Distributed Low-latency Rendering for Mobile AR

W. Pasman, F. W. Jansen
Delft University of Technology, UbiCom project
Mekelweg 4, Delft, Netherlands
W.Pasman@its.tudelft.nl

Abstract

Wearable augmented reality (AR) can help the task of a user by adding virtual objects to his view on the real world. To save power in the mobile unit, rendering can be offloaded to the backbone as much as possible. However, because of low latency requirements, images for mobile AR can not be rendered completely in the backbone. We developed a system capable of end-to-end latencies of 10ms, with a seamless fitting dynamic level-of-detail framework extending the VRML and Inventor language, and building on current trends in QoS handling. In this paper we outline the structure and components of our system, and discuss a demo application projecting a statue on the campus.

1. Introduction

Mobile outdoor AR is technically extremely challenging. The wearable unit must be small and light, including a small battery, and is therefore limited in processing power and communication bandwidth. But at the same time the system has to render virtual objects with low latency between the user's movement and the corresponding update of the display (**end-to-end latency**), requiring both low-latency tracking and low-latency rendering.

The type of display the user wears has major consequences. Optical AR, where the virtual objects are merged optically into the real world via a half-transparent mirror, is preferred for mobile outdoor AR applications. The alternative, video AR, where the real world is recorded with a camera, and both virtual and real objects are shown on a monitor before the user's eyes, will degrade the user's view on the real world and usually avoids the end-to-end latency issue by delaying the real-world view as well. This will decrease the quality of his view on the real world, which may decrease his performance, may cause simulator sickness and might even be dangerous in case of a system failure.

When choosing for optical AR, a number of challenging issues have to be resolved:

- When using optical AR, the requirements for accurate positioning of virtual objects in the real world (**accuracy**) directly imply requirements for the maximum end-to-end latency. However, even high-end graphics workstations today are optimized for throughput and not for low latency.
- Virtual objects cannot just be rendered independently of the environment. If virtual objects have to be occluded properly by real-world objects, the rendering system needs to know about objects in the real world. Also, the system may need to be aware of other aspects of the environment, for instance for proper lighting of the virtual objects.
- Batteries are a point of concern for mobile AR applications. To save energy, it is essential to carefully trade off the battery usage against wireless network load and final image quality. A first approach could be to move computations to the fixed backbone systems. However, the results of the computations still have to be transferred back to the user, and if these results are large or latency-sensitive, the costs of the wireless link may be prohibitive.
- Current trackers are not suited and/or accurate enough to support outdoor AR. In indoor immersive VR, for which most current tracking systems have been designed, tracker errors cause no immediate problems because all visible objects have the same error. However with optical AR, tracker errors are directly visible. Fast and highly accurate outdoor trackers do not exist yet. The most suitable outdoor position tracking system is DGPS [1]. It has a positional accuracy of some centimeters which is sufficient for many applications, but it does not provide orientation information, has large latencies and has quite high power consumption.

Within the UbiCom project [2] an overall approach to mobile AR has been developed, including the radio, system software, protocol layers, low-power retinal scanning displays, outdoor high-accuracy tracking, robust video transmission and low-latency rendering. In this paper we focus on the graphics system. The paper is organised as follows. In section 2 we discuss related work. In section 3 we discuss the UbiCom Graphics subsystem. Next, we describe a demo application projecting a statue on the campus. Section 5 draws some conclusions.

2. Previous work

End-to-end latency is the major source of misplacement of virtual objects in the real world [3]. Especially rotation of the head can be very fast, in the order of 300 degrees per second and faster if necessary, and in such a situation a latency of 10 ms will already amount to a misplacement of

3 degrees. It has been shown experimentally that humans can detect latencies as low as 8 ms and probably less [4]. Fortunately, many tasks can be accomplished reasonably effective in presence of delays. For instance, for immersive VR, latencies in the order of 40 ms are acceptable, even for relatively latency sensitive tasks as car driving [5]. Configurations using optical AR however, are more latency sensitive than immersive VR configurations since real world objects have no latency and delays of virtual objects can be seen directly. For mobile AR tasks such as tourist information, remote maintenance and support, we estimate that 10 ms will be acceptable [6].

There are a number of components in the critical path amounting to the end-to-end latency of a mobile AR system: the tracker and its connection path to the graphics system, the graphics system and its frame buffers, and the display itself. Typical latencies amount to hundreds of milliseconds [3], and even most systems designed specifically for low latency still have in the order of 35-50 ms latency [7, 8].

The mobile system will be unable to render complex scenes given the latency requirements and limits on power usage. To lower the load on the mobile system, LOD rendering can be used. In this scenario, the full scene is available in the backbone, and the application is handling this full representation, but only a simplified version is available in the mobile system for rendering. There are a number of variants for LOD rendering (Figure 1): simple imposters [9], meshed imposters [10], images with depth [11], layered depth images [12] and simplified polygon models [13].

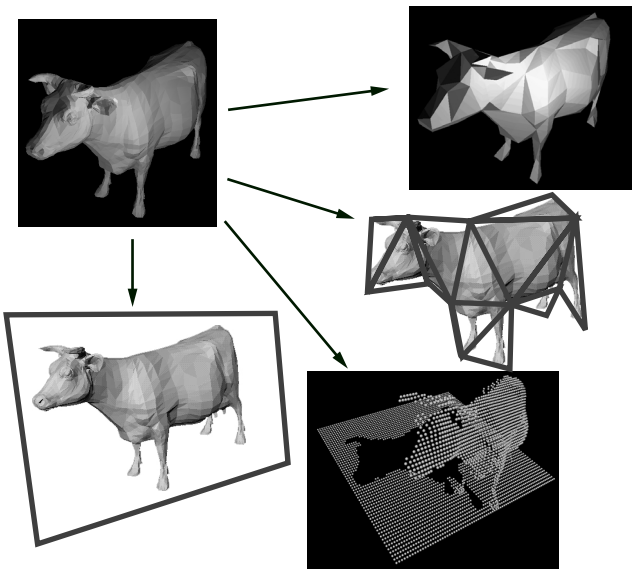

Figure 1. A number of scene simplification methods. From top left, in clockwise order: the original object, simplified polygon object, meshed imposter, image with depth and simple imposter.

In most cases the application programmer should not be bothered about all these optimizations and generation of numerous LODs. Especially for imposters, automatic generation and refresh is a prerequisite, and when the application is going to use third-party 3D objects, for instance from the web or from GIS databases, system support is required. A number of systems using automatically updated imposters have appeared in literature. Most of them support only simple imposters [14, 15]. The system of Decoret [16] supports meshed imposters.

A number of LOD frameworks have been developed that support an arbitrary object representation. Funkhouser and Sequin [17] select a combination of representations such that the overall cost/benefit ratio is maximised. The amount of polygons, pixels and vertices in the object are used to make an estimation of the rendering costs and benefits of the objects. Mason and Blake [18] improved this framework by adding support for hierarchical scene descriptions and clustering or grouping of objects. Furthermore, they added viewing-direction dependent cost/benefit ratios, enabling the scheduling of imposters. A problem with these approaches is that their complexity is NP-complete, and only solutions that iterate towards a near-maximum cost/benefit ratio over a number of render cycles exist. Because of this, the application has no accurate control over the cost and benefit. Moreover, these approaches do not incorporate dynamically refreshed LODs. Especially if the frontend is extremely constrained, for instance in our wearable AR terminal [2], these restrictions are unacceptable. Finally, optimising the cost/benefit ratio does not necessarily lead to a consistent final image quality; some objects may have higher-than-average, others lower-than-average quality.

3. The UbiCom graphics system

The UbiCom graphics subsystem is an overall approach to address the problems discussed in the previous section. It consists of a low-latency low-power renderer and a dynamic LOD system, working together via latency layering and the UbiCom-wide QoS management philosophy. In this section we discuss these components and their relations in more detail.

3.1. Latency-layering

In order to meet the end-to-end latency requirement of 10 ms on a wireless terminal, a latency-layered hierarchy is used [19]. Figure 2 shows the latency layering for the graphics system. In the inner loop, the user position is tracked and the image is re-rendered and displayed within 10ms. To run this inner loop within 10ms, we use extremely fast (2 ms) but somewhat inaccurate inertial trackers, and render and display a simplified scene closely resembling the full scene. The correction of the tracker positions and update of the simplified objects run at a lower rate in the backbone. The tracking system uses similar latency layering.

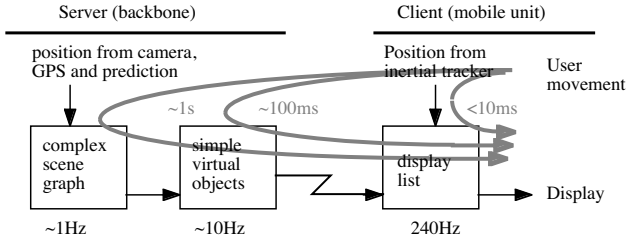

Figure 2. Latency layering in the rendering system.

3.2. Low latency rendering

To render and display the display list within 8 ms (so that we have 2 ms left for the tracking system), we developed a low-power variant of the just-in-time pixel approach of Mine and Bishop [20], running on a conventional Voodoo2 3D game card [21]. Our rendering system renders frame slices just ahead of the raster beam (see Figure 3). For a 65Hz display, a frame is displayed in approximately 16 ms. Dividing the frame into horizontal slices, and rendering just ahead of the slice being displayed, reduces the latency to 8 ms (4 ms rendering and 4 ms display).

We have measured a latency of 8.5 ms between the moment the tracker data comes in and the moment the picture part corresponding to that position has fully been displayed [6, 22].

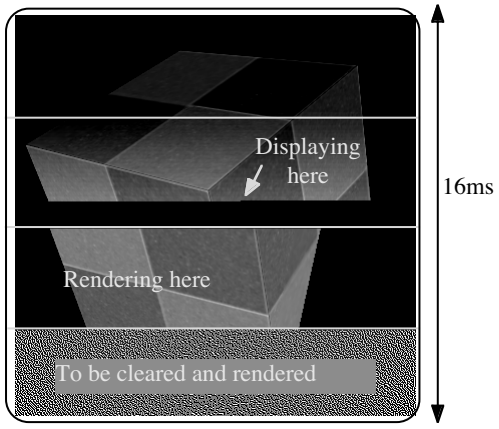

Figure 3. Our 8 ms latency rendering system runs on a conventional 3D game card, and renders slices just ahead of the rasterbeam.

3.3. LOD handling

The number of polygons in the scene has to be reduced drastically in order to be manageable by the mobile unit and wireless link. Our system can accomodate all conventional level of detail methods from Figure 1.

In order to estimate the potential benefits and requirements of distributed dynamic LOD handling, we made an extensive model of the various simplification methods, to find out their respective CPU and memory requirements, and load on the wireless network as a function of the required image quality and user movements. Figure 4 shows some typical results. The life time of the simplification includes predicted user movements. It can be seen that for distant objects, simple and meshed imposters are very suited to replace the virtual object, but for nearby objects the original or simplified polygon models have to be used. Because imposters are only simple images on a mesh, they can only be used as long as the user is approximately in front of them, and have therefore to be refreshed regularly. Simplified polygon models also may need to be updated, for instance if the meshes were optimized for good contours [23] or specular highlights [24]. For a more detailed description of the model, the reader is referred to [25, 19].

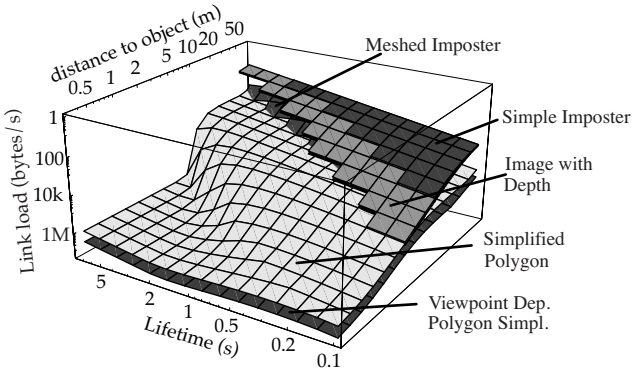

Figure 4. Typical results of our mathematical model. Here the average load on the link is plotted as a function of the planned lifetime of simplified objects and their distance to the user.

3.4. QoS management

It is difficult to schedule simplification methods such that the required quality is reached while minimising the resource load. The mathematical model as described in the previous section is useful, but ultimately only the application can decide which quality and resource load is acceptable. We therefore developed a three-phase quality of service (**QoS**) mechanism that allows the application to steer the scheduling, quality and resource load. So before any rendering takes place, the application gets the opportunity to pick the proper operation point on a curve representing the current quality/load tradeoff.

This QoS mechanism is based on the project-wide adaptive resource contracts (**ARC**) QoS management philosophy [26]. The ARC mechanism is designed to support resource management in a distributed environment, where none of the components has system-wide knowledge. Such a mechanism is essential in large systems, such as our mobile distributed AR system. Each component can be regarded as server for its users, while it is a client of lower-level layers. ARC components communicate QoS and resource utilisation parameters through their interfaces. The operation-space parameters are transformed to a multi-objective optimization problem

local to the component in question. Analysing and optimising the options in operation space yields requested trade-off curve.

In ARC, requesting a service goes in three phases. In the first phase (Figure 5), the client transforms its internal requirements (mode) into the proper options in operation space. If the server knows all constraints, it enters the second phase. In case the server depends on lower-level services however, it transforms the request and passes it on, so as to gather the necessary constraint information.

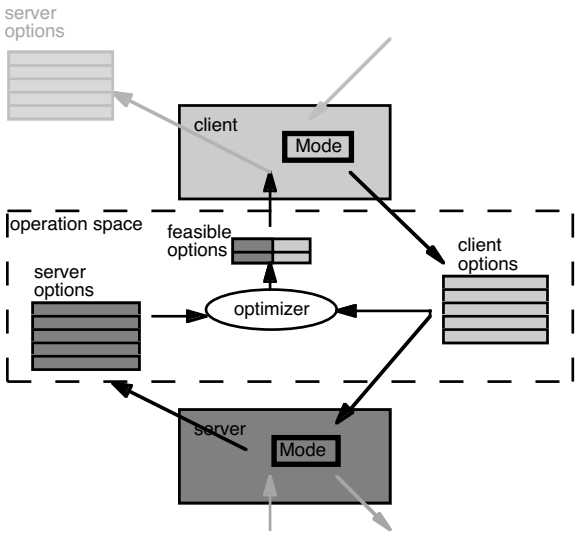

Figure 5. Phase 1 in the ARC style of QoS management. See text.

In the second phase, the server has received all constraints and returns the trade-off curve (server options). An optimizer searches the best match between the posed request and the offered options, to transform the resulting set back into the client mode.

Finally, at the top of the hierarchy a client (typically, the application) starts the third phase by deciding on a specific operation point and instructing its server (establishing a contract). To settle the contract at the server-side of a component, the component may also establish its own contracts at its client-side. As soon as contracts have been settled real service can commence.

3.5. Accuracy curves

To fit the ARC model to rendering, we have to split the render action into three phases. Firstly, the application requests the whole range of options from the rendering system. Secondly, the application picks the proper operating point (typically, the required accuracy and the amount of resources to be used), and finally, it requests rendering of the scene graph using this operating point.

To expose the rendering possibilities to the application, we use accuracy curves. An accuracy curve describes the relation between the resources and the reachable accuracy (see Figure 6). The curve is monotonically increasing, as providing more resources (for instance, more polygons) should never imply lower accuracy. Note that we extended the ARC concept slightly, as we allow linear interpolation between the discrete points in operation space.

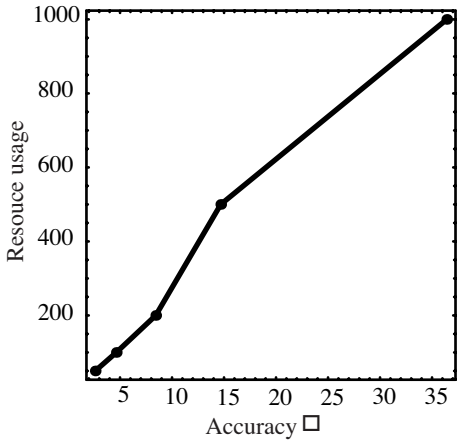

Figure 6. Typical accuracy curve: Garland's cow [13]. Horizontally the accuracy, vertical the amount of resources required to reach that accuracy. See text.

In our prototype implementation, accuracy is the inverse of the maximum displacement of pixels on the screen (due to the simplification used internally in the rendering system), and the resource usage is represented by the number of polygons in the simplifications.

To find the accuracy curves for the entire scene graph, the curves are first calculated for the leaf nodes and propagated upwards. For leaf objects the curves can be either derived mathematically, automatically estimated or explicitly given by the scene designer. Propagating curves through the VRML LOD and group node can be done efficiently.

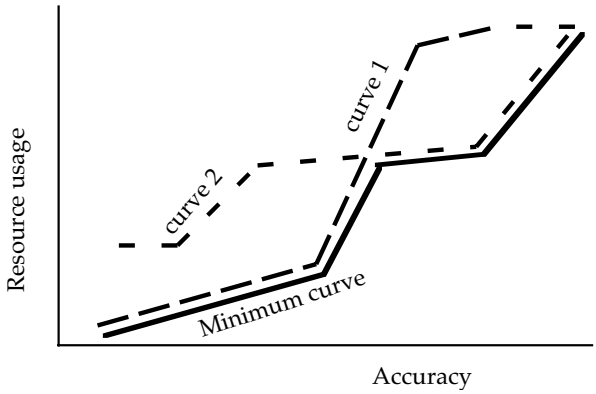

Figure 7. Propagation of curve1 and curve2 through an LOD node, the result is the minimum curve.

Figure 7 illustrates the most interesting case: propagation through a LOD node. During rendering, an LOD node will pick the cheapest representation that gives the required accuracy. Therefore, the minima of the resource usages of the curves of the children of the LOD node are

taken for the curve of the LOD node. Many details, optimizations and exceptions have to be taken into account. For more details, the interested reader is referred to [25, 27].

4. Statue application

To test the functionality of our system, we placed a virtual statue on the campus. The statue was scanned in from a real statue, and consists of 343k polygons. Our current prototype can not handle this model in full detail in real time, because all VRML files are currently transmitted to and parsed uncompressed in the backbone simplification machines. Instead, we used a 10k polygons model internally.

At this time, our outdoor tracking system is not yet operational. To still get an impression of the behaviour of our real-time rendering system, we replaced the real world and the tracking system by a video file annotated with position and orientation data. The video was played back with synchronous position information routed to our rendering system. Our rendering system thus behaved exactly as if a user was walking over the campus, and generated AR images in real time. These were mixed optically with the video to avoid mixing artefacts [28]. The set-up thus obtained is shown in the diagram of Figure 8.

The environment video was recorded by moving around with a video camera in front of the planned location of the virtual statue. The movements are similar to the head movements a person with our AR headset would make. The video was corrected both for barrel distortion and for the time difference between the time the upper and lower line of the image are displayed on a conventional TV.

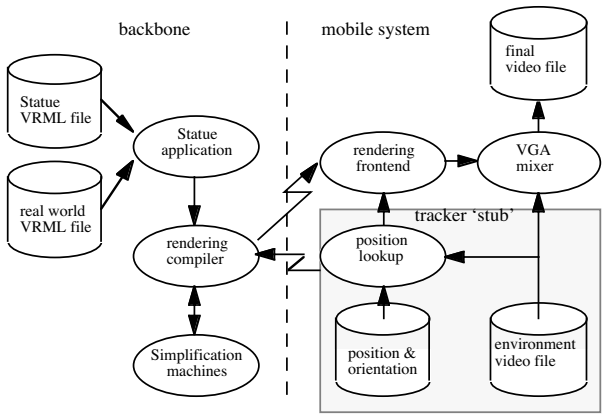

Figure 8. Setup to test our real-time low power rendering system.

The position and orientation data was recovered from the individual camera images. The position was recovered for each camera halfframe, by matching a few recognised points (typically 3) with the real-world positions stored in a GIS database. A simple symmetric 5-tap low-pass filter was applied to remove some of the noise on the estimated camera orientation.

We used the audio channel of a digital video tape to synchronise the tracker stub with the video playback, so that the position and orientation corresponding to the current video frame could be picked from the position file.

Figure 9 shows two snapshots from the final video file. Figure 9a shows a nearby view of the statue. The statue is half-transparent due to the optical mixing of our AR system. Figure 9b shows a distant view, where the virtual statue is partly occluded by real-world artwork. The video shows that our prototype reaches the real-time requirements, and that this results in stable projection of the statue even when fast head movements occur. Also, the latency layering shows to work out well in practice.

Figure 9a. Nearby view at the statue.

Figure 9b. Detail of more distant shot, where the virtual statue is partially occluded by real-world artwork.

To get a better impression of the behaviour of our QoS mechanisms, we made two plots of the resource usage versus the image quality. To challenge the system somewhat more, for this test we walked around in a more complex scene with a cow, two spheres and a cylinder at a few meters distance. Figure 10 shows the plot when the application requests a constant resource load of 200 polygons. As can be seen, the resource load is accurately maintained and never exceeded. The visual quality measure (SNR) shows a few dips, caused by temporal absence of virtual objects on the display. Figure 11 shows the result for a constant target accuracy of 200/rad. Again this target is met accurately, and the SNR is quite stable around 10dB. More details can be found in [27].

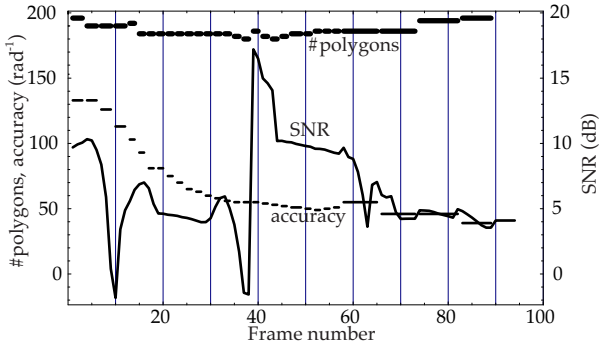

Figure 10. Resource load versus image quality when constant resource load is requested.

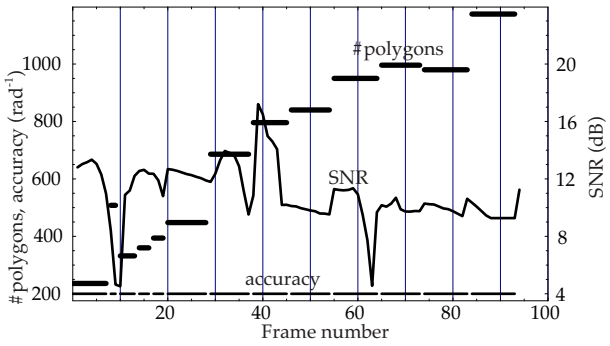

Figure 11. Resource load versus image quality when constant accuracy is requested.

5. Discussion and conclusions

The current viewpoints could be handled with simple and meshed imposters. If the observer would come closer to the statue, it becomes necessary to represent the object with a polygon model, because the bandwidth to the headset does not allow us to update images quick enough. Although the meshed imposters work fine in the statue application we discussed, disturbing artefacts were noticed when the virtual objects are relatively deep. This topic could use more research.

The geometric distortions of simplified polygon models are acceptable at near range, but to keep the statue visually appealing, a texture should be mapped on the polygon model. Although the simplification software can handle such textures, we don't have software to generate the required textures. Mechanisms for automatic generation of textures exist [29]. But they require the object surface to be cut into a large amount of patches, and this process will disturb dynamic simplification of the mesh because texture coordinates can not be interpolated anymore [30].

Our prototype system is not really low power, as it is based on off-the-shelf PC hardware. But the current NVidia GeForce2-go single-chip accelerator uses less than 1W and has higher performance [31], so it can theoretically replace our Voodoo2 graphics card. We are working on hardware for a wireless low-power platform [32].

We discussed the concept of the rendering system of the UbiCom mobile AR project. To explain the total concept and its relations, latency layering, low-latency rendering, LOD handling, and QoS management using accuracy curves were described. Finally we showed the capability of the rendering system to render complex virtual objects in a stable and robust way, while at the same time using only little power.

6. References

[1] Tiberius CCJM. Recursive data processing for kinematic GPS surveying. PhD thesis, Delft University of Technology, faculty of Civil Engineering and Geosciences, 1998.

[2] UbiCom. Ubiquitous Communications: Aiming at a new generation systems and applications for personal communication. Interdisciplinary program (DIOC) at Delft University of Technology. 2000. Available Internet: http://www.ubicom.tudelft.nl.

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

[4] Poot HJG de. Monocular perception of motion in depth. Doctoral dissertation, Faculty of Biology, University of Utrecht, Utrecht, The Netherlands, 1995. ISBN 90-393-0820-9.

[5] Padmos P, Milders MV. Quality criteria for simulator images: A literature review. Human Factors, 1992, 34 (6), 727-748.

[6] Pasman W, Schaaf A van der, Lagendijk RL, Jansen FW. Accurate overlaying for mobile augmented reality. Computers & Graphics, 1999, 23 (6), 875-881.

[7] Olano M, Cohen J, Mine M, Bishop G. Combatting rendering latency. Proceedings of the 1995 symposium on interactive 3D graphics (Monterey, CA, April 9-12), 1995, 19-24 and 204. Available Internet: www.cs.unc.edu/~olano/ papers/latency.

[8] Cohen J, Olano M. Low latency rendering on pixel-planes 5. UNC Chapel Hill department of computer science technical report TR94-028, 1994. Available FTP: //ftp.cs.unc.edu/pub/users/cohenj/LowLatency/ lowlat.ps.Z.

[9] Aliaga DG, Lastra AA. Smooth transitions in texture-based simplification. Computers Graphics, 1998, 22 (1), 71-81.

[10] Darsa L, Costa B, Varshney A. Walkthroughs of complex environments using image-based simplification, Computers and Graphics, 1998, 22 (1), 55-69. Available Internet: http://www.cs.sunysb.edu/~varshney/publications.html.

[11] McMillan L. An image-based approach to three-dimensional computer graphics. PhD Thesis, University of North Carolina at Chapel Hill, 1997. Also available as UNC Technical Report TR97-013. Available Internet: http://graphics.lcs.mit.edu/~mcmillan/Publications/diss.pdf.

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

[13] Garland M. Quadric-based polygonal surface simplification. Doctoral thesis, Carnegie Mellon University, 1999. Available Internet: http://www.cs.cmu.edu/ ~garland/thesis.

[14] Lengyel J, Snyder J. Rendering with coherent layers. Proceedings of the SIGGRAPH'97, 1997, 233-242.

[15] Schaufler G. Dynamically Generated Impostors. GI Workshop "Modeling - Virtual Worlds - Distributed Graphics", D.W.Fellner (Ed.), infix Verlag, November 1995, 129 - 135. Available Internet: http://zeus.gup.uni-linz.ac.at/~gs/papers/imposter.

[16] Decoret X, Schaufler G, Sillion F, Dorsey J. Multi-layered impostors for accelerated rendering. Computer Graphics Forum (Proceedings of Eurographics '99), 1999, 18 (3), 61-73. Available Internet: http://www-imagis.imag.fr/~Jean-Marc.Hasenfratz/EG99/paper-EG99.html.

[17] Funkhouser TA, Séquin CH. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. Computer Graphics (SIGGRAPH'93 proceedings) 27, 1993, 247-254.

[18] Mason AEW, Blake EH. Automatic Hierarchical Level of Detail Optimization in Computer Animation. Computer Graphics Forum (Eurographics'97 proceedings), 1997, 16 (3), C191-C199. Available Internet: http://www.cs.uct.ac.za/ ~amason/euro.

[19] Pasman W, Jansen FW. Latency layered rendering for mobile augmented reality. Proceedings of the 21th symposium on information theory (May 25-26, Wassenaar, The Netherlands), 2000, 45-54. Available Internet: http://www.cg.its.tudelft.nl/~wouter/publications.

[20] Mine MR, Bishop G. Just-In-Time Pixels. University of North Carolina at Chapel Hill Technical Report TR93-005, 1993. Available Internet: http://www.cs.unc.edu/~mine/mine_publications.html.

[21] Creative Labs. 3DBlaster Voodoo2. 1999. Available Internet: http://www.creativelabs.com/graphics/voodoo2/spec.html.

[22] Pasman W. Measured latency of low-latency frontend. Internal document, Delft University of Technology, 1999. Available Internet: http://www.cg.its.tudelft.nl/~wouter/publications/publ.html.

[23] Luebke D, Erikson C. View-dependent simplification of arbitrary polygonal environments. Proceedings of the 24th annual conference on Computer graphics & interactive techniques (SIGGRAPH'97), 1997, 199-208.

[24] Xia J, El-Sana J, Varshney A. Adaptive Real-Time Level-of-detail-based Rendering for Polygonal Models. IEEE Transactions on Visualization and Computer Graphics , 1997, 3 (2), 171 - 183. Available Internet: http://www.cs.sunysb.edu/~varshney/publications.html.

[25] Pasman W, Jansen FW. Comparing object simplification methods for thin client 3D rendering systems. Submitted to IEEE Transactions on Visualization and Computer Graphics, 2000.

[26] Dijk H, Langendoen K, Sips H. ARC: A bottom-up approach to negotiated QoS. Proc. WMCSA'2000, (7-8 December 2000, Monterey, CA), 128-137.

[27] Pasman W, Jansen FW. Scheduling Level of Detail with Guaranteed Quality and Cost. Submitted for the ACM Symposium on Virtual Reality Software and Technology (VRST'01, Banff Center, Canada), 2001.

[28] Pasman W. Optical mixing with a half-transparent mirror and two monitors. Internal report, UbiCom, Delft University of Technology, February 2001. Available Internet: http://www.cg.its.tudelft.nl/~wouter/publications/publ.html.

[29] Cohen J, Olano M, Manocha D. Appearance-perserving simplification. Proceedings of the 25th annual conference on Computer Graphics, 1998, 115 - 122.

[30] Garland M, Heckbert PS. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. IEEE Visualization 98, 1998, 263 -269, 542. Available Internet: http://www.cs.cmu.edu/~garland/quadrics/quadrics.html.

[31] NVidia. FAQ: How much power does the GeForce2 Go consume? 2000. Available Internet: http://www.nvidia.com/Products/GeForce2Go.nsf/ faqs.html.

[32] Bakker JD, Mouw E, Joosen M, Pouwelse J. The LART Pages. Delft University of Technology, Faculty of Information Technology and Systems, 2000. Available Internet: http://www.lart.tudelft.nl.