PREPRINT. To appear in Proceedings 21th Symposium on Information Theory

Latency layered rendering for mobile augmented reality

Wouter Pasman[1] and Frederik W. Jansen[1]

[1] Delft University of Technology, Department of Information Systems and Technology, Mekelweg 4, 2628 CD, Delft, Netherlands
{w.pasman, f.w.jansen}@twi.tudelft.nl
http://www.ubicom.tudelft.nl/

Abstract. Augmented reality requires accurate alignment of virtual objects with objects in the real world. This requires extremely low end-to-end latency from tracking to display. For mobile use these requirements are even more difficult to achieve when one considers the unrestricted situation of a user walking around and using a wireless, wearable unit that should be small and have low power consumption. To meet these conflicting requirements, we propose a latency-layered system that combines fast tracking and rendering techniques, using approximate geometric models, with slower but more accurate techniques. In this way very fast viewpoint estimates and image updates can be achieved, followed by slightly slower but more correct viewpoint estimates and image updates. We present a model to trade-off image accuracy and model complexity with respect to the available rendering power and memory at the mobile unit and the cost of transferring information over a wireless link from a computer station to the mobile unit.

1 Introduction

With augmented reality, virtual objects can be projected in overlay with the real world. For applications such as remote maintenance, the placement of these virtual objects is quite critical and should be accurate within an order of tenths of a degree. In addition, if we want keep the objects at a fixed position with respect to the real world while the viewpoint is changing, then the system should also meet strict latency requirements, i.e. render a new image within 10 ms. These constraints are even more difficult to achieve when we use augmented reality in a mobile situation where no accurate tracking is possible and the wireless link itself has a considerable latency. At the same time the mobile system should be lightweight and low power.

To meet these clashing requirements, we propose a latency-layered system that combines fast tracking and rendering for first approximate estimates, and slower techniques to achieve better estimates. For the position tracking, fast viewpoint estimates are made with an inertial tracker. The estimation is kept accurate with a lower frequency optical tracking system. For the rendering we apply a viewpoint correction four times within the rendering and display of a frame, keeping the latency for individual scan-lines to less than 10 ms. This allows us to render a few hundred textured-mapped polygons within that time span. In order to render more complex

scenes, we apply several stages of scene simplification to the model, from general simplification to viewpoint dependent simplification and the use of image imposters.

In this paper we describe our UbiCom-system for Ubiquitous Communication and how the mobile unit carried by the user works together with a compute station at a remote location over a wireless link. Then we describe how we meet the latency requirements with our layered tracking and rendering system. We present an analysis of image-based rendering and scene simplification techniques and we define a model to calculate the trade-offs between image accuracy, rendering complexity, and the load on the mobile link.

2 Overview of the system

The UbiCom system is an infrastructure for mobile multi-media communication, developed at Delft University of Technology as a cooperative efford of several groups in radio communication, image compression, graphics and distributed processing [1]. Users will wear a headset with a stereo augmented reality display connected to a small wearable transceiver (the mobile unit). The transceiver is connected via a wireless link to a nearby base station. Base stations are connected to other base stations and a compute station via a backbone. The compute station offers cheap compute power, storage space for large databases, and connections to the internet (Figure 1). The mobile unit tries to off-load computations to the server to save the scarce resources of the mobile unit, carefully trading off compute power for load on the wireless link.

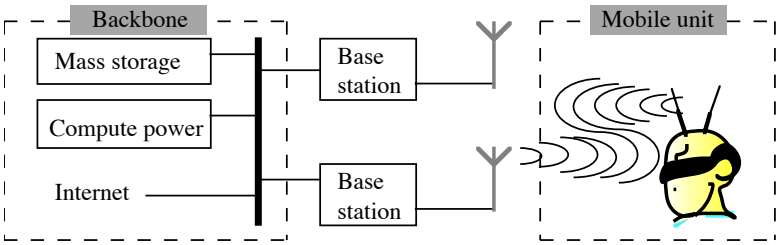


Fig. 1. Overview of the UbiCom system. See text

The configuration of the headset is shown in more detail in Figure 2. It contains an optical see-through stereo display, an inertial tracker, two video cameras and a sender and receiver for the wireless link. Dedicated hardware is available for 3D rendering, image compression and decompression, and for tracking features in the camera images. The wireless communication system uses a 17 GHz radio link outdoors and infrared indoors, and is described in more detail in [2].

The compute station has a database with a detailed 3D model of the environment and an associated set of most important features as input for the vision positioning system.

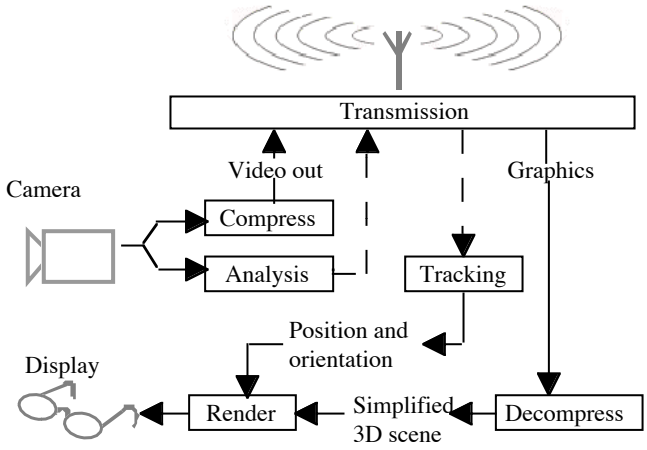
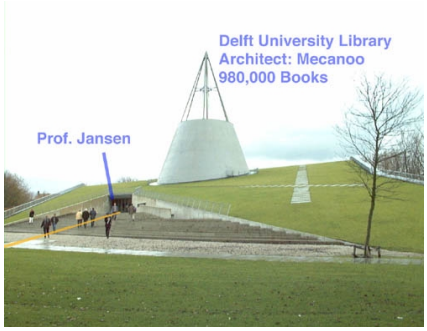


Fig. 2. The mobile unit.

Typical applications we have in mind are way finding (Figure 3a) and remote maintenance (Figure 3b). Applications may also include person-to-person communication, information services for 'info-on-the-spot', games and personal assistants. We don't yet have a "killer app", but we anticipate that video and 3D animation will play a key role.



(a)

(b)

Fig 3. Two sample applications for the UbiCom system.

3 Latency-layered tracking and rendering system

It turns out that latency is the major constraint on the accuracy with which virtual objects can be placed in an environment [3, 4, 5, 6] (Figure 4). Humans can rotate their heads very fast (360˚/s [7] up to 2000˚/s [8]), and can easily detect errors of 0.25˚. Combining these extreme values leads to a latency requirement of 0.1 to 0.7 ms, several orders of magnitude under the current latencies of 3D systems. But we expect that humans will not notice small alignment errors while moving fast, and moreover that small alignment errors during motion will not hinder the task at hand.

For the applications we have in mind we think that a maximum alignment error of 0.5˚ at a head rotation speed of 50˚/s is acceptable, leading to a latency requirement of 10 ms. This is similar to the suggestions of other authors [9, 4, 10].

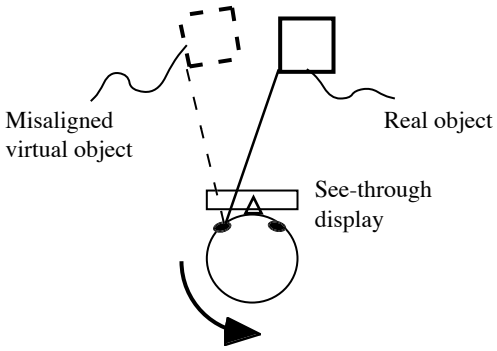


Fig. 4. Latency-alignment relation. In this example we try to align a virtual object with a real object, while the observer is rotating his head to the left. The virtual object has a latency and therefore is displayed in the direction it should have been a short while ago. Therefore the real and virtual object are not aligned anymore. The amount of error depends on the head rotation speed and the latency.

In order to meet the latency requirement of 10 ms, we propose a latency layered structure. This latency-layered structure can be made for the two essential components in the critical path: the tracking and the rendering system.

The tracking system is primarily based on a vision system. A camera on the headset captures images which are analysed for line features. These features are sent –together with GPS data– to the backbone to be matched with a pre-selected set of features from the data base. After the feature identification, the new view position estimate is calculated and send back to the mobile unit, where the new viewpoint estimate arrives about hundred milliseconds after the initial feature image was captured. Within this time frame the movements and rotations of the headset are sensed by inertial trackers and by integrating the measured acceleration an up-to-date viewpoint position is calculated every few milliseconds. The orientation is tracked at the same rate with gyroscopes. As the integration accumulates errors, the headset position is corrected when a new accurate estimate arrives from the vision system. See figure 5.

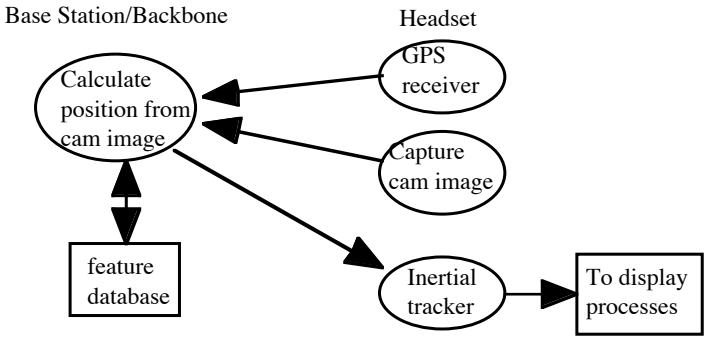


Fig. 5. Low-latency tracking system.

For the rendering system, the latency requirements are extremely severe. Table 1 shows the usual performance of the different stages of the viewing pipeline.

Table 1. Typical latencies in the graphics pipeline. See also [6].

Lag type	Description	Typical lag
Tracker lag	internal computations inside the tracker, physical delays	5-20 ms
Interface lag	transmitting tracker data, OS scheduling delays	10-30 ms
Image-generation lag	rendering time	25-200 ms
Video sync lag	lag while waiting for next video frame	10 ms
Display lag	Delay from frame swap until actual display of pixel	0-20 ms
Internal display lag	some displays don't refresh all visible pixels immediately	0-40 ms

The actual display of a rendered image already takes about 20 ms from the first scanline at the top of the image to the last scanline at the bottom. To reduce this part of the total latency, we divide the image into a number of segments and render just ahead of the raster beam (Figure 6). When a quarter of the image is displayed, the next segment is rendered. With standard graphics hardware we then can achieve a latency of 10 ms (2 ms tracking, 4 ms rendering and 4 ms display) with a rendering load of a few hundred texture mapped polygons
(see also [11]).

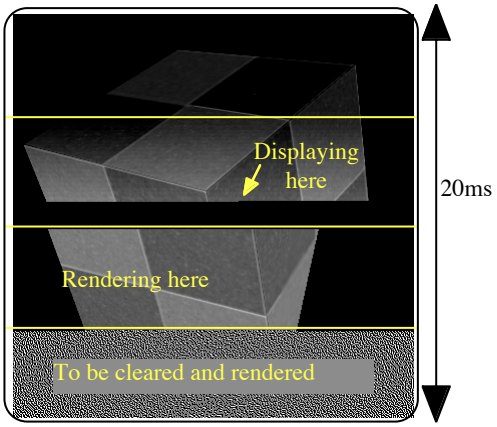


Fig. 6. Overlapped rendering and display with four image segments.

To render scenes with a larger number of polygons we simplify the objects to fit within the available 'budget'. First, complex objects are simplified to a small number of polygons with textures in the compute station. This is done considering the expected near-future viewpoint of the user. The simplification may go as far as to replace complete objects by one image. The resulting scene description is sent to the mobile unit, where it is rendered using the actual position of the user (Figure 7).

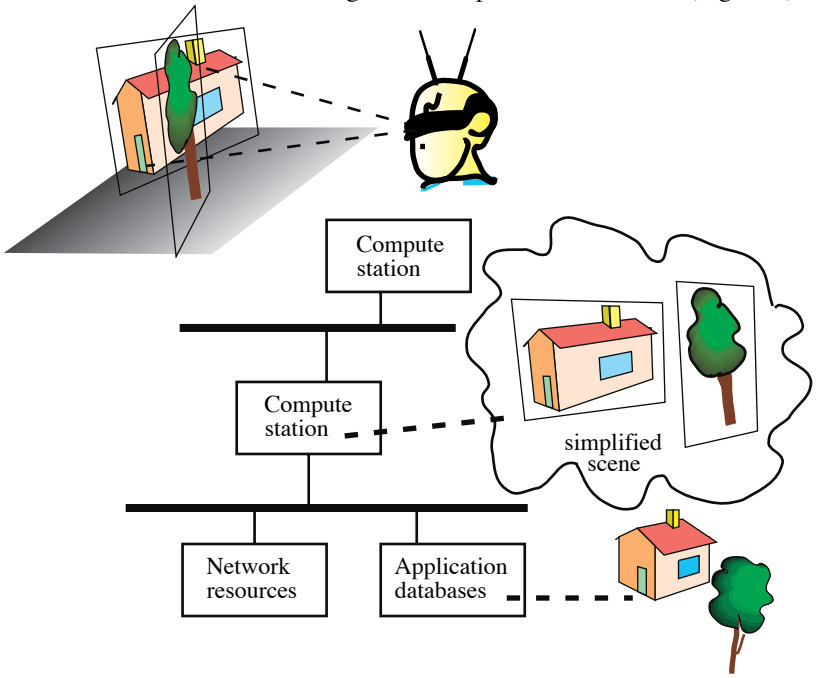


Fig. 7. overview of the rendering system.

As the image imposters have only a short life time (only accurate near the initial viewpoint), new imposter updates will have to be generated at the compute station and sent to the mobile unit. Together with the latency-layered tracking, this gives us the following latency-layered structure (Figure 8). The image is updated each 10 ms after a user moves, but with only an approximate image. More accurate updates (over the mobile link) are available only after 100 ms and 1 second.

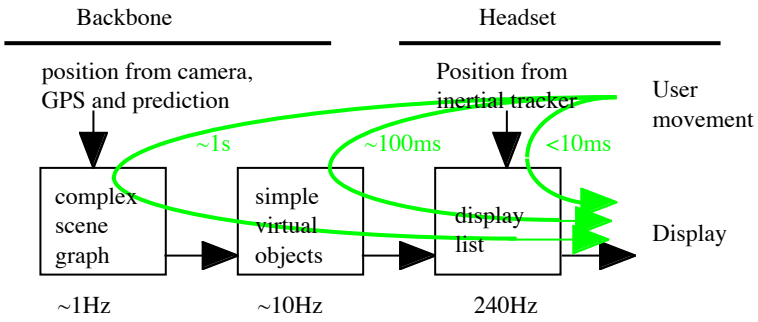


Fig. 8. Latency-layered structure of the system

4 Scene representation within the mobile unit

As discussed above, the mobile unit will hold only simplified versions of the scene. A number of possibilities are open here. We consider simplified polygon objects, meshed imposters, images with depth, and simple imposters. See Figure 9.

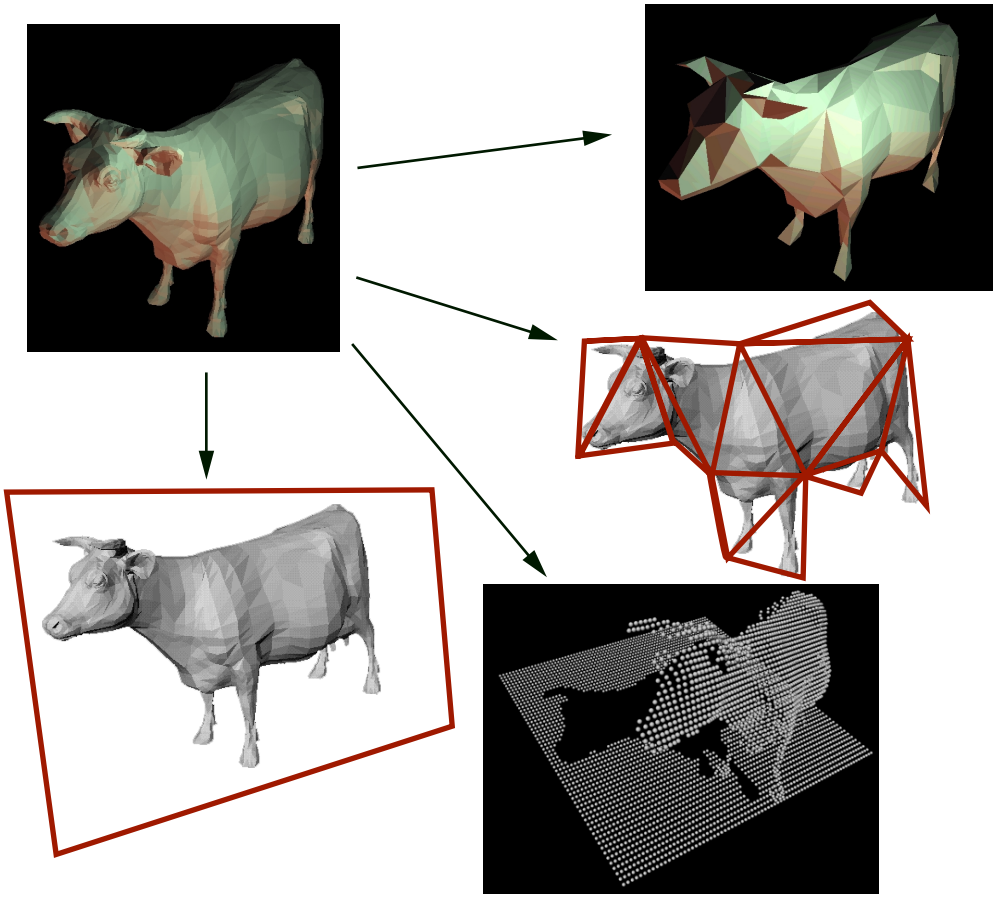


Fig. 9. Several simplifications of the cow model: in clockwise direction a simplified polygon object, meshed imposter, image with depth, and simple imposter.

Simplified polygon models seem a good choice to replace nearby objects, because the observer can move around them freely without requiring the system to refresh the model. But for distant objects this is overly accurate, as the observer will probably never see the back faces.
Another option is to use simple imposters. For nearby objects this is not a good choice, as the texture on the imposter would have to be updated very often because the poster is only correct for a small region around the initial viewpoint. But for distant objects this seems a good choice.
The meshed imposter and image with depth seem suited for the in-between cases: they are more accurate than simple imposters and therefore can be used longer, but they still have no back faces.

This analysis suggests the configuration of Figure 10: nearby objects are represented with simplified polygon models, more distant objects with images with depth or meshed imposters, and far away objects with simple imposters. The switching

distances may be closer for objects outside the viewing cone of the observer, to avoid too much effort being put in rendering things the user is not looking at.

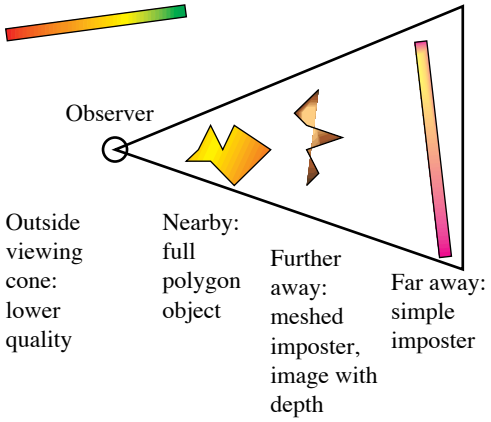


Fig. 10. Intuitive way of using various objects with varying distance.

However, for deciding which representations to use for which objects in order to stay within the limit of a few hundred polygons/imposters, we need a model that tells us how to trade off accuracy against the memory and rendering capacity at the mobile unit and the latency of the mobile link. We did a thorough analysis of the various techniques, to estimate the accuracy as a function of the number of polygons and the load they impose on the wireless link. This model is too large to discuss here, instead we sketch our approach (Figure 11).

We started with estimating the distortion (D) in the images rendered from a model given the distance to the object (d), the size of the object (r), the number of polygons available (N), the desired life time of the object (T) and the number of pixels in the texture for the object (Tex). The texture compression ratio can also be estimated from the texture refresh pattern: if the model needs very frequent updates we can reach MPEG-like ratios, while less frequent updates will reach lower, JPEG-like ratios.

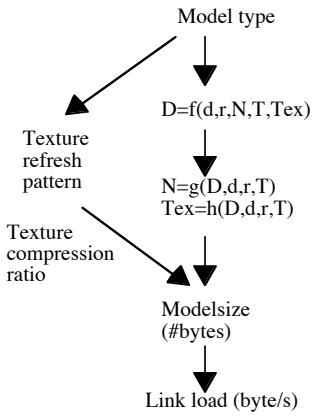


Fig. 11. Sketch of our resource usage analysis of various choices for model representation in the mobile unit.

The next step involves inversion of the function D: we set a maximum distortion and derive the number of polygons (N) and texture pixels (Tex) as a function of the deviation from the initial viewpoint (which follows from the life time T). To come to an estimation of the model size, we assume a constant number of bytes for each polygon and divide the number of texture bytes by the expected compression ratio (Figure 12). We can see that the imposters are very close in size, and that the polygon model is much larger but does not grow as fast with decreasing viewing distance. All models break down at some point: at closer distances the model cannot reach the distortion and life time requirements anymore. The simple imposter breaks down earlier than the other models, because it is very much confined to the initial viewpoint. The polygon model also breaks down at some point, because we have a limited amount of polygons and distortions will become evident anyway if the observer can come close to the object within the life time of the object. In this case, a near clip distance will have to be defined to avoid excessive model refresh.

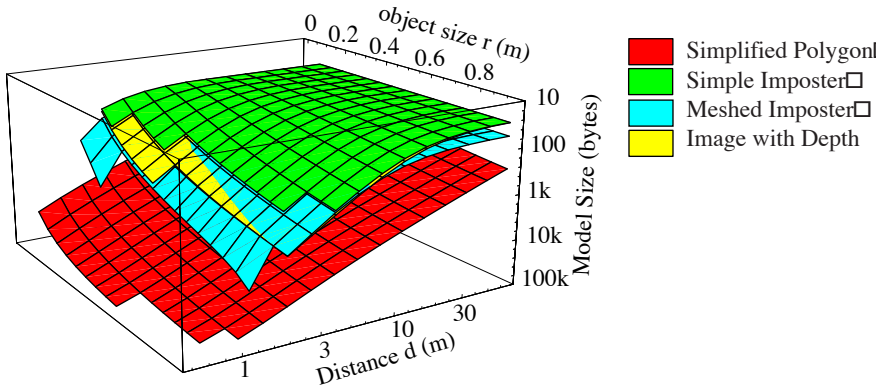


Fig. 12. Model size as a function of the distance to the object and the size of the object. Lifetime was set to .1 second, the maximum distortion D=0.005 rad and the polygon budget N=1000. The model size is small because of the light distortion requirement in this example.

As the model size is a function of the chosen life time (refresh rate), we can find the average link load by dividing the model size by its life time (Figure 13). We also accounted for the probability that the user walks in such a direction that model refresh is required. The meshed imposter has a slightly longer lifetime at a given maximum distortion than a simple imposter, but essentially only a polygon model can reach the distortion requirements at close distances. The wireless link will not form a bottleneck, but the expected latency of 100 ms (Figure 8) strongly limits the use of nearby imposters because, with realistic imposter life times of less than 100 ms, the link can not provide fast enough updates from the compute station.

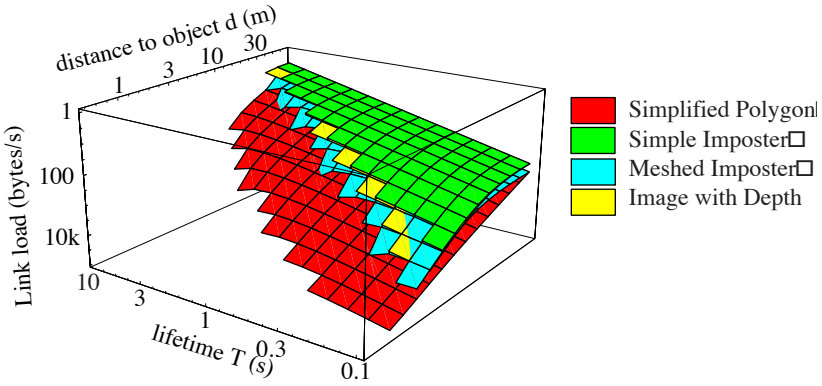


Fig. 13. Load on the wireless link as a function of distance to the object and the lifetime. Settings as in Figure 12, the object's size r=1 meter.

The results of our estimation of the load on memory and CPU is straightforward: as the number of polygons in our system will be much smaller than the number of pixels to be put to the screen, the main CPU and memory load follows from the amount of pixels in texture memory and on the screen. Therefore, the main load comes from the way these pixels are stored (are they RGB, RGBA or RGBA + depth), and the amount of per-pixel calculations (images with depth require additional per-pixel warps and splat kernel calculations). Simple imposters, meshed imposters and polygon models all impose very similar loads. For images with depth this depends on the rendering method, ranging from a 33% higher CPU load for forward splatting [12] to a 60% higher CPU load plus a 100% higher memory load for a hybrid mapping [13].
Concluding, the intuitive sketch of Figure 10 seems confirmed by our theoretical model, although the exact boundaries and parameter settings require some refinement of the model and some experimental testing.

5 Conclusions

We discussed our latency-layered approach to meet the high requirements for mobile augmented reality, and we sketched an approach for deciding which model representations are appropriate for the mobile unit. The presented analysis suggests that simple imposters are appropriate for distant objects, meshed imposters can extend their life slightly, and that polygon models are required for nearby objects.
Our near-future work will be the integration of this approach into our demonstrator system (see figure 14). Longer term research will be on scene simplification issues related to incremental texture updating, and its close relation to the video compression and decompression system.

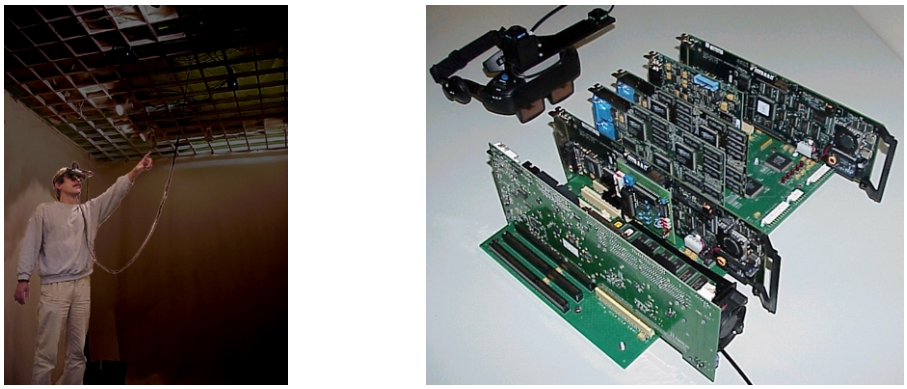


Fig. 14. Prototype system.

References

[1] Deprettere, E. (1999). Ubiquitous Communications:Aiming at a new generation systems and applications for personal communication. Available Internet: http://www.ubicom.tudelft.nl/.

[2] Pouwelse, J., Langendoen, K., & Sips, H. (1999). A Feasible Low-Power Augmented-Reality Terminal. 2nd IEEE and ACM Int. Workshop on Augmented Reality (San Francisco), 55-63.

[3] Azuma, R. T. (1997). Registration errors in augmented reality. Available Internet: http://epsilon.cs.unc.edu/~azuma/azuma_AR.html.

[4] Ellis, S. R., & Adelstein, B. D. (1997). Visual performance and fatigue in see-through head-mounted displays. Available Internet: http://duchamp.arc.nasa.gov/research/seethru_summary.html.

[5] Olano, M., Cohen, J., Mine, M., & Bishop, G. (1995). Combatting rendering latency. Proceedings of the 1995 symposium on interactive 3D graphics (Monterey, CA, April 9-12), 19-24 and 204.

[6] Holloway, R. L. (1997). Registration error analysis for augmented reality. Presence, 6 (4), 413-432.

[7] List, U. (1983). Nonlinear prediction of head movements for helmet-mounted displays. U.S. Air force Human Resources Laboratory, Technical paper AFHRL-TP-83-45, December.

[8] Aliaga, D. G. (1997). Virtual objects in the real world. Commun. ACM 40, 3 (Mar), 49-54.

[9] Azuma, R. (1993). Tracking Requirements for Augmented Reality. Communications of the ACM, 36 (7): 50-51. Available Internet: www.cs.unc.edu/~azuma/cacm.html.

[10] Poot, H. J. G. de (1995). Monocular perception of motion in depth. Unpublished doctoral dissertation, Faculty of Biology, University of Utrecht, Utrecht, The Netherlands.

[11] Pasman, W., Schaaf, A. van der, Lagendijk, R. L., & Jansen, F. W. (1999). Accurate overlaying for mobile augmented reality. To appear in Computers & Graphics, 23 (6).

[12] Gortler, S. J., He, L., & Cohen, M. F. (1997). Microsoft technical report MSTR-TR-97-09. Available Internet: http://www.research.microsoft.com/~cohen.

[13] Shade, J., Gortler, S., He, L., & Szeliski, R. (1998). Layered depth images. Proceedings of the 25th annual conference on Computer Graphics (SIGGRAPH'98), 231- 242.