Stable real-time AR framework for training and planning in industrial environments

Luca Vacchetti, Vincent Lepetit, Michal Ponder, George Papagiannakis, Pascal Fua, Daniel Thalmann, Nadia Magnenat Thalmann.

Abstract
Augmented Reality systems can be effectively used to enhance manufacturing and industrial processes. However, not all the existing prototypes of AR systems can be used in industrial environment, due to heavy constraints such as low robustness or cumbersome equipment. Our Augmented Reality system relies on purely passive techniques to solve the real-time registration problem and it can run on a portable PC. We combine a powerful VR component-based simulation framework with Computer Vision techniques, turning it into an Augmented Reality system. The resulting system allows us to produce complex rendering and animation of avatars, and to blend them into the real world. The system tracks the 3D camera position by means of a natural features tracker, which, given a rough CAD model, can deal with complex 3D objects. The tracking method is robust and can handle large camera displacements and aspect changes. The target applications of our AR system are industrial maintenance, repair and training. The tracking robustness makes the AR system able to work in real environments such as industrial facilities and not only in the laboratory.

1 Introduction

Virtual and Augmented Reality are becoming inextricably integrated strands of the new emerging digital visualization fabric. With the advent of the recent powerful, low cost consumer graphic computers, it becomes possible to build highly realistic real-time AR and VR simulations. At the same time, recent developments in human animation have led to the integration of virtual humans into synthetic environments.

As the demand for Augmented Reality systems grows, so will the need to allow these virtual humans to coexist and interact with real objects and scenes. This is especially true in case of industrial and manufacturing applications where AR systems are highly suitable for training, ergonomics evaluation and rapid validation of prototypes before letting them into the manufacturing phase. Virtual teachers can be used to demonstrate complex machine operation to the novice users. Similarly, AR is an ideal approach for designing objects by having a virtual human interactively performing evaluation tests on an object composed of real and virtual components.

Including real machinery and surroundings into the interactive simulation increases realism. It decreases time that would be otherwise required to model complex virtual environments. Finally it eliminates the computation costs involved in rendering them. Fig. 1 illustrates this approach: a virtual worker demonstrates how to use a machine in a complex industrial environment. The user of the AR system can change the point of view at any time, and since the camera position is correctly registered to the real scene, the virtual worker will always be correctly blended into the streaming video. Another application of our system is shown in Fig. 2, where a virtual human guides the user through an unknown building.

We therefore view our contribution as twofold:

- an accurate real-time vision-based camera tracker, which is responsible for the registration of the virtual humans into the streaming video and does not require engineering the environment;
- its integration into an existing VR component-based simulation framework (VHD++), that provides human-computer interface and rendering of realistic virtual humans.

Thanks to the strong component-based character of the VHD++ simulation framework development of the tracking component as a VHD++ plug-in was straightforward.

2 Related Work

Many papers, such as (Drummond and Cipolla 2000), (Neumann and You 1999), (Simon et al.), (La Cascia et al.), have been published on tracking, and some characteristics such as accuracy and speed, can now be found in many existing systems. The robustness is much more challenging.

Previous AR systems required to modify the environment by introducing markers, as it was done in (Kato et al.) for instance. Many other methods, in order to track the camera displacements, exploit certain features of the objects of the scene. These features can be edges, like in (Drummond and Cipolla 2000) or in (Marchand et al. 1999). Unfortunately edge-based methods cannot be effectively used in real industrial environment since the usually cluttered background introduces too strong noise. Our AR system makes use of an approach to real-time tracking based on natural feature points, which are automatically detected and tracked as they appear. Contrarily to previous methods, which also consider natural feature points, our method is not limited to piecewise planar scenes but can handle arbitrarily complex scenes, with no limits to the camera displacement.

Many other algorithms obtain high accuracy, even without an *a priori* knowledge, by matching natural features such as interest points. For example

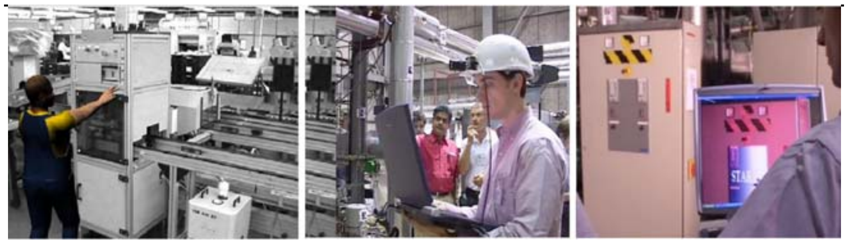
Fig. 1. First and second pictures from left: a virtual human demonstrates the use of a real machine (offline test requiring two machines). Third and forth testing the real-time tracking with basic rendering in a factory (one machine).

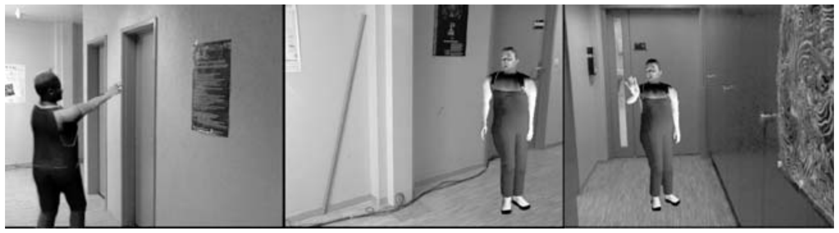
Fig. 2. The same virtual human guides a visitor through an unfamiliar corridor

(Fitzgibbon and Zisserman 1998) process the image sequence hierarchically to derive robust correspondences and to distribute error over the sequence. Considering speed as not critical issue, these algorithms take advantage of time consuming but effective techniques such as bundle adjustment. Many other methods perform the same task for real-time applications but tend to be less reliable since they can not rely on batch computations. Those that work without the *a priori* knowledge are not really practical: for example (Azarbayejani and Pentland 1995) assumes absence of correspondences errors, and (Beardsley et al.) assumes the camera center motion to check if the correspondences respect the epipolar constraint.

This paper is organized as follows: in the next section we present the VHD++ component-based simulation framework, then in Section "Stable real-time camera tracking" we describe our tracking algorithm, and finally we will present our experiments and results.

3 VHD++ for AR applications

The VR part that has been used in our application example presented in Fig. 4 enables integration of heterogeneous simulation technologies, such as real-time 3D rendering, skeleton and skin animation, behavioral control, etc. VHD++ virtual humans show large range of animation capabilities, as introduced in (Ponder et al. 2003). The VHD++ framework constitutes an extendible, real-time, audio-visual

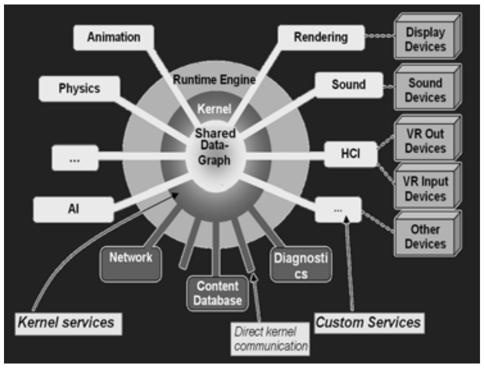

Fig. 3. VHD++ Development Framework

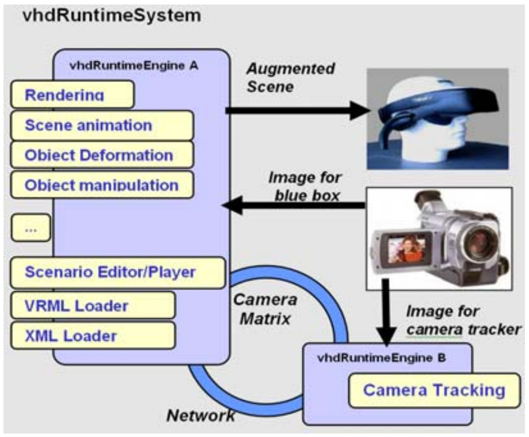

Fig. 4. VHD++ architecture overview by example.

simulation engine with special support of advanced real-time virtual characters simulation technologies.

AR systems rely heavily on the interplay of complementary heterogeneous technologies. Because of that interdisciplinary character, the AR domain can be viewed as a melting pot of various technologies, which are non-trivial to put together (Azuma 1995). The VHD++ is a component-based software development framework that supports composition of high performance, real-time, interactive, audio-visual applications. It provides an extendible set of functional components like 3D rendering, 3D sound, advanced synthetic character simulation, AI, behavioral control, interactive scenario authoring, diagnostics, networking, runtime data base management, etc. VHD++ development methodology relies on massive design (runtime engine) and code (pluggable components) reuse. In effects applications are being composed out of the components rather than developed from the scratch. Fig. 3 shows a high level abstraction of the VHD++ architectural model. Pluggable, custom components encapsulating heterogeneous simulation technologies are marked as the thin spikes coming out of the kernel. Some of them are connected to the rectangles representing device abstractions.

Motivation behind VHD++ is based on the following observation related to the VR/AR systems. It occurs that on the low, system infrastructure level most of the VR/AR systems feature strong concurrency, network distribution, support of various input and output devices and of course real-time performance. It occurs as well that most of the systems use quite similar sets of low level fundamental components responsible for data loading, containment, serialization, sharing of resources, multitasking, synchronization, time scheduling, event handling, networking, brokering, etc. On the high, application level the VR/AR systems tend to draw from a continuously growing spectrum of heterogeneous and complementary simulation technologies which repeat frequently in different configurations depending on particular application requirements and context (e.g. camera tracking, 3D rendering, 3D sound, collision detection, physics, skeleton animation, skin deformation, face animation, cloth simulation, crowd control, behavioral control, multi-modal interaction, etc).

In effect of the above observations the VHD++ semantics has been developed around the following four key semantical elements:

- vhdRuntimeSystem (whole diagram);
- vhdRuntimeEngine (outer embossed circle);
- vhdServices (spikes coming out of the sharable data);
- vhdProperties (sharable data on the diagram).

In order to illustrate the respective roles and mutual relationships between the above elements, Fig. 4 depicts in a schematic way an example of a VHD++ based application, including its possible physical deployment strategy. The example shows our target vhdRuntimeSystem featuring two vhdRuntimeEngines hosting a total of five plug-able vhdServices distributed here over two computer nodes.
Each of the computer nodes features some input/output devices that are used throughout the simulation. We can easily imagine for example that starting from the left, the first vhdRuntimeEngineA hosts some services responsible for multi-modal interaction using connected VR input devices and behavioral control of the simulation. Lower right vhdRuntimeEnginB hosts for example a vhdService responsible for real-time vision based camera tracking and passing of the low bandwidth camera matrix updates to the vhdServices hosted on the previous vhdRuntimeEngineA which takes care of the 3D rendering, synthesis of real and virtual images, 3D sound, character animation. It is one of the deployment scenarios for our VR/AR application example. Existing vhdServices may be easily adapted to new requirements through derivation and overriding of virtual methods. If necessary, developers may provide easily new vhdServices that will be added to the global pool and then available for future reuse.

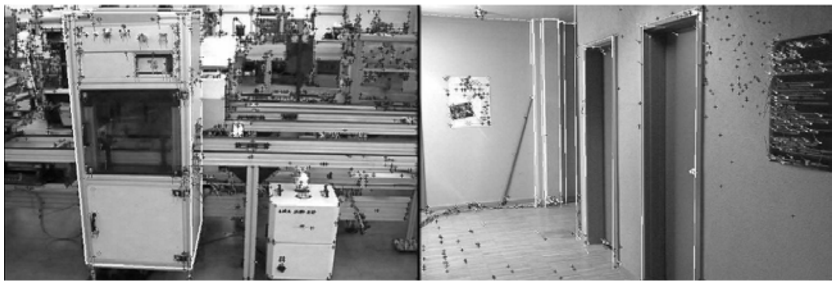

Fig. 5. Matching of interest points: the crosses are the detected interest points in the picture.

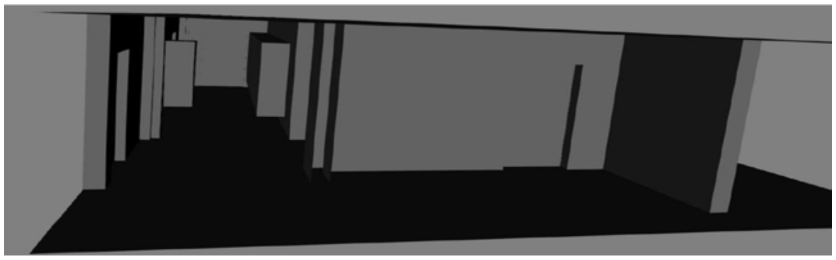

Fig. 6. The model we used for tracking the corridor. It was extruded from a CAD plot and some details have been added. It took about 1 day of work to be designed. It is a conventional VRML model

4 Stable real-time camera tracking

Our tracking method is suitable for many types of 3D textured objects that can be described either as a wire frame models or a triangulated meshes. It starts from 2D matching of interest points, and then it exploits them to infer the 3D position of the points on the object surface. Interest points are points where a discontinuity occurs in the image signal, and can be detected using the method proposed in (Harris and Stephens 1998) or (Shi and Tomasi 1994). They are a reliable primitive to track, since they cannot be confused with any other parts of the texture. Moreover, being a local feature that can be normalized with respect to the luminance, interest points do not depend on lighting conditions.

When the interest points on the object are tracked it is possible to retrieve the camera displacement in the object coordinate system using robust estimation. In the following paragraphs we will describe in detail our tracking algorithm. First we present a simpler version that tracks the camera displacement frame by frame. This method works well but suffers from error accumulation for long sequences. We show how to prevent this problem by considering key-frames (defined offline

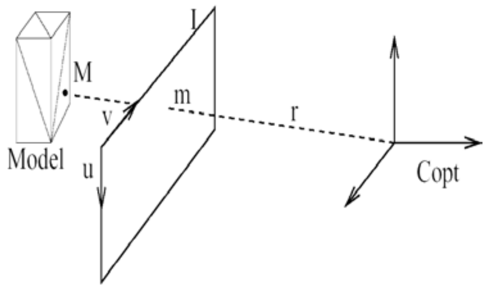

Fig. 7. The back-projection process

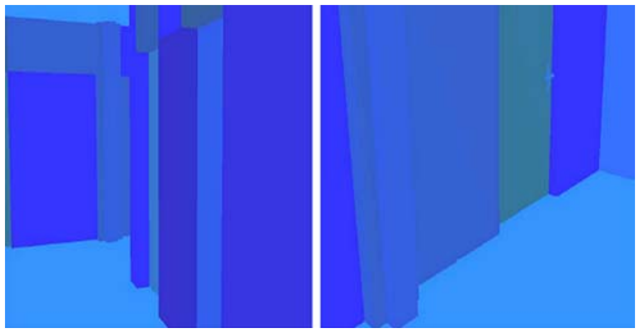

Fig. 8. Examples of “Facet-ID” images.

or online). This method is more complex but allows us to consider sequences without duration restriction.

4.1 Initialization

We use computer vision techniques to roughly compute intrinsic parameters by means of a calibration grid. The algorithm starts when the user moves the camera or the object close to a known position that may be shown on the screen.
It is important to stress that a rough, approximate adjustments is sufficient.
The matching algorithm receives as input the incoming image and a “bootstrap” reference frame; if the frames are close enough, the point matching number increases above a given threshold and the tracking starts.

4.2 Simple recursive tracking

First, we detect the strongest interest points in the current source image using the Harris corner detector (Harris and Stephens 1998). Feature detection may also be performed by means of the method proposed in (Shi and Tomasi 1994). The

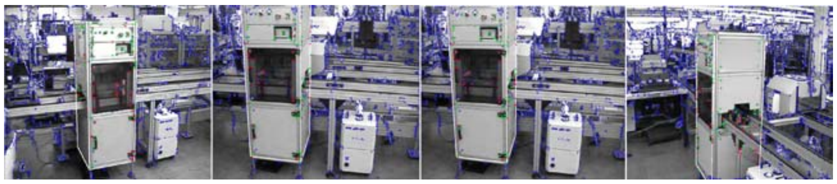

Fig. 9. This sequence has been tracked using a very simple model of one of the objects in the environment (four frames of the sequence).

Fig. 10. This picture shows six frames of the tracked sequence of the corridor (700 frames).

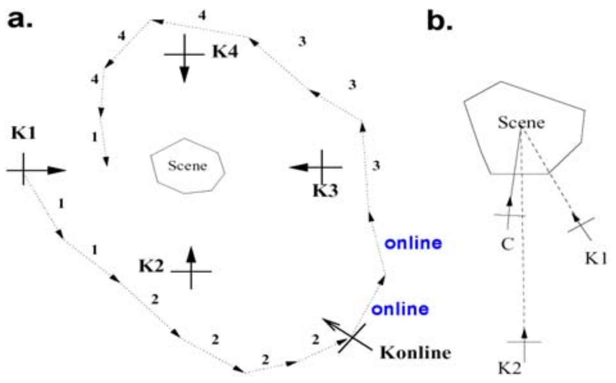

Fig. 11. Online and Offline Keyframes. a. Tracked camera displacement with four offline keyframes and one online keyframe. The dotted arrows represent the camera displacement from one frame to the next, and the number shows which keyframe is being used. K1 to K4 are the camera positions of the offline keyframes. When the current camera position gets too far from any known offline keyframe, a new online keyframe denoted Konline is generated. b. Choosing the best keyframe between *K1* and *K2*. *C'* is the previous camera position.

strongest points are the image areas having color intensity significantly different from the neighborhood: we show the interest points detected in real images in Fig. 5. Let the interest points detected at the time *t* be:

$$m_t = \left\{ m_t^{\,0} \ldots m_t^{\,n} \right\} \tag{1}$$

Given a previous frame, let m_{t-1} be the set of 2D points that we detected in it and M_{t-1} be their 3D position. Assuming that the rotation and translation parameters vector $\left[R\,|\,T\right]$ is known in the previous frame, but new parts of the object may have appeared, we want to take into account the new 2D interest points. So we back-project them in order to find their 3D coordinates M_{t-1}, keeping only the interest points that are on the object surface and discarding all the others. To do so, we first use a "Facet-ID" image to detect on which face of the 3D model each 2D point lies. Some examples of this kind of image are shown in Fig. 8. That image is generated by encoding the index *i* of each facet f_i as a unique color, and projecting the whole model into the image plane, using a standard OpenGL rendering. Once the facet-ID is known, we find the intersection with the found facet and the line passing through the camera centre of projection and the 2D point in the image plane. To compute this intersection we cast a ray from the center of the camera $Copt$, passing through the 2D interest point on the image *m*, and then intersecting the object model in the point *M* (see Fig. 7).

Given the two matrices of intrinsic (*A*) and extrinsic ($\left[R\,|\,T\right]$) parameters that define the projection matrix as:

$$P = A\left[R\,|\,T\right] \tag{2}$$

We can compute the $\vec{r}$ vector, which gives us the direction of the ray. It can be expressed as:

$$\vec{r} = \left(AR\right)^{-1} m \tag{3}$$

where:

$$m = \left(u\ v\ 1\right)^{T} \tag{4}$$

The optical center $Copt$ (the origin of the ray) can be computed using:

$$Copt = -R^{T}T \tag{5}$$

The intersection with the model, in the case of a triangle mesh, can be computed by means of the efficient algorithm presented in (Moeller and Trumbore 1997), however it can be any kind of line to geometric shape intersection algorithm.

The algorithm in (Moeller and Trumbore 1997) is a very efficient computer graphics algorithm that computes the intersection between a triangle mesh and a ray. It can be extended to high polygon number meshes allowing tracking also complex objects without slowing down the tracker. Being the 3D position M_{t-1} in that frame known, we have two sets of points, respectively 2D and 3D:

$$m_{t-1} = \left\{ m_{t-1}^{0} \ldots m_{t-1}^{n} \right\} \qquad (6)$$
$$M_{t-1} = \left\{ M_{t-1}^{0} \cdots M_{t-1}^{n} \right\}$$

such that:

$$m_{t-1}^{i} = A\left[R_{t-1} \mid T_{t-1} \right] M_{t-1}^{i} \qquad (7)$$

where M_{t-1} and R_{t-1} and T_{t-1}, the camera rotation and translation estimated for the previous frame, are expressed in the object coordinate system. *A* is the internal parameters matrix. We are looking for the R_t and T_t matrices for the current frame. We match the 2D points between m_{t-1} and m_t, choosing for each point in the set m_{t-1} the one in the set m_t that maximizes a correlation measure that is insensitive to illumination changes (Hartley and Zisserman 2000).
As a result, most of the current image points m_t are matched to the previous image points m_{t-1}:

$$m_t^{j} \leftrightarrow m_{t-1}^{i} \qquad (8)$$

Since m_{t-1}^{i} must re-project on m_t^{j} we should have:

$$A\left[R \mid T_t \right] M_{t-1}^{i} = m_t^{j} \qquad (9)$$

Therefore also the 3D points belonging to M_t^{j} can be associated to the 3D points of M_{t-1}^{i}, giving in this way the 3D coordinates of the unknown points:

$$M_t^{j} = M_{t-1}^{i} \qquad (10)$$

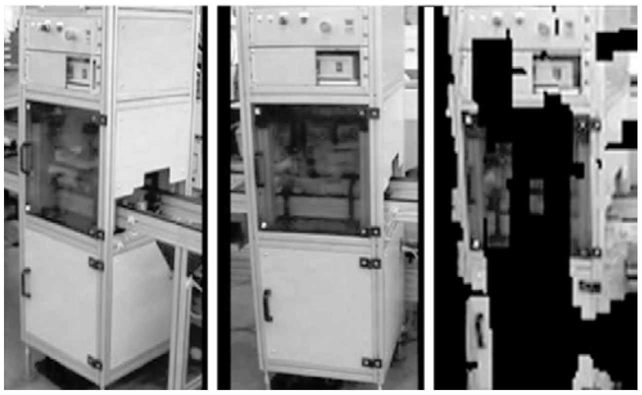

Fig. 12. From the left: the first image has to be matched with the second one, but the two images are not close to each other. The point patches of the first image will be skewed as shown in the third image to make the matching possible.

The 3D points are the same for both the images if the 2D points have been correctly matched. Once all the 2D-3D correspondences are done, we have enough information to compute the camera position in the object reference system. This is done using the algorithm proposed in (Dementhon and Davis 1992) and the robust estimator RANSAC to discard outlier matches (Hartley and Zisserman 2000). Using RANSAC in our case means that we consider several small sets of four random points to compute a temporary pose, find a displacement $\left[R \mid T\right]$ and re-project the whole set of points using this $\left[R \mid T\right]$. Only the configuration that re-projects the most of points will be accepted. This algorithm is efficient to consider a large number of sets of four points and it does this robust estimation in a reasonable time, which means we can do three hundred iterations in around ten milliseconds. In this way we can detect only the correct points, and discard all the outliers.

4.3 Keyframe based tracking

In short, the simple method presented in the previous part works with very good precision without jittering. However, the simple recursive approach is too weak from the point of view of error accumulation, and it is not suitable for a real-time environment. Thus, one must improve the method with some additional information. This can be done using some prior knowledge, supplied by the keyframes. This section explains how to use keyframes in order to track any sequence with no drift and no limits on the camera position. During the training stage the user creates offline keyframes, and in the tracking stage the previous information is used to track. During the tracking the camera may move too far away from any known keyframe: In that case a new “online” keyframe is added to the others and it will be re-used when the camera position passes close to it a second time.

During the offline stage, the user is asked to choose a set of images representing the scene from many different points of view or, at least, the positions that the camera will probably reach. Usually it is enough to record a video sequence all around the object and to choose a frame for some camera displacement. While tracking the sequences presented in this paper we only used 4 keyframes. Complex aspect changes such as 360 degree rotations and 6 degrees of freedom may need from 10 to 20 keyframes.

After the reference image choice the user is asked to accurately calculate the $[R \mid T]$ for each image. There are many methods to calculate the $[R \mid T]$. In our early test stage we were using a tool built by us using the method described in (Dementhon and Davis 1992): it is enough to get the 2D position of 4 known points in every image to calculate the objects pose. The user can even make use of commercial post-production tools, such as the ones of RealViz™ or 2D3™. The commercial products can retrieve the object position over the whole sequence with good accuracy, since they work offline.

When $[R \mid T]$ is known for every keyframe, the user has completed the offline stage. Then the system performs interest point detection and back-projects the points that lie on the object surface to compute their 3D positions.

4.4 Visibility criterion for keyframe choice

The first step of the tracking algorithm is to choose the best keyframe. This choice is a critical task on which the quality of the matching depends. An aspect of a keyframe must be as close as possible to the current frame. As shown in Fig. 11.b, simply evaluating the camera position is not enough. The point C represents the current camera position, and K1 and K2 are two keyframes. Just taking the keyframe that minimizes the euclidian distance means that the closest keyframe is K1. However its aspect is not as close as K2, which is further away but has a closer line of sight. To correct this problem, we should evaluate the angle between the two lines of sight. However, this is still not a complete method, because it does not take into account object non convexities and self occlusions. Instead, we use an appearance-based method. We use the following criteria:

$$\sum_{\forall f \in Model} \Big(Area\big(f, A_p[R_P \mid T_P]\big) - Area\big(f, A_K[R_K \mid T_K]\big) \Big)^2 \tag{11}$$

where $Area(f, P)$ is the 2D area of the facet *f* after projection by *P*. We reuse the method we introduced in the previous section for an accelerated OpenGL rendering of the object model. Every facet is rendered in a different color, representing the facet index, using the camera R and T estimated for the previous frame. We histogram this “Facet-ID” image and compare the result to the keyframe histograms, which have been created offline during the learning stage. We get the contribution of the area of every single facet in the model as it is reprojected in the 2D image. Every histogram bar represents the number of occurrences of every facet's pixels. This method has constant complexity, and requires only a single read of the image.

4.5 Wide baseline matching

This section presents our method to handle the perspective distortion on the correlation window. Conventional methods make use of a square bi-dimensional correlation window. This technique gives good points matching under the assumption of very small perspective distortion between two frames. However, to effectively use keyframes, the ability to match distant frames becomes essential. Consequently we specify a point matching algorithm between a square 2D window in the current frame and a perspective distorted window in the keyframe image, that we call the “re-rendered"” image. We skew the 30x30 pixel patches around each interest point from the keyframe image in order to bring them to a position close to the current one. Each patch in the keyframe is related to the corresponding image points in the “re-rendered image” by a planar homography.

Given the patch corresponding to the plane π having coordinates $\pi = (\vec{n}^T, d)$ so that for points on the plane $\vec{n}^T X + d = 0$, the general expression for the homography induced by the plane is (according to (Hartley and Zisserman 2000)):

$$H = A'\left(R - t\vec{n}^T / d\right)A^{-1} \tag{12}$$

between two different views defined by their projection matrices $P = A[I \mid 0]$ and $P = A'[R \mid t]$. The homography equation for the general case can easily be obtained by changing the reference system; we get:

$$H = A_K\left(\delta R - \delta t\vec{n}'^T / d'\right)A^{-1}{}_P \tag{13}$$

$$\delta R = R_P R_K{}^T \ ; \ \delta t = -R_P R_K{}^T t_K + t_P \ ; \qquad \vec{n}' = R_K \vec{n} \ ; \ d' = d - t_K{}^T\left(R_K \vec{n}\right) \tag{14}$$

where $A_K[R_K \mid t_K]$ and $A_P[R_P \mid t_P]$ are the projection matrices of the keyframe and the previous frame. The resulting image is a re-rendering of the interest points' neighborhood in a more convenient position as shown in Fig. 12.

This method allows us to effectively match views even where there is as much as 60 degrees of rotation. An alternative solution to the homography would be to re-render a 3D representation of the object using an OpenGL textured 3D object, but we choose the other way to have a better result around the points.

4.6 Offline and online keyframes

Assuming we already have a consistent set of keyframes, in this subsection we show how to employ them to track a sequence. As shown in Fig. 11.a, while the camera moves around the scene, the system switches from one keyframe to the other, always choosing the one that is closest to the current image.

When the current camera position gets too far from any known offline keyframe, a new online keyframe denoted Konline is generated. It will be added to the keyframe set and treated like the other ones. The criterion we use for deciding to generate an online keyframe is the minimum number of inliers (correctly matched points): when they are less than 15-30 (depending on the type of object) we switch to a new keyframe. After some time the camera will again pass close to a known position, re-using the keyframes that have been generated online. If the sequence is difficult the system needs more offline keyframes.

An interesting characteristic of this method is that when some error has been accumulated over a part of the sequence, it will be reset to zero when an offline frame is used. The online frames can be considered as a kind of “second chance” method used to recover when there are no offline keyframes, and it has only to guarantee no complete divergence before the camera gets close to an offline frame.

5 Experiments and results

The whole system runs close to real-time. Due to the time-consuming algorithms we are obliged to use two separate 2.5GHz PC machines, one for tracking and another for VHD++ rendering respectively. In this configuration the tracking yields from 15 to 25 fps depending on the object size, while rendering goes above 30 fps. Given the hardware configuration the current tracking performance can be achieved with 320x240 images. Once more CPU power is available, we will use bigger images and the tracking quality will improve.

Aiming at the tracking performance as the most critical, we have also built many tests by means of the tracking algorithm but using only a light rendering part. In this case our system can run on a single 2.6 GHz laptop machine using an IEEE 1394 camera.

We tested the tracking algorithm in a factory environment, where we succeeded to track pre existing equipment and to superpose virtual parts. We also use the same algorithm for tracking and augmenting a variety of 3D objects (e.g. tea boxes, small toys). The same method is used for face tracking and augmentation in

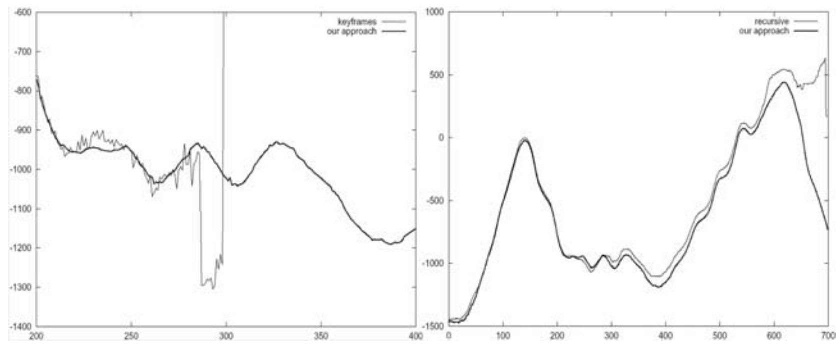

Fig. 13. Plots showing a 700 frame sequence tracked using three different methods. The first plot shows the jitter of keyframe method compared to our method, the second one shows the error accumulation of the recursive method compared to ours. In both plots, the bold line corresponds to our results. We use our result as a ground truth being visually correct when we re-project the model.

real-time; the executable of this algorithm is available for both IEEE1394 Point Grey Dragonfly camera or web cam at the link:

http://cvlab.epfl.ch/software/download.html

All our demonstrations make use of small portable cameras that can be installed on a helmet. The visualization is screen display, which solves all the problems of tracking delay.

Being our camera lens of good quality we have never addressed the radial distortion problem; however it can be easily solved by means of the most common camera calibration programs (e.g. Australis, Intel OpenCV Calib Filter).

Examples of tracked objects and scenes are shown in the Fig. 1 and 2. Fig. 1 shows a virtual human showing the use of a real machine in a factory. In Fig. 2 we show the result of a sequence in which the camera is moved through the corridor and is turning around the corner, and an avatar is walking in front of the user, showing the way to follow.

In order to have a more qualitative analysis of our method, we made it work first in some limited conditions that are closer to the more conventional approaches, than using the complete algorithm. So we used a feature matching approach to track a scene three different times:

- using only chained transformations, like a recursive tracker,
- using only keyframes (offline information),
- combining both using our proposed method.

The result of the combined method is used as ground truth, since it works with good accuracy for the whole sequence. We verified this by re-projecting the model on the images. Fig. 13 depicts the evolution of one of the camera center coordinates with respect to the frame index. The first plot compares our method to the keyframes-only method. The keyframe-only method suffers from jitter and fails at frame 295 because there is no available keyframe providing enough point matches for that point of view. The second plot compares the recursive method to ours. Error accumulation in the recursive method does not corrupt the tracking immediately, but eventually also provokes tracking failure around frame 550.

At the end, when the tracking has been correctly performed, the rest of the work is done by VHD++. The rendering module provides the main support for video image synthesis of virtual humans-objects correctly registered to the real video image. Thus we have implemented successfully the integration of the tracking and the rendering module (currently on the same vhdRuntimeEngine), realizing the architecture depicted in Fig. 4.

6. Conclusion

In this paper we presented a novel framework for building real-time VR/AR applications. Our system has not only been tested in a laboratory, but also in a real environment. The focus of this work is to build applications for training and planning in industrial environment, but it can be used as well for many other tasks, such as ergonomics, repair, prototyping, emergency situations training, and all the cases where a dangerous or too expensive to create situation is present and it cannot be reproduced by a real scene.

The tracker embedded in the framework is based on natural feature points, and it can be used with a large set of scenes. The model information is exploited to track every aspect of a given target object even when occluded or only partially visible, or when the camera turns around the scene. The result is a practical system that we have been able to test in a real factory environment.

Bibliography

Azarbayejani and Pentland 1995 A Azarbayejani and A P Pentland Recursive Estimation of Motion, Structure and Focal Length. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 17, no. 6, pp. 562–575, 1995.

Azuma 1995 R Azuma A survey of augmented reality. Computer Graphics (SIGGRAPH'95 Proceedings), August 1995, pp. 1–8.

Beardsley et al. 1997 P A Beardsley, A Zisserman, and D W Murray Sequential update of projective and affine structure from motion. International Journal of Computer Vision, vol. 23, no. 3, pp. 235–259, 1997.

Dementhon and Davis 1992 D DeMenthon and L S Davis Model-based object pose in 25 lines of code. European Conference on Computer Vision, 1992, pp. 335–343.

Drummond and Cipolla 2000 T Drummond and R Cipolla Real-time tracking of multiple articulated structures in multiple views. ECCV (2), 2000, pp. 20–36.

Fitzgibbon and Zisserman 1998 A Fitzgibbon and A Zisserman Automatic Camera Recovery for Closed or Open Image Sequences. European Conference on Computer Vision, Freiburg, Germany, June 1998, pp. 311–326.

Harris and Stephens 1998 CG Harris and MJ Stephens A combined corner and edge detector. Fourth Alvey Vision Conference, Manchester, 1998.

Hartley and Zisserman 2000. R. Hartley and A. Zisserman, Multiple View Geometry in Computer Vision,. Cambridge University Press, 2000.

Kato et al. 2000 H Kato, M Billinghurst, I Poupyrev, K Imamoto, and K Tachibana Virtual object manipulation on a table-top AR environment. Proceedings of International Symposium on Augmented Reality, 2000.

La Cascia et al. 1999 M La Cascia, S Sclaroff, and V Athitsos Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3d models. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 4, April 2000.

Marchand et al. 1999 E Marchand, P Bouthemy, F Chaumette, and V Moreau. Robust real-time visual tracking using a 2d-3d model-based approach. IEEE International Conference on Computer Vision, ICCV'99 (1), Sept. 1999, pp. 262–268.

Moeller and Trumbore 1997 T Moeller and B Trumbore Fast, minimum storage ray triangle intersection. Journal of graphics tools, 2(1):21-28, 1997.

Neumann and You 1999 U Neumann and S You Natural feature tracking for augmented reality. IEEE Transactions on Multimedia, vol. 1, no. 1, pp. 53–64, 1999.

Ponder et al. 2003 M Ponder, G Papagiannakis, T Molet, N Magnenat-Thalmann, D Thalmann, VHD++ Development Framework: Towards Extendible, Component Based VR/AR Simulation Engine Featuring Advanced Virtual Character Technologies. Computer Graphics International (CGI) 2003, to appear.

Shi and Tomasi 1994 J Shi and C Tomasi. Good Features to Track. IEEE Conference on Computer Vision and Pattern Recognition, pages 593-600, 1994.

Simon et al. 2000 G Simon, A Fitzgibbon, and A Zisserman Markerless tracking using planar structures in the scene. Proc. International Symposium on Augmented Reality, October 2000, pp. 120–128.