Augmented Reality through Real-time Tracking of Video Sequences Using a Panoramic View

Christophe Dehais Matthijs Douze Géraldine Morin Vincent Charvillat
IRIT/ENSEEIHT, 2 rue Camichel, 31071 Toulouse cedex 7, France {dehais, douze, morin, charvi}@enseeiht.fr

Abstract

We propose a 2D approach for Augmented Reality (AR) applications where the real scene is modelled as a static panorama. We adapted a sparse tracking method based on homographies to track the orientation and zooming parameters of the camera during a video sequence. AR scenarii (synthetic object insertion, real object or character extraction) can be performed in arbitrary static environments (from wide outdoor scenes to virtually augmented desktops or conference rooms).

Introduction

AR applications need to register virtual augmentations with respect to the real scene in order to melt them seamlessly. We tackle this problem by registering into a panorama each frame of a video sequence (Figure 1(a)). This gives us a mapping from which orientation and zooming parameters of the camera are recovered. This information can then be used to setup virtual augmentations, for example generated from a CAD model rendered in real-time or from a compatible panorama rendered offline. This framework also allows extracting Video Object Planes (VOP) and integrating them in a virtual environment.

Possible uses of this method include for example, a conference room whose background is known, but where the attendees move or CCTV systems where louts have specific types of motion.

In a first section, we review the different approaches used to design augmented reality applications. Then, section 2 presents the context and the different stages of our approach. Section 3 details the tracking method we used. Finally, in section 4, experimental results demonstrate the use of our tracking method in different augmented reality scenarii.

1. Related work

The registration of augmentation objects with respect to a real scene can include geometrical information: this is the case for pose estimation. Depending on how much information is known about the scene, different solutions can be considered.

Many approaches try to recover 3D information about the scene using vision-based techniques or physical sensors [1]. Among vision-based methods, some try to track the pose of an object (often polyhedral), whose wireframe model is known [8, 4].

Methods inspired by image-based rendering aim at using as little 3D information as possible. In [5], Hung et al. propose to augment a panorama by compositing it with video objects. A virtual 3D referential has to be localised on the panorama. Augmenting objects are generated by “view-morphing” (synthesized from images taken at different angles).

Our approach requires less user interaction: it only needs the registration of the initial image.

2. System overview

Our system works in two stages. The first one is performed offline: we acquire a panoramic view of an arbitrary static environment. This reference view can be captured by a dedicated device like an omnidirectional camera or be composed by mozaicing range views of the scene (left part of Figure 1(a)). We can then alter this panorama to create a new augmented panorama. In Figure 2(a), our reference panorama is augmented with a virtual desktop (this can be done easily through image editing or more rigorously by using 3D landmarks). We may also render a compatible virtual background (Figure 2(d)).

During the second stage we shoot the scene with a classical video camera (right part of Figure 1(a)). The optical cen-
3. The tracking algorithm

We shoot the scene with a classical pinhole camera and we want to remap the image sequence with respect to the reference panorama, denoted I. This is possible if the scene is motionless (occlusions are handled in section 3.2) and if the optical centre coincides with the optical centre of the panoramic camera (the camera may only turn and zoom in and out).

At time $t$, I can be remapped with a frame $I_t$ of the video sequence using a homography. There exists $\mu = [h_1 \cdots h_8]^T$ such that, for any 3D point which has projection $(x, y)$ on I and $(x', y')$ on $I_t$:

$$
\begin{bmatrix}
    x' \\
    y'
\end{bmatrix} = h \left( \begin{bmatrix}
    x \\
    y
\end{bmatrix} \right) = \begin{bmatrix}
    h_1 x + h_2 y + h_3 \\
    h_4 x + h_5 y + h_6
\end{bmatrix}
$$

The parameter vector $\mu$ must be estimated for each frame of the video sequence. We assume that the initial mapping $\mu_0$ is given by an ad hoc method (automatic image registration or user interaction).

3.1. Point-based tracking of a region

Here, we suppose that there is an interest region, $J_0$, on I, that is entirely contained in all the images of the sequence. This assumption will be relaxed in section 3.2.

**Modelling and resolution.** At frame t, the homography with parameters $\mu_t$ maps from $J_0$ to $I_t$. Thanks to the chromatic calibration, we assume that the image of a 3D point does not change colors, or graylevels in our case: $\forall x \in J_0, I_t(h(\mu_t, x)) = I(x)$.

We assume also that, between the shots, the motions of the object and the camera are small. This means that the difference $\Delta \mu_t = \mu_t - \mu_{t-1}$ is small, so that we can use $\mu_{t-1}$ as a close initial estimate of $\mu_t$. The tracking is sparse: only a finite subset of reference points $K_0 = \{x_1, ..., x_n\}$ of $J_0$ is used. We take more than 200 points to get a reliable estimate of the homography. We use a melt of Harris’ interest points [7] and arbitrarily spread regular points (Figure 1(b)).

Thus, tracking consists in finding the series $(\mu_t)_{t \in N}$ that minimizes the quadratic differences

$$
O_t(\mu) = \sum_{x \in K_0} (I_t(h(\mu, x)) - I(x))^2
$$

$$
\mu_t = \arg\min_{\mu} O_t(\mu)
$$

We implement two resolution methods to solve the problem: a non-linear method [2] and that by Jurie and Dhome [6]. They are carried out in two stages:

1. An offline learning stage, that uses only the panorama. The algorithm chooses the reference points $K_0$. It records their graylevels $G_0 = \{I(x)\}_{x \in K_0}$ and calibrates the colors of the tracking camera.

2. An online tracking stage: using graylevels $G_t = \{I_t(h(\mu_{t-1}, x))\}_{x \in K_0}$, the algorithm attempts to find $\Delta \mu_t$ that minimizes $O_t(\mu_{t-1} + \Delta \mu_t)$. This stage executes in real time.

**Estimating $\Delta \mu$.** The non-linear (NL) method [2] directly minimizes the criterion $O_t(\mu)$. This is a non-linear least squares problem, thus it can be solved with an iterative Levenberg-Marcquart algorithm [3]. The algorithm is accurate (sub-pixel when bilinear interpolation is used), but needs a close estimate of the solution to converge.

The Jurie and Dhome [6] method (JD), begins with a costly learning stage. It simulates “experiments” that consist in applying a disruption on the points of $K_0$ by random homographies $\mu' = \mu_0 + \Delta \mu$, and measuring the resulting graylevel variation vector at the reference points $\Delta G = \{I(h(\mu', x))\}_{x \in K_0} - G_0$. When the disruption is not too big, $\Delta \mu$ and $\Delta G$ are related linearly: $\Delta \mu \approx A \Delta G$. By carrying out $N$ such experiments, A can be estimated in the least squares sense.

During the tracking stage, we measure $\Delta G_t = G_t - G_0$, and compute $A \Delta G_t$. By a composition of homographies we can boil down to the relation on I and compute the corresponding $\mu_t$.

The standard deviation $\sigma$ of the disruptions during the learning stage allows us to adjust the robustness-precision tradeoff: if $\sigma$ is big, the tracking is less prone to “loose” its target, but it shakes more, and vice versa.

**Synthesis.** One of our contributions has been to combine the two algorithms. We apply the robust methods first, then the more precise ones. We compute several $(A_t)_{t=1,n}$ matrices during the JD learning stage by applying disruptions with decreasing standard deviations: $\sigma_1 > \sigma_2 > \cdots > \sigma_n$. During the tracking, we apply the corresponding JD steps followed by some NL iterations. We make sure that each new estimate lowers the criterion $O_t$.

3.2. Generalization: tracking over any sequence

There is no guarantee that all the images of the sequence have a region in common, so what will the interest region
The tiles. Therefore, we use various interest regions (tiles) that cover the whole panoramic image. The tiles are not necessarily disjoint. We choose tiles small enough so there is at least one appearing on each video frame, but big enough for the estimation to remain relevant. For each tile, we choose a set of interest points, and do a learning stage.

During the tracking stage, we use $\mu_{t-1}$ to calculate which tiles are fully appearing on image $I_{t-1}$. We apply the tracking algorithm on each of these tiles, which provides several “suggestions” for $\mu_t$.

Combining the results. Some tiles are not reliable, if their background is hidden by an occluding object. Tiles whose interest points have “weird” graylevels or whose optimization criterion is too high are not taken into account in the combination. Each remaining tile provides a mapping to $I$ for each of its 4 corner points. Thus, if we use $r$ tiles, we have $4r$ point correspondences, and $8r$ equations in the components of $\mu_t$. This system is solved using ordinary least squares.

4. Experiments and applications

We first tested the tracking method on a single tile, a synthetic image (of $1024 \times 1024$ pixels) that rotates faster and faster (from $2^\circ$ to $20^\circ$ between two frames). We know the ground truth in this case, so we can define a validation criterion (more reliable than $O_t$).

We found the best combination of regular and interest points, the optimal number of experiments in the JD learning stage ($N$). Then we tested different sequences of tracking methods. The sequence offering the best robustness and precision is: 4 JD steps with $\sigma = 12.5, 10, 5, 2.5$ and 5 NL iterations.

We then tried out different tile sizes and layouts. We found that we needed 10 to 20 tiles per frame, in staggered rows.

4.1. Augmented Reality applications

The first experience is a classical AR scenario, where a CAD model is drawn over the real scene. The second experience is more complex: a real character is extracted from the video sequence and composed with a computer generated background.

Synthetic object insertion. As we track camera orientation and zooming over the sequence, we can add synthetic objects. A panorama with a transparency channel can be rendered to lay the synthetic objects on. For each frame, the inverse homography maps the synthetic panorama to the frame (Figure 2(a,b)). Note that we could also extract some extrinsic and intrinsic camera parameters from the homography to update at each frame an initially calibrated camera matrix and then use it to parameter a real-time rendering engine, so that objects can change or move over the sequence.

Object or character extraction. (Figure 2(d,e,f)) In this case, we remap the panorama with the inverse homography to get a frame background. It is possible then to extract objects that appear only in the video sequence, and use the computed motion of the camera to integrate such objects consistently in a synthesized world.

In background regions, the pixel difference between the frame and the remapped panorama is low. As we handle color images, the maximum difference among the HSV channels gives a grayscale mask (Figure 2(e)). This mask is noisy, so we smooth it with a morphological opening, then threshold it, which gives a binary mask.

4.2. Implementation

To handle real images, our program in C processes a digital video stream on a 3 GHz Pentium 4 under Linux. We
optimized the most costly computations with vector instructions (in SSE2 and AltiVec). The display, in OpenGL, executes asynchronously and does part of the compositing between the video frames and the augmented reality objects. Table 1 shows the timings for each operation, on a 1876 \times 872 panorama (150 tiles) and half-size frames (360 \times 288). In total it executes at about 10 frames per second.

## Conclusion

We have proposed a simple way to perform AR, based on real-time tracking of a video sequence within a panoramic view of the scene. Our experiments show that it is sufficiently flexible for various AR scenarios. Future improvements will aim at improving the stability of the tracker (which sometimes tends to tremble excessively). This could be done using Kalman filtering or reducing the number of parameters of the motion model (a pan/tilt/zoom camera requires only three parameters).

## References