure, the proportion of coverage was computed by registering the mosaic at each iteration with a scale model of the football field and computing the proportion of the field covered by registered mosaic pixels. As is clearly evident, our method comfortably outperforms the baseline methods. By the 25^{th} iteration, our algorithm selects frames covering 96% of the in-bounds football field. In other words, within only 25 iterations, our procedure is able to select video frames covering essentially the entire scene depicted in a challenging 15-video collection.

Figure 2. A mosaic generated from a collection of 14 short (100-200 frame), high-resolution videos, seven of which contain a portion of the depicted scene, and seven of which were "noise" videos. The mosaic contains essentially the entire scene and was constructed in 15 iterations of our algorithm in a total of 2450 seconds.

In an additional multi-video experiment, we tested our method with a collection of 14 videos, seven of which contained portions of the same scene and seven of which were "noise" videos. Our greedy procedure was able to identify the relevant videos and extract frames representing essentially the entire scene in only 15 iterations, totalling 2450 seconds. The resulting mosaic is depicted in Figure 2.

6. Conclusion

In this paper, we have presented a flexible and efficient method for building mosaics from a *collection* of videos. Our work represents the first known exploration of the multi-video mosaicing problem. Our algorithm builds mosaics one frame at a time by greedily optimizing a utility function that can be defined to capture arbitrary, user-defined properties of mosaic quality. Its efficiency stems from the fact that it avoids computing image features in every video frame. Using single- and multi-video experiments, we have demonstrated that our approach can build high-quality mosaics in a fraction of the time required by existing video mosaicing methods.

References

[1] M. Brown and D. Lowe. Automatic panoramic image stitching using invariant features. In *IJCV*, 2007. 1, 2, 5

[2] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3), 1979. 2

[3] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman, 1979. 2

[4] R. Hess and A. Fern. Improved video registration using non-distinctive local image features. In *CVPR*, 2007. 5

[5] R. Hess and A. Fern. Discriminatively trained particle filters for complex multi-object tracking. In *CVPR*, 2009. 5

[6] R. Hess, A. Fern, and E. Mortensen. Mixture-of-parts pictorial structures for objects with variable part sets. In *ICCV*, 2007. 5

[7] M. Irani and P. Anandan. Video indexing based on mosaic representations. *Proceedings of the IEEE*, 86(5), 1998. 1

[8] M. Irani, S. Hsu, and P. Andan. Video compression using mosaic representations. *Signal Processing: Image Communication*, 7(4–6), 1995. 1

[9] M. Irani and S. Peleg. Improving image resolution by image registration. *Graphical Models and Image Processing*, 53(3), 1991. 1, 2

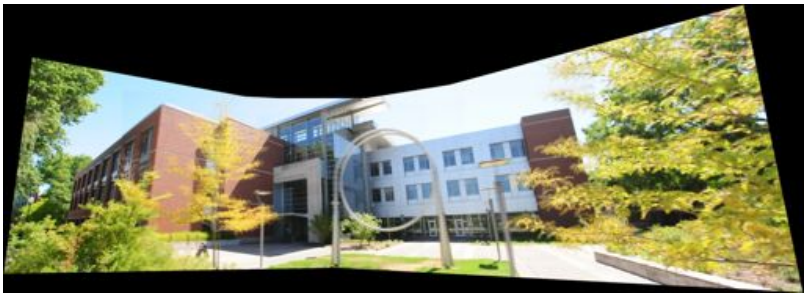
[10] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2), 2004. 1

[11] D. Steedly, C. Pal, and R. Szeliski. Efficiently registering video into panoramic mosaics. In *ICCV*, 2005. 1, 2, 5

[12] Z. Zhu, G. Xu, and X. Lin. Constructing 3D natural scene from video sequences with vibrating motions. In *Virtual Reality Annual International Symposium*, 1998. 1

[13] Z. Zhu, G. Xu, E. M. Riseman, and A. R. Hanson. Fast construction of dynamic and multi-resolution 360-degree panorama from video sequences. In *IEEE International Conference on Multimedia Computing and Systems*, 1999. 1

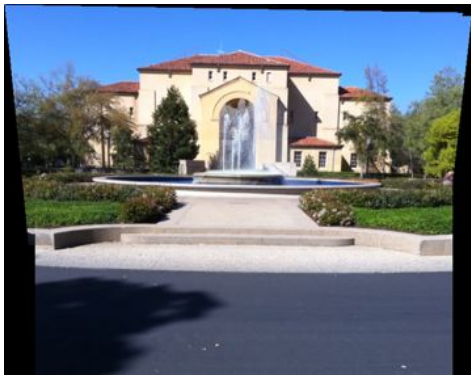
(a) 288 frames; 1280 × 720 down-sampled to 711×400; shot center-down-up; 5 frames selected in 57 seconds vs. 401 seconds to compute SIFT features in all frames



(b) 96 frames; 1280 × 720 down-sampled to 711 × 400; shot right-left; 5 frames selected in 40 seconds vs. 159 seconds to compute SIFT features in all frames

(c) 192 frames; 1280 × 720 down-sampled to 711 × 400; shot left-right; 5 frames selected in 105 seconds vs. 303 seconds to compute SIFT features in all frames

(d) 1000 frames; 1280 × 720 down-sampled to 711 × 400; shot right-left; 4 frames selected in 75 seconds vs. 1085 seconds to compute SIFT features in all frames



(e) 750 frames; 1280 × 720 down-sampled to 711 × 400; shot nearly stationary at center with a fast pan downwards and then upwards in the middle of the video; 4 frames selected in 55 seconds vs. 1284 seconds to compute SIFT features in all frames

Figure 3. Mosaicing results for five single-video experiments. Depicted for each experiment is the final mosaic generated after the specified number of iterations of our greedy procedure. In each case, high quality mosaics are constructed in only four or five iterations of our algorithm, and SIFT features are computed for only the corresponding number of frames. Included for reference in each subfigure caption is additional information for each experiment, including the total time required to compute SIFT features in every frame of the given video.

(a) 5 iterations; 27 seconds

(b) 10 iterations; 256 seconds

(c) 15 iterations; 702 seconds

(d) 20 iterations; 1365 seconds

(e) 25 iterations; 2464 seconds

Figure 4. Mosaics generated for the 15-video American football data set after 5, 10, 15, 20, and 25 iterations of our greedy procedure. The total run time after each iteration is also given (note that these do not scale linearly with the number of iterations because the utility function $U(\cdot)$ must be computed for additional videos as they are selected for observation by our procedure in later iterations). Nearly the entire football field is covered with a high-quality mosaic after only 25 iterations of our procedure, or only about 41 minutes. By comparison, just computing SIFT features in every frame of each video in the data set took 4 hours and 7 minutes.