Markerless Augmented Reality

George Kudrayvtsev

April 25, 2019

I. Introduction

Augmented reality has taken the world by storm after the barrier to entry lowered; with mobile operating system developers introducing straightforward SDKs such as Apple's ARKit and Android's ARCore, even those with no understanding of computer vision can develop real-time AR applications.

This work presents a combination of techniques learned throughout CS6476 into a near-real-time augmented reality video processor. Given an input video or image, the processor will inject it into a real-world scene, creating the illusion that it actually exists in the real world by following the geometry and projection parameters of the scene.

II. Algorithm Discussion

The processing algorithm is based largely on [9]. Let's assume we begin with 4 coordinates representing the corners of the injected frame, $\mathbf{c}_0$; we will discuss automatically choosing a suitable starting location shortly. The corners are assumed to represent an orthographic projection onto the input video geometry in the first frame; in other words, the mapped surface is initially assumed to be perpendicular to the camera lens. Though techniques for mapping the scene geometry from the get-go using point clouds exist [3, 12, 14], they often require pre-existing scene knowledge. Many other methods (such as those employed by the Kinect) depend on depth sensor information which is not present here.

With the corners mapped for frame t_0, we need to determine the transformation that occurs from $t_0 \to t_1$. Given sufficient corresponding points in the frames, we can use RANSAC [2] to approximate the resulting homography, $\mathbf{H}_1$, then $\mathbf{c}_1 = \mathbf{H}_1\mathbf{c}_0$. To find corresponding points, we use the ORB feature detector [8]; it gives the best quality-to-performance ratio which we need for near-real-time processing [11].

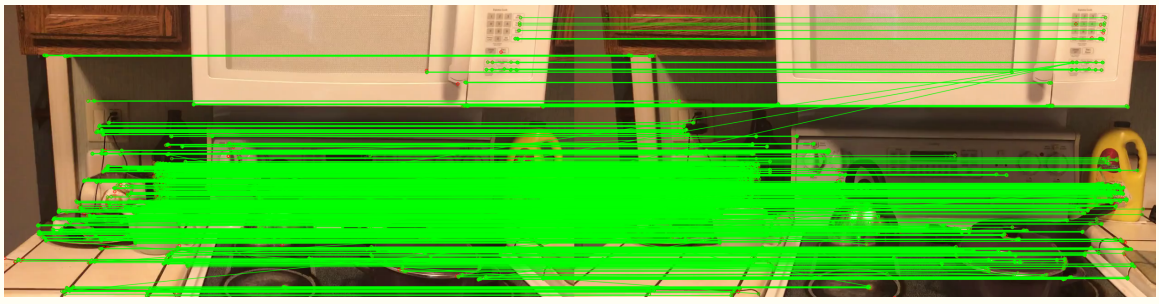

Figure 1: Keypoint matching between frame t and $t+1$.

To find the warping from the unprojected injection frame, we calculate a second homography between the four injection frame corners $\{(0,0),(\text{width},0),\ldots\}$ and $\mathbf{c}_1$ directly. RANSAC is not necessary since we have a direct mapping between four correspondence points [5, pg. 80].

Figure 2: Demonstrating the robustness of the algorithm to both rotation and perspective change (zooming) from one of the sample videos, albeit with slight drift towards the end. The left-most image is the first frame, for reference of the scene.

Start Location

The starting location is chosen automatically from the first frame of the video. As we've already mentioned, the projection surface is assumed to be perpendicular to the viewing direction. The algorithm does preprocessing on the frame to simplify the scene and coalesce "similar" areas that have minor lighting or texture variations; this is visually summarized in Figure 3. Specifically, bilateral filtering on the grayscaled frame followed by a quantization that keeps the top 3 bits reduces the image to a series of simplified components. That image is run through connected component labeling, with small components[1] being merged into their neighbors. After edge detection on the final simplified image and a filtering of small edge contours, the distance transform allows us to rank components by their size based on their distance to an edge (i.e. another component). Finally, the largest distance peak is chosen as a suitable location for injection.

Figure 3: Calculating a suitable starting location. From left to right: the simplified scene after blurring, quantization, and a closing morphology to eliminate small components; the distance transform on the resulting edge image; and the final image with the injected frame. Notice that the microwave is not chosen; this is because of its proximity to the edge and aspect ratio incompatibility.

Blending

As an experimental feature, we explored individually blending each injected frame with projected surface. At a significant computation cost (discussed in Performance), blending produces a more natural and aesthetic effect worth using on high-performance machines or when offline processing is an option. The blending weights are the distance transform of the injected frame: pixels closer to the edge have a smaller weight. We clip the non-edge pixels

[1] specifically, components occupying $< 5\%$ of the total image

by eroding the thresholded image. Then, the injected frame is blended with the scene using the mask and blend weights. The results are shown in Figure 4; clearly, the injected frame looks much more natural with blending.

Figure 4: A processed frame without blending (left) and with blending of 25% of each edge (right).

Problems

By treating each pair of frames independently, we can avoid drift and build-up of homography errors. On the other hand, this can result in massive adjustments to the projection from one frame to another because the sequence isn't considered as a whole. Without smoothing or coalescing with other motion-detection methods such as Lucas-Kanade or SIFT optic flow [6, 7], jitter can occur.

Furthermore, the method heavily relies on the presence of "good features;" without sufficient correspondence points, the resulting homography is completely incorrect. Scenes such as the one in Figure 5 are too plain to have sufficient RANSAC inliers to determine the correct homography. Clean video is also essential: fast movements, shaky hands, and lens refocusing all cause homography errors resulting in incorrect projections.

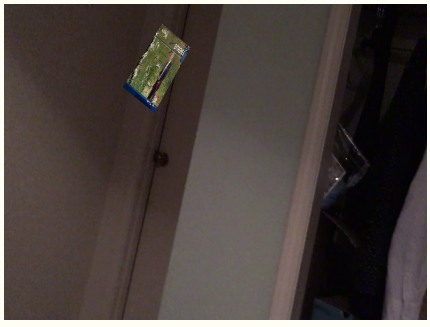

Figure 5: With insufficient correspondence points, the homography can go crazy.

Blending has edge cases as well, though they are pure artifacts rather than projection errors and much less noticeable. Since the amount of "edge to blur" changes with the projection transformations, the image may appear slightly more or less blended relative to other frames depending on the shift.

III. Performance

Performance tests were performed on a Intel i5-7300U CPU (2.60GHz) running Arch Linux. All performance tests were done with 1280x720 (HD Ready) videos. Neither blending nor injection are done if the injection area completely exceeds the viewport bounds; corner and homography updates are still done, but a massive amount of time is saved (especially when blending is enabled) when the injection area pans out of the frame. Naturally, downsampling to a lower resolution would garner large performance gains, and in fact may be a viable optimization for many of

the steps: downsampling prior to feature detection could be a viable avenue for improvement, though lower quality features might means higher likelihood of mismatches and homography errors.

Detector	Processing Time (s)	Frames Per Second
ORB	73.0	17.5
BRISK	91.6	13.9
AKAZE	183	7.0

Table 1: Processing times for a single 1277-frame 720p video.

Though [11] gives us great insights about the performance of each feature detector and their relative weaknesses, manual testing confirmed the superiority of ORB. Table 1 highlights how much faster ORB is than the other tested detectors (testing was done without blending). Generally, the difference between injecting a video or an image is negligible: videos are *slightly* slower (< 1 fps penalty) due to the additional I/O necessary.

The visual improvement provided by the adaptive non-maximal suppression algorithm described in [1] was not worth the performance penalty. Like blending, it dominated the cost of each frame; this was due to the increased number of necessary features. Its effect on projection quality was negligible, especially with a "busy" video with plenty of trackable features, so it was ultimately excluded from the code. From the paper itself, they cite the benefits of ANMS as being most prevalent given small amounts of overlap between keyframes:

> *"It is important that interest points are spatially well distributed over the image, since for image stitching applications, the area of overlap between a pair of images may be small."* [1, pg. 2]

With such small deltas between our video frames, ANMS provides little benefit for the cost.

Blending hurts performance significantly, nearly doubling processing time. For example, a particular 1277-frame video took 73s without blending (17.5 fps) using ORB, but took 101s with 10% blending (12.7 fps); that's approximately a 38% performance penalty.

Table 2 shows the breakdown of the top 3 costliest operations on a profiled execution of the processor on the same video using the ORB detector with and without blending.

	Most Expensive	cost ←	
Standard (60.8s)	Feature detection and matching (40.7%)	I/O (23.9%)	Frame injection (14.4%)
Blending (106.9s)	Blending (47.3%)	Feature detection and matching (25.2%)	I/O (14.2%)

Table 2: A breakdown of the top 3 costliest subroutines under the two most common scenarios (15% blending was performed). The percentage is of the total execution time.

IV. State-of-the-Art

As mentioned in the Introduction, the state-of-the-art in augmented reality is probably the impressive real-time processing supported by the two major mobile SDKs. As demonstrated in many videos online, these boast a far more robust feature set, though they sometimes suffer similar pitfalls.

Most significantly, Google's ARCore features natural lighting and shadow simulation on its objects (see Figure 6) [10], while Apple's ARKit appears to have pre-calculated lightmaps though it does support realistic rendering. Though research exists exploring light source estimation both with [4] and without [4, 13] markers, it does not reach the level of quality in Google's ARCore. Furthermore, both SDKs feature non-orthographic projection which isn't present here. As discussed in the introduction, there are segmentation methods for both images and point clouds to estimate multiple planar surfaces [3, 14, 12], but such techniques aren't integrated here. Furthermore, the advanced set of sensors (depth, infrared) on the iPhone and other cutting-edge smartphones make such techniques far more reliable, efficient, and stable.

Figure 6: A demonstration of real-time lighting estimation in Google's ARCore.

Both our method and the state-of-the-art struggle in environments with low features, non-traditional geometry (like glass countertops), or strange lighting (heavy specularity). Both ARKit and ARCore experience jitter in such environments, though they appear much more stable and recoverable relative to some of the extreme error cases in our methods.

The markerless augmented reality system we've presented here has a fairly robust near-real-time processor. Though it occasionally requires "Goldilocks" input footage—proper lighting without much specularity, no motion blur, gentle transformations in high-feature areas—it can handle such footage with little tweaking. The starting-point finding algorithm reliably finds a suitable location for the projection, even if it may not be the intuitively "best" location

to the human eye. Its issues and performance are not out-of-line for the state-of-the-art, and is a great stepping stone into a deeper AR project.

References

[1] Brown, M., Szeliski, R., and Winder, S. Multi-image matching using multi-scale oriented patches. In *Conference on Computer Vision and Pattern Recognition* (2005), IEEE.

[2] Fischler, M. A., and Bolles, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6 (June 1981), 381—395.

[3] Galloa, O., Manduchia, R., and Rafii, A. CC-RANSAC: Fitting planes in the presence of multiple surfaces in range data. *Pattern Recognition Letters 32*, 3 (November 2011), pp. 403–410.

[4] Kanbara, M., and Yokoy, N. Real-time estimation of light source environment for photorealistic augmented reality. In *Proceedings of the 17th Int'l Conference on Pattern Recognition* (2004), IEEE.

[5] Kudrayvtsev, G. An unofficial companion guide to the Georgia Institute of Technology's CS6476: Computer Vision. From here, 2019.

[6] Liu, C., Yuen, J., Torralba, A., Sivic, J., and Freeman, W. T. SIFT flow: dense correspondence across difference scenes. In *European Conference on Computer Vision* (Berlin and Heidelberg, 2008), Springer-Verlag.

[7] Lucas, B. D., and Kanade, T. An iterative image registration technique with an application to stereo vision. In *Proceedings of Imaging Understanding Workshop* (1981), pp. 121–130.

[8] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. ORB: an efficient alternative to SIFT or SURF. In *Int'l Conference on Computer Vision* (2011), IEEE.

[9] Simon, G., Fitzgibbon, A. W., and Zisserman, A. Markerless tracking using planar structures in the scene. In *Int'l Symposium on Augmented Reality* (2000), IEEE and ACM.

[10] Sugden, B., et al. Virtual light in augmented reality, October 2014. US Patent #US8872853B2 *[Google Patents]*.

[11] Tareen, S. A. K., and Saleem, Z. A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. In *Int'l Conference on Computing, Mathematics and Engineering Technologies* (2018), IEEE.

[12] Trevor, A. J. B., Gedikli, S., Rusu, R. B., and Christensen, H. I. Efficient organized point cloud segmentation with connected components.

[13] Wang, Y., and Samaras, D. Estimation of multiple directional light sources for synthesis of augmented reality image. *Graphical Models 65* (2003), 185–205.

[14] Yang, M. Y., and Förstner, W. Plane detection in point cloud data. Tech. Rep. TR-IGG-P-2010-01, Department of Photogrammetry, University of Bonn, 2010.