Online Environment Model Estimation for Augmented Reality

Jonathan Ventura
University of California, Santa Barbara

Tobias Höllerer
University of California, Santa Barbara

ABSTRACT

Augmented reality applications often rely on a detailed environment model to support features such as annotation and occlusion. Usually, such a model is constructed offline, which restricts the generality and mobility of the AR experience. In online SLAM approaches, the fidelity of the model stays at the level of landmark feature maps. In this work we introduce a system which constructs a textured geometric model of the user's environment as it is being explored. First, 3D feature tracks are organized into roughly planar surfaces. Then, image patches in keyframes are assigned to the planes in the scene using stereo analysis. The system runs as a background process and continually updates and improves the model over time. This environment model can then be rendered into new frames to aid in several common but difficult AR tasks such as accurate real-virtual occlusion and annotation placement.

Index Terms: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Augmented reality; I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Stereo; I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Color;

1 INTRODUCTION

For mixed and augmented reality applications to combine real and virtual worlds efficiently and effectively, they need some model of the real environment as well as the virtual. Users need to situate annotations in the real world, and virtual interactions take place against a real world backdrop. Virtual content needs to be rendered seamlessly into real world imagery. The augmented reality experience becomes more convincing and useful when virtual content respects and responds to the real environment. Environment modeling enables this powerful relationship to be realized.

We can arrive at such an environment model through different means depending on the AR scenario. If we generate the real environment from an *a priori* computer model, such as a CAD model, then no further effort is needed. Without a pre-existing model, we need to sense and reconstruct the environment, possibly with the help of user interaction. Offline modeling programs can be used, for example, which take a set of images or a video sequence as input. However, having to model the environment beforehand introduces major restrictions on the AR experience. A complete set of observations of the environment needs to be collected and manually organized before any interaction can take place. This generally slow and cumbersome process makes it difficult to bring AR into unknown environments. Furthermore, real-time changes to the environment are not reflected in offline models. We would like to enable convincing live augmented reality experiences in unprepared environments, as part of our research agenda called "Anywhere Augmentation [6]."

In this paper, we introduce a novel online modeling and rendering approach for augmented reality. This approach uses a single moving camera to roughly model the environment with textured, planar surfaces. Although this type of model cannot capture all of the depth detail of complex scenes, it still proves very useful for core AR tasks such as annotation and foreground object occlusion. The model improves and expands over time, as more image data is acquired. Our modeling and rendering techniques complement existing approaches to feature-based tracking and retain the real-time performance afforded by state-of-the-art solutions in that area. We show how our environment model enables occlusion of virtual objects by foreground objects, and point-and-click annotation with accurate depth and normal estimation.

After discussing related work in Section 2, we describe our approach to environment model estimation and view synthesis in Section 3. Notes on our implementation are given in Section 4, and we examine view synthesis, occlusion and annotation results in Section 5. Finally, conclusions and directions for future work are discussed in Section 6.

IEEE International Symposium on Mixed and Augmented Reality 2009
Science and Technology Proceedings
19 -22 October, Orlando, Florida, USA
978-1-4244-5389-4/09/$25.00 ©2009 IEEE

2 RELATED WORK

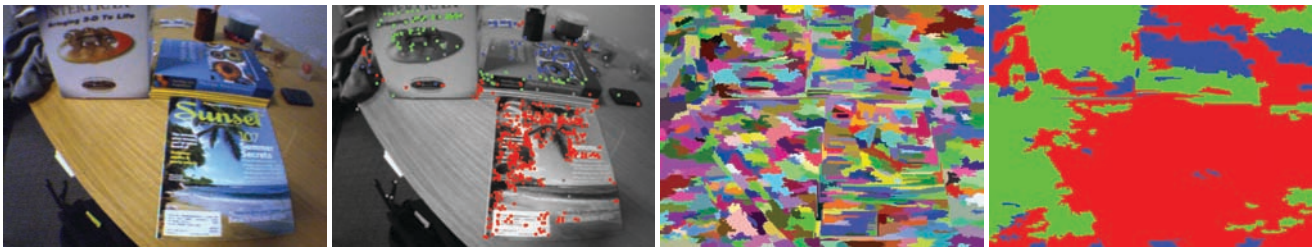
Previous work has explored online scene modeling for visualization and interaction. Rachmielowski and co-authors developed a system which finds suitable keyframes during SLAM tracking and uses those keyframes for 3D reconstruction [12]. They reconstruct a scene model by meshing sparse 3D points using a Delaunay triangulation in 2D image-space. Thus the reconstruction is unlikely to be a good approximation to the actual surface, unless the density of the point cloud is very high. Later work by Chekhlov and others uses RANSAC to find planes in a SLAM map, as in our system [3]. We go beyond this prior work by using multi-view matching to determine the extent and texture of discovered planes.

One use of our modeling and rendering system is to detect foreground objects which should occlude virtual content. Previous approaches to automatic occlusion detection have used a stereo camera to estimate depth, which is then used for compositing virtual content [16, 7]. In contrast, our system uses a single camera, and maintains a static model of the environment. Berger described a method for matching contours over several frames to find the outlines of occluded objects [1]. This promising early work has, to our knowledge, not been shown to be real-time or sufficiently robust for continuous AR viewing. Klein describes an AR system which uses a known model for detecting occlusions [8]. Their occlusion refinement step could be used in combination with our occlusion detection method to produce nicely blended edges. Others have used physical props such as specially colored foam [10] or planar patterns [11] to aid color segmentation of the user's hands.

We also use the environment model produced by our system to enable easy annotation in an AR environment. The offline modeling methods discussed above would also enable annotation, albeit with the same limitations we mentioned. Reitmayr, Eade and Drummond introduced a semi-automatic annotation method which uses online triangulation and tracking of user-specified polygons [13]. However, their system is limited to annotations that can be tracked by their edge tracker. A laser range finder has also been explored as a fast annotation device in unknown environments, which works well for depth estimation in the outdoor case [15].

3 DENSE MODELING FOR AR

In our system we model the environment using planar surfaces. The assumption of planar surfaces makes sense for many AR applications, both indoors and out. Inside a room, we may want to use a



(a) (b) (c) (d)

Figure 1: *Plane detection and assignment.* The tracker collects keyframes as the camera moves (a). In this sequence, the system has detected three planes; we show the triangulated 3D points in the map colored according to their plane assignment (b). By matching with a second keyframe and aggregating over superpixels (c), we assign all pixels in the keyframe to a plane (d).

table, the walls, the ground, or flat objects such as books for information display, annotation and interaction. Outside, the most likely surfaces for annotation and interaction are sides of buildings, signs, and the ground. All of these surfaces could be modeled by our system, as long as they have enough distinguishing visual texture.

We assume the availability of a real-time camera tracking system which generates three types of data: a camera pose for every frame; tracked feature points in 3D space; and a set of keyframes which observe those points. The points and keyframes together form "the map." We use Klein's PTAM system, for which code is publicly available [9].

3.1 Finding planes

To find planes in the environment, we look for planar groups of well-estimated feature points. A SLAM tracking system is one source of such points, especially indoors. A laser range finder would be more appropriate for outdoor AR, where points need to be sampled from a greater distance than multi-view matching can handle [15].

Once we have assembled a set of 3D estimated points from the environment, we find groups of points which roughly lie on planes. We allow some error in the planar fit, to allow for error in the point estimates and also for surfaces that are not perfectly planar. The RANSAC estimation procedure is well-suited to robustly finding planes in the point set [5]. Our RANSAC procedure randomly samples sets of three points in the point set, fits a plane to the three points, and classifies remaining points as inliers or outliers based on the point's distance from the plane along the normal. The plane hypothesis with the most inliers is returned; if a sufficient number K of inliers is found, we accept the plane. The procedure is then repeated on the remaining outliers until no more planes are found.

To classify inliers and outliers, we use a distance error threshold of three centimeters in our experiments. Although this was appropriate for our indoor experiments, the parameter might need to be tuned for other environments, such as outdoor urban scenes. We chose K, the number of inliers needed to accept a plane, according to the density of points expected in the set; $K = 50$ worked well for our data sets. This threshold needs to be high enough to avoid finding spurious groups of points which fit a plane but do not actually represent a physical surface in the environment.

The plane finding procedure is initially run on the first batch of points added to the tracking map. The PTAM tracker initializes tracking by finding a plane in the initial feature tracks. This world coordinate system is transformed so that this plane lies at $z = 0$. Because of this initialization step, we always start with the plane $z = 0$ in our set of detected planes.

When we add a new point to the map, we first see if the point can be classified as an inlier to a plane. Initially, the only plane available is the $z = 0$ plane mentioned above. If the point is not an inlier to any known plane, the point remains unlabeled. During our plane re-estimation procedure, we try to find new planes in the set of unlabeled points, using RANSAC as described above. Then all planes are re-estimated based on their sets of inliers.

Our plane re-estimation procedure is run after adding a new keyframe to the map, and after full bundle adjustment converges. Figure 1(b) shows the three groups of points found in one tracking sequence.

3.2 Texturing planes

Once the planes have been estimated, we can determine the visible extent of each plane in the keyframe images using multi-view matching.

For both the multi-view matching process and the later rendering step, we would like to use the keyframe which has the best visual overlap with the reference frame. A simple approach would be to rank a keyframe by its translational distance to the reference camera. However, cameras which are close in space may not have the most visual overlap, either because of rotation or occlusion. Instead, we rank a keyframe by the number of feature observations it shares with the reference frame. This is a more accurate indicator of visual overlap which is already measured by the tracking system.

In our system, we already have a set of plane hypotheses generated from the point set. We test these plane hypotheses by projecting the matching frame into the reference using the corresponding homography (with radial distortion applied). For each projection of a keyframe, we calculate the per-pixel matching error by Euclidean distance in RGB space.

Several recent multi-view modeling papers have shown the advantages of using over-segmentation (superpixels) for matching cost aggregation [2, 17]. The over-segmentation naturally respects image edges, and aggregating over a larger region can help when matching texture-less regions. We use a recent image segmentation method which can segment a 640×480 image in less than one second on a 2.1 GHz machine [4]. Real-time speeds are not necessary since the reconstruction runs as a background thread. Figure 1(c) shows an example over-segmentation.

After segmenting the reference image, we can aggregate matching costs from the image warping described above. We sum up per-pixel matching costs over each segment, and for each segment choose the plane resulting in the lowest cost. Figure 1(d) shows the patch labels chosen for the frame from Figure 1(a). After segmentation, multi-view matching and patch label assignment take under 0.5 ms per plane. Thus segmentation is clearly the bottleneck for the multi-view matching. Fortunately, keyframes only need to be segmented once, when they are first added to the map.

The system maintains a queue of keyframes which need to be processed or re-processed by the multi-view matching procedure. After a new frame is added to the map, or after the set of planes is updated and re-estimated, we add all keyframes to this queue. This way, all keyframes will reflect the updated planar model, and

our scene model can expand and improve as the environment is explored.

3.3 View synthesis

After plane detection and multi-view matching, each keyframe contains its own planar model of a portion of the environment. Given any camera pose, we can synthesize a new view of the environment from one or more keyframes. In our system, we use the camera pose as estimated by the tracking system to synthesize a view of the scene. Obviously, the camera itself provides the true image of what is observed from that pose. The availability of the true camera image gives us an advantage for rendering: we can directly measure the per-pixel rendering error. The synthesized view allows us to estimate depths and normals, and produce an occlusion map for AR rendering, as described in Section 5.1.

Our rendering algorithm combines nearby keyframes together to produce a synthesized view. We may need to use more than one keyframe to cover the entire extent of the desired camera view. Up to four of the nearest keyframes (as determined by the ranking described in Section 3.2) are rendered from the viewpoint to be synthesized. The rendering procedure is similar to the image matching procedure described in Section 3.2. For each plane and each nearby keyframe, we calculate the homography H which projects the matching frame into the reference frame. However, only pixels which are assigned to that particular plane are projected (with radial distortion). At a single pixel location, we collect all the projected pixels from nearby frames. We output the pixel with minimum color matching error, calculated by Euclidean distance in RGB space. It is possible that a superpixel aggregation similar to that used in the multi-view matching would improve our rendering results, but the segmentation algorithm is too slow for real-time performance.

4 Implementation

The frame-to-frame tracker and the renderer run in the same thread, while multi-view matching runs in the same background thread which handles bundle adjustment for the tracker. Because both the renderer and the multi-view matcher use the OpenGL pipeline, they each have their own graphics context, and share the pipeline using a mutex lock.

We used a Unibrain Fire-i camera with a 2.1mm wide-angle lens, and tested our system on a 2.5 GHz dual-core laptop, with a GeForce 8600M GT graphics card. We were able to achieve about 7 fps while running the entire system.

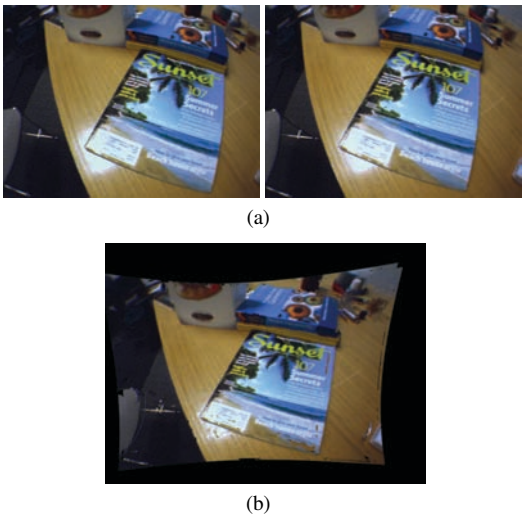
5 Results

Figure 2 shows an example of the results of the view synthesis algorithm. The two keyframes used for rendering have already been labeled with planes. The area outside of the view of either keyframe is black; eventually new keyframes added to the map could fill in these areas. The output is rendered with radial distortion, to match the output of the camera.

Some errors in the view synthesis can be seen in areas that do not physically belong to any of the estimated planes, and thus are mislabeled by the matching algorithm. The view synthesis result is not as important there, since it is unlikely that AR interaction would take place in these areas. It is possible that such mislabeled regions could be detected by thresholding the average color error in an image patch.

5.1 Dynamic real-virtual occlusion

Often in augmented reality interfaces, virtual objects are placed on the static background environment, while real foreground objects (such as hands or interaction devices) dynamically move above the virtual content. Without any ability to determine occlusions from the camera viewpoint, virtual objects always have to be rendered on top of the camera image. This can break the augmented reality illusion, since the virtual objects are not correctly occluded, as in Figure 3(a).



(a)

(b)

Figure 2: *View synthesis example.* Two keyframes (a) are used to produce a synthesized view (b) from a new viewpoint.

One useful benefit of our online modeling approach is its ability to detect when foreground objects such as hands occlude the background, and thus should occlude rendered virtual objects as well. Figure 3(b) shows this effect. Without a full depth map of the scene, we cannot handle all occlusion cases, such as when a virtual object actually should occlude a foreground object. However, we handle the most common case, since foreground objects such as hands and interaction devices will most often lie between the user's viewpoint (the camera) and the virtual content.

We calculate the occlusion map by thresholding the differences between the camera image and the synthesized view. The differences are already calculated and used by the rendering algorithm described in Section 3.3, so our occlusion approach adds little overhead. When compositing the camera image with the virtual object image, any pixels with difference $d > \tau$ are removed from the virtual object image. This is performed in a GPU shader which also applies radial distortion to the virtual image. In our experiments we found a threshold of $\tau = 0.03$ to work well (where each color channel has the range [0,1]). This threshold depends on how constant the illumination is in the scene. The presence of specular surfaces, variable lighting conditions, and shadows will adversely affect the background subtraction, and thus require a higher threshold.

By thresholding the per-pixel difference between input and background, we are essentially performing the well-known technique of background subtraction [14]. However, our system has a major difference: we are able to produce a background image from any viewpoint, whereas traditional methods require a static camera. The only requirement is that keyframes in the map should represent only the background environment, and should not contain foreground objects. In a typical usage of our system, we first scan the environment with the camera, to allow enough keyframes to be recorded. After this brief exploration period, we tell the tracker to stop adding new keyframes to the map, so that we can safely put foreground objects in the camera's view without corrupting the map.

5.2 Annotation with the environment model

A second benefit of our modeling and rendering system is that we can estimate both depth and a normal direction for each pixel of the camera image, in real-time. This has the potential for greatly im-

(a) (b) (c) (d)

Figure 3: *Occlusion detection and annotation.* In a typical AR system, virtual objects like the synthetic green cube (a) are rendered on top of foreground objects, which breaks the AR illusion and hinders spatial comprehension. Using moving viewpoint background subtraction, we correct for occlusion artifacts (b). We also use the model for online text annotation (c), with accurate depth and normal information. A later frame from the same sequence (d) shows how the model expands to new areas as they are explored.

proving AR interaction with unknown environments, without any offline modeling of the environment.

Our view synthesis method assigns to each pixel in the camera image a plane from the set of detected planar surfaces. Using ray casting, we can calculate the depth of a pixel by intersecting with its plane in world space, after accounting for radial distortion. The normal is given by the plane normal.

With this technique, the user can easily add annotations to the environment by simple point-and-click, which is an important AR task. In Figures 3(c) and 3(d) we show how a text banner can be placed on any of the planes detected by the system.

6 Conclusions and Future Work

We have introduced a novel system for online modeling of arbitrary AR environments, by multi-view analysis of detected planar surfaces. We demonstrated our system's effectiveness for improving the augmented reality experience. Our approach is fast and can be used as part of a live system. The model updates and improves as more frames are added, and can expand as the user explores the environment. With our image-based rendering method, we enable simplified annotations which are automatically oriented to the scene model, and foreground object occlusion detection.

An important avenue of future work is better detection of either spurious planes or badly labeled image patches. The stereo matching error could be used to detect parts of the image which are not well-modeled by the detected planes. This also might give evidence for detected planes which do not represent physical surfaces in the environment, and thus would allow us to remove such planes. This would reduce the importance of the threshold parameter on the number of inliers needed to accept a plane.

Acknowledgements

The authors wish to thank Georg Klein and Pedro Felzenszwalb for making their source code available. This work was partially supported by NSF CAREER grant IIS-0747520, and NSF IGERT grant DGE-0221713.

References

[1] M. O. Berger. Resolving occlusion in augmented reality: a contour based approach without 3d reconstruction. In *Conference on Computer Vision and Pattern Recognition (CVPR '97)*, 1997.

[2] M. Bleyer and M. Gelautz. Graph-cut-based stereo matching using image segmentation with symmetrical treatment of occlusions. *Image Commun.*, 22(2):127–143, 2007.

[3] D. Chekhlov, A. P. Gee, A. Calway, and W. Mayol-Cuevas. Ninja on a plane: Automatic discovery of physical planes for augmented reality using visual slam. In *ISMAR '07: Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–4, Washington, DC, USA, 2007. IEEE Computer Society.

[4] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, 2004.

[5] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[6] T. Höllerer, J. Wither, and S. DiVerdi. *Location Based Services and TeleCartography*, chapter Anywhere Augmentation: Towards Mobile Augmented Reality in Unprepared Environments, pages 393–416. Lecture Notes in Geoinformation and Cartography, G. Gartner and M. Peterson and W. Cartwright, Eds. Springer, February 2007.

[7] M. Kanbara, T. Okuma, H. Takemura, and N. Yokoya. A stereoscopic video see-through augmented reality system based on real-time vision-based registration. In *IEEE Virtual Reality*, pages 255–262, 2000.

[8] G. Klein and T. Drummond. Sensor fusion and occlusion refinement for tablet-based AR. In *Proc. Third IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pages 38–47, Arlington, VA, November 2004.

[9] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *ISMAR '07: Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–10, Washington, DC, USA, 2007. IEEE Computer Society.

[10] W. Lee and J. Park. Augmented foam: a tangible augmented reality for product design. In *Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 106–109, Oct. 2005.

[11] S. Malik, C. McDonald, and G. Roth. Hand tracking for interactive pattern-based augmented reality. In *ISMAR '02: Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, page 117, Washington, DC, USA, 2002. IEEE Computer Society.

[12] A. Rachmielowski, N. Birkbeck, M. Jägersand, and D. Cobzas. Realtime visualization of monocular data for 3d reconstruction. In *CRV '08: Proceedings of the 2008 Canadian Conference on Computer and Robot Vision*, pages 196–202, Washington, DC, USA, 2008. IEEE Computer Society.

[13] G. Reitmayr, E. Eade, and T. W. Drummond. Semi-automatic annotations in unknown environments. In *ISMAR '07: Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–4, Washington, DC, USA, 2007. IEEE Computer Society.

[14] C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages –252 Vol. 2, 1999.

[15] J. Wither, C. Coffin, J. Ventura, and T. Hollerer. Fast annotation and modeling with a single-point laser range finder. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 65–68, Sept. 2008.

[16] M. M. Wloka and B. G. Anderson. Resolving occlusion in augmented reality. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 5–12, New York, NY, USA, 1995. ACM.

[17] C. L. Zitnick and S. B. Kang. Stereo for image-based rendering using image over-segmentation. *Int. J. Comput. Vision*, 75(1):49–65, 2007.