Scheduling Level of Detail with Guaranteed Quality and Cost

W. Pasman
UbiCom
Delft University of Technology
+31-15-2783107, Delft, Netherlands
W.Pasman@twi.tudelft.nl

F. W. Jansen
Computer Graphics/Mediamatics
Delft University of Technology
+31-15-2785517, Delft, Netherlands
F.W.Jansen@twi.tudelft.nl

ABSTRACT

This paper presents a framework for handling and on-the-fly generation of levels of detail. The application directly controls the trade-off between the amount of resources to be used and the accuracy of the final images. The basis for this choice is an accuracy curve which explicates the trade-off. This curve is calculated and updated hierarchically, which makes it especially suited for use with a scene graph. Integration of the framework with VRML is described. Measurements on our prototype implementation show that target resource loads and accuracies can adequately be reached.

Categories and Subject Descriptors

I.3.3 [**Computer Graphics**]: Picture/Image Generation – *Viewing algorithms*; I3.6 [**Computer Graphics**]: Methodology and Techniques – *Graphics data structures and data types*.

General Terms

Algorithms, Management, Performance.

Keywords

On-the-fly Level of Detail Generation, Scheduling, Quality, Cost, Trade-off, Scene Graph.

1. INTRODUCTION

Appropriate levels of detail (LOD) can maximise the visual quality of the rendering while minimising the rendering costs. Especially in thin client systems, simple and meshed imposters are very good in reducing the number of polygons and the resource usage in the client, while preserving the quality of the final images.

VRML is a convenient language to describe a virtual scene. Unfortunately standard VRML is not very suited for using imposters. First, it uses only the distance between the viewpoint and the LODs to pick the appropriate LOD for rendering. In general, this is insufficient for imposters, because they can be used only when the user is in front of the imposter – the distance to the imposter is probably not even very critical. Second, the scene designer has to do a lot of effort to get a constant final image quality. Furthermore, VRML has no mechanism to dynamically generate the imposter, and therefore the scene designer would have to design a large number of imposters and put them into the scene. Finally, VRML does not support progressive meshes, a convenient way to store multiple resolution versions of virtual objects.

A radical approach would be to uncouple the rendering entirely from the scenegraph, and to allow the rendering system to introduce simplifications anywhere it thinks appropriate, completely transparent to the application and VRML. However, this leaves very little control to the application and scene designer with respect to the realtime behaviour of the system. Furthermore, current approaches in this direction are computationally so complex that the best they can do is to iterate towards a – hopefully sufficient – optimum. Other systems in this direction support only simple imposters in addition to the original polygon objects, which is inefficient and maybe even unusable because the original polygon objects may be too complex to render while the object may be too close to be represented with a simple imposter.

These quality-performance trade-off issues are generally not critical in high-quality rendering systems. However, when the rendering system is part of a mobile system, the rendering system becomes one of the many components competing for scarce system resources – the battery, memory, CPU, the wireless radio link, the video compressor, the camera, the positioning systems (GPS, gyros) etc. The optimization question thus stretches over the entire system, and the optimum can not be found by looking at the graphics system alone.

Within the UbiCom program [24] an overall quality of service (QoS) approach was developed to enable this propagation [29]. In this paper we describe the LOD mechanism that we developed to fit this QoS approach, and the way it fits into a scene graph. The paper is organised as follows. Section 2 discusses related work. Section 3 presents our framework for LOD scheduling, and discusses how the framework fits within VRML. Section 4 presents some measurements on our prototype implementation, and discusses possible improvements.

2. RELATED WORK

Various LOD rendering methods have been proposed to simplify complex virtual objects, such as simple imposters [1, 2], meshed imposters [3, 4], images with depth [5, 6, 7], layered depth images [8, 9] and simplified polygon models [32, 10, 30, 23, 31].

Simplified polygon models are 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. If the polygon model is simplified extremely, the simplification will become very distorted. Instead, simple imposters holding a picture of the object are much better when it comes to simplifying objects to a very low number of polygons. Meshed imposters are most useful at moderate distances. See Figure 1. Additionally, imposters are important to cluster or group a number of separate objects into a single simplified representation. Polygon clustering methods are not well-developed and usually are limited to gap filling techniques [10].

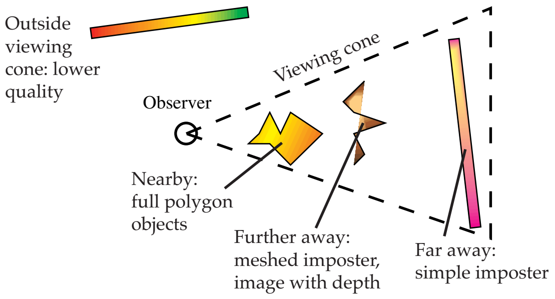
Outside viewing cone: lower quality
Viewing cone
Observer
Nearby: full polygon objects
Further away: meshed imposter, image with depth
Far away: simple imposter

Figure 1. Efficient way of using different simplification methods depending on the distance to the observer.

Thus, each LOD method has its own optimal range where it can be used, depending on the size of the object it replaces, the distance to the observer, the observer speed, the performance of the rendering engine, and the bandwidth between the rendering frontend and the place where the LODs are stored or generated. We modeled all these parameters, and comparisons were made of the network load, rendering load and memory load when using various simplification methods. Figure 2 gives an impression of the results. The mathematical model is fully exposed in earlier work [11]. This paper describes how to map the theoretical model into deterministic and efficient software, and integrates this model with VRML and quality of service handling.

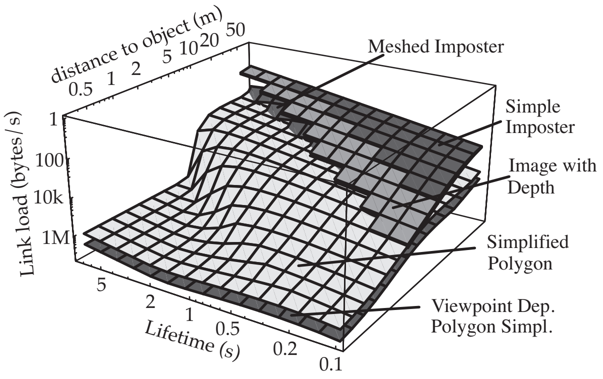

Figure 2. Typical results of our mathematical model. Here the average load on the link is plotted as a function of the planned lifetime of simplified objects and their distance to the user.

A few frameworks for LOD handling have been proposed earlier. The main problem with current dynamic simplification systems is that they support only a subset of the available simplification methods. Most proposed systems focus on a single simplification method. In [18] and [8], space is partitioned hierarchically, using a k-d tree, and previously rendered simple imposters for objects or groups of objects within a bounding box of the k-d tree are automatically reused. Imposters for larger bounding boxes can also be refreshed using images from smaller bounding boxes. To increase the lifetime of the imposters, it was proposed [19] to use a more elaborate kind of simple imposter called layered impostors. However, this introduces problems with visibility gaps, similar to those in an image with depth. Lengyel and Snyder [20] group objects considering their relative velocity, perceptual distinctness and ratio between transparent and opaque pixels in the simple imposter. To support this process, they use a dynamically updated k-d tree, which also allows their system to cope with animated geometry. Decoret et al. [21] introduced dynamically updated meshed imposters. They group several objects into multiple meshed imposters. The overlap between the objects is used to steer the grouping of objects into the meshed imposters, such that occlusion artefacts are minimised.

There are a few general frameworks that can support any kind of representation. Funkhouser and Sequin [22, 23] select a combination of representations maximising the overall cost/benefit ratio. The number of polygons, pixels and vertices in the object are used to make an estimation of the rendering costs and benefits of that object. Mason and Blake [28] improved that framework, by adding support for hierarchical scene descriptions and for clustering or grouping of objects. Furthermore they added viewing-direction dependent cost/benefit ratios, enabling the use of imposters. A problem with these approaches is that their complexity is NP-complete, and only solutions that iterate towards a near-maximum for the cost/benefit ratio over a number of render cycles exist. Because of this, the application has no accurate control over the cost and benefit. Furthermore, these approaches do not incorporate dynamically refreshed LODs. Especially if the frontend is extremely constrained, for instance a wearable AR terminal [24], these restrictions are unacceptable.

3. THE FRAMEWORK

Central to the proposed framework is the extension of every node in the scene graph with an accuracy curve. This curve describes, for that node, the resource load as a function of the accuracy in the final image. It represents not a single point, but the full range of possibilities, such that in the end a suitable point can be found quickly. This accuracy curve can be calculated for any object and group of objects in the scene graph. For the leaf nodes, the curve is specified, estimated or derived mathematically, and for the intermediate nodes in the scene graph the curves of the children are combined into a new curve (Figure 3). The application can pick the required accuracy and/or resource load at the top node, and trigger the use of that configuration or decide to take other actions. LODs are generated or updated only after the application has chosen the proper operation point.

The following sections describe the accuracy curves in more detail, how they propagate through the scene graph, and how VRML can be adapted to fit our framework.

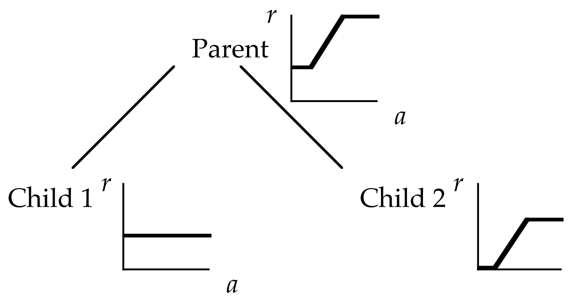

Figure 3. Propagating accuracy curves upwards through the scene graph. In the curves horizontally the accuracy *a*, vertically the amount of resources *r*.

3.1 Accuracy curves

An accuracy curve describes the minimal amount of resources required to reach an accuracy. The curve is monotonically increasing, as providing more resources should never give lower accuracy.

Usually a geometric distortion measure for simplified objects is calculated by dividing the maximum distance between the original and simplified geometry by the size of the original object. Alternatively, we propose accuracy measure $a=1/d$, where d is the usual distortion measure. This is advantageous because for most polygon reduction methods the geometric distortion $d \approx K/r$ [23, 11] where K is some constant depending on the object, and r the available resources, such as the number of polygons or vertices. Therefore the accuracy becomes a simple linear equation $a = 1/d = \frac{1}{K} r$. In order to be able to match the less trivial curves of real objects and groups of objects, we use the slightly more general form $r = R + K \cdot a$, where R and K are constants depending on the object complexity, and make for the accuracy curve a partwise linear approximation (Figure 4). Note that the accuracy in practice never goes to infinity because curved surfaces of real objects can only be approximated by triangles. For simplicity we use the geometric accuracy, but probably this can be generalised to a perceptually more precise measure.

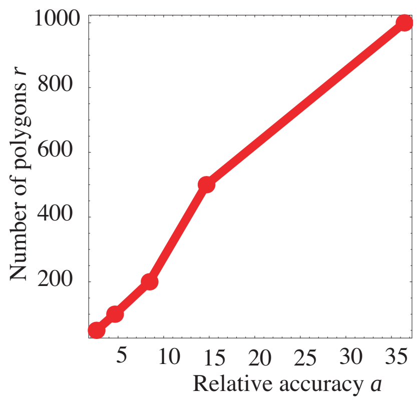

Figure 4. Accuracy curve for Garland's cow [13]. Horizontally the achievable accuracy *a*, vertically the available resources *r*. This curve is so linear that it could also be conveniently modelled with a single part .

Technically, we represent the accuracy curve C as a list of triples (A,R,K). A is the starting point of a linear curve part. The list is sorted by increasing A, and each A in the list is unique. Each triple represents a part of the curve starting at A. If (A_1, R_1, K_1) and (A_2, R_2, K_2) are two subsequent triples in C, the curve follows $r = K_1 \cdot a + R_1$ for $A_1 \le a < A_2$ The curve should be monotonically increasing, so if (A_1, R_1, K_1) and (A_2, R_2, K_2) are two subsequent triples, then $R_2 + K_2 A_2 \ge R_1 + K_1 A_2$. Note that this representation is more powerful than just a list of discontinuity points, because our representation allows vertical jumps. Such jumps will show up when curves are combined.

The resource r may be a multidimensional vector, for instance a tuple containing the load on the wireless link and the amount of polygons in the simplified scene. We expect that the real behaviour of those other dimensions also can be modelled reasonably accurate with partially linear curves. Difficulties arise from resource trade-off issues: it may be possible to reach an equal accuracy with different amounts of the various resources. We have ideas about how to extend our approach for this, but this falls out of the scope of this paper.

Two kinds of accuracy can be distinguished: relative accuracy and absolute accuracy. The relative accuracy is expressed as a fraction of the object's size, for instance 50 means that the maximum distortion, or distance between the original object and the simplified object, is 1/50=2% of the object's size. The absolute accuracy expresses an angle in the field of view, for instance 400 means that the maximum angular distortion is 1/400 rad. The application will typically require a minimal accuracy in screen space, while objects in the scene graph will specify their accuracy in the viewpoint-independent relative accuracy.

Converting from relative to absolute accuracy is straightforward. Using the smallest distance between the object and the observer, the angle that the object covers in the field of view can be calculated, and multiplying this angle by the relative distortion gives the worst-case absolute distortion.

3.2 Propagating accuracy curves

At the primitive objects in the leaf nodes, an approximation of the curve has to be made. For some primitives, such as the sphere and cylinder, this curve can be derived mathematically. For more general indexed face sets, we have either to ask the scene programmer to give the curve for these objects, or we have to estimate the curve. Currently we use a simple unix shell script, using QSlim [13] for polygon simplification and Metro [33] for estimation of geometric distortion, to calculate the exact curves. The curve can then be approximated by picking a few points on major knicks in the curve, while only a start- and endpoint suffices in most cases. The estimation can be combined efficiently with the generation of a progressive mesh [30], allowing quick re-generation of simplified polygon objects at runtime.

The accuracy will go to zero anyway if the viewer can come arbitrarily close. To avoid single objects taking all resources, a certain minimal distance will be used in cases that the user can come too close.

Two nodes are essential for the construction of the scene graph: the group node and the LOD node.

3.2.1 Propagating through a group node

The group node is just a container holding a number of children nodes that all have to be rendered. To calculate the accuracy curve of the group, we should add the resources required for the children separately at every possible accuracy, as follows. If $(a,r_1)\ldots(a,r_n)$ are points in the curves of the children nodes (same a, different r), then $(a,r_1+r_2+..+r_n)$ is a point in the curve of the group node (Figure 5). Technically, this can be implemented efficiently using our curve representation.

Special attention is required if the start and/or end accuracy of the two curves is not the same. Assume that the curve of child 1 starts at smaller a than the curve of child 2, and the application requests to render the scene at that small a. In that case, we expect child 1 to render at that low accuracy, while child 2 should render its lowest valid accuracy. The same holds for the end points, where we expect a child to render at its highest quality when higher is requested. Thus, it is convenient to extend the ranges horizontally, to accommodate the largest available values, when adding the curves.

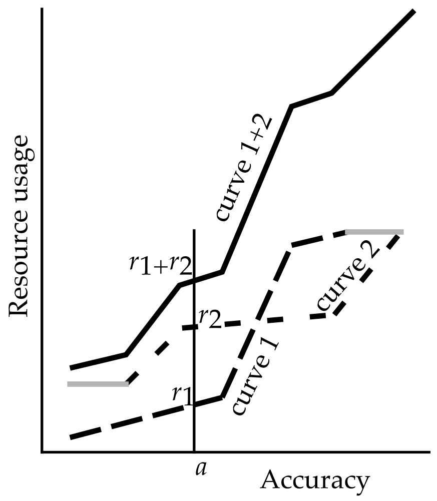

Figure 5. Adding of resources is required when two objects have to reach some accuracy. The light grey extensions of the curves indicate which values to use for adding outside of the valid range of the curve. See text.

3.2.2 *Propagating through an LOD node.*

An LOD node holds several representations for the same object. Thus, it should select the cheapest sufficient child for rendering. Only one of the children has to be rendered, so if $(a,r_1)\ldots(a,r_n)$ are points in the curves of the children then $(a,\min(r_1\ldots r_n))$ is a point in the curve of the group node (Figure 6). Again, this poses no technical problems with our curve representation although some care is needed in case two triples cross each other, as in Figure 6.

Now we can not extend the curves horizontally at the high accuracy side. If child 2 is valid up to a higher accuracy than child 1, and we request a that high accuracy, we expect child 2 to be selected by the LOD node. However, if child 1 is extended horizontally, and it happens to have lower resource usage at its endpoint than child 2 has at the requested accuracy, child 1 would be chosen instead (Figure 7). Properly, we should not extend right sides. At the left side, horizontal extension is proper: rendering a child with a higher than requested accuracy is an option, if it uses less resources.

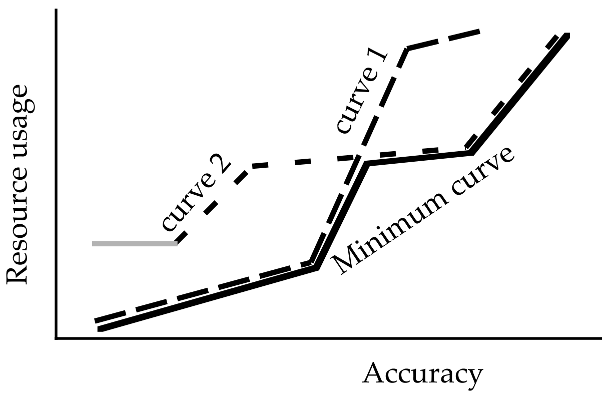

Figure 6. Taking the minimum of two accuracy curves. Note that curves should not be extended to the right when taking the minimum. See text.

Technically, it is important that our choice of representation for the curve parts makes adding and taking the minimum efficient and avoids rounding errors. Because of our choice of parameters, only the starting points of curves change when taking the minimum, only the calculation of intersections of curves has to be done somewhat careful to avoid small 'cracks' in the curve. Rounding errors during addition are very small as well.

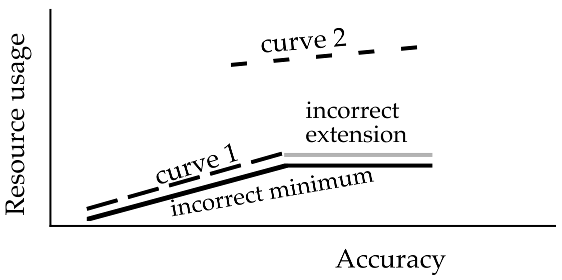

Figure 7. Extending LOD nodes horizontally to the right is incorrect when taking the minimum.

3.3 Runtime behaviour and complexity

The accuracy of the final image is equal to its least accurate part, so it makes no sense to increase the quality of arbitrary parts while rendering others at a lower accuracy – unless the application explicitly requested this. Thus, during rendering the targeted accuracy can directly be imposed on all objects in the scene. This restriction of all objects behaving synchronous is enough to avoid the NP-completeness caused by traditional scheduling approaches that consider every possible combination of LODs.

3.4 Extended use of relative accuracy

The propagation of accuracy curves as discussed in the previous section only holds for absolute accuracies. Propagating a relative curve is possible only if all children provide a relative curve. Adding a relative accuracy differs slightly from adding an absolute accuracy, because the diameter of the group differs from the diameter of the separate objects. Therefore the accuracy curves of the objects to be added have to be multiplied first, by $\frac{new\ diameter}{old\ diameter}$, to match the new radius before adding them. Typically the accuracy will become higher, because the absolute distortion stays the same while the bounding box grows. More formally, if $d_1 ... d_n$ are the radii of the children, D is the diameter of the group, and $(ad_i/D, r_i)$ are points on the accuracy curves of the children then $(a, r_1 + r_2 + .. + r_n)$ is a point in the curve of the group node.

Because absolute accuracy curves are depending on the viewpoint, absolute curves have to be refreshed after the observer made a significant movement. This can be done in a top-down manner. However, regular refresh is quite inefficient. Furthermore, absolute accuracies are calculated using the worst-case movement predictions, and therefore are worst-case values. Relative curves are not depending on the viewpoint. Therefore doing the conversion from a relative to an absolute curve as high as possible in the scene graph will improve efficiency. Doing the conversion higher in the scene graph also reduces the number of conversions required.

The use of relative accuracy curves is limited to the point where viewer gets close to – or even in – the bounding box. Therefore the best distance to do the conversion is around the point where the distance to the group becomes smaller than the group's bounding box. We assume these conversions are all done in the backbone, where maximum efficiency is not essential, so the conversion level is not very critical.

3.5 Valid range indications

Both imposters and accuracy curves are restricted in their use to only a part of 3D space. Existing interfaces specify a minimum distance from which a representation is valid [18, 21, 25, 26]. But in order to give more control to the application program, a maximum distance is useful as well. Furthermore, especially for simple and meshed imposters a direction is required. Finally, it should be possible to test quickly whether a sphere (all positions that a user might reach within a certain amount of time) completely falls within the range.

To describe such parts in space, we could use a conically shaped bounding volume (**conic range**) as in Figure 8. A problem is that we must be able to take the intersection of range indications efficiently. For instance, the valid range of a combined distortion curve will be the intersection of the valid ranges of the individual curves that the combined curve is made of. Unfortunately, intersections of conic ranges will usually not result in a conic range. Because we need concave objects to indicate minimum distances, alternatives such as a general convex hull [27] are not very suited either. Currently we use the conic range and quickly fall back to simple spheres, which are degenerate conic ranges, when taking the intersection.

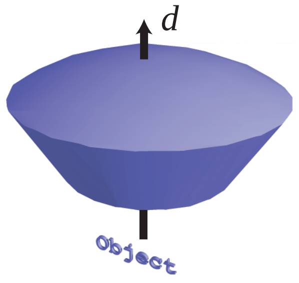

Figure 8a. A conic range can indicate a minimum and maximum to the object, as well as a maximum angular distance from some direction vector.

```
ConicRange
{
   field  SFVec3f   center      0 0 0      # <-∞,∞>, meter
   field  SFVec2f   distance    1 10       # <0,∞>, meter
   field  SFVec3f   direction   0 0 1
   field  SFFloat   alpha       3.1415     # rad, <0,p]
}
```

Figure 8b. The ConicRange VRML node.

3.6 Integration with VRML

To bring our framework into VRML, a few new VRML nodes have to be defined, and a few nodes have to be changed. All objects now also have to provide a function returning the current accuracy curve, given the transformation between the viewpoint and the object and the required life time of the simplified version of the object.

To accommodate the various simplification methods, we introduce a SimpleImposter, a MeshedImposter and a SimplifiedPolygon node. These nodes replace their children with a single simplification.

Figure 9 shows the SimpleImposter node. The rotation of the imposter node is updated automatically each time the imposter texture is refreshed, and its orientation is adjusted to optimise the expected lifetime. The size field holds the width and height of the imposter, and is by default updated automatically. The center fields work similarly. The center field is required because the children of an imposter are usually not centered around 0,0,0 while a default imposter would be. This unbalance would require the imposter to be excessively large, in order to cover the objects. The default orientation of the imposter is along the xy plane, and scaling affects the imposter in these dimensions only. If the imposter is too small, the image will hold only part of the objects it replaces. The MeshedImposter node is similar, and the SimplifiedPolygon node is similar to a Group node.

```
SimpleImposter
{
   eventIn        MFNode    addChildren
   eventIn        MFNode    removeChildren
   exposedField   MFNode    children         []
   exposedField   SFVec2f   size             2 2
   exposedField   SFBool    autosize         true
   exposedField   SFVec3f   center           0 0 0
   exposedField   SFBool    autocenter       true
}
```

Figure 9. The SimpleImposter VRML node.

The semantics of the VRML LOD node are changed slightly. Only after checking which levels are valid and have enough accuracy, the one with the lowest resource load is picked.

With the proposed nodes, very specific simplification mechanisms can be devised by the application. For instance, to render a cylinder, using either its original geometry, a meshed imposter or a simple imposter, we can use the following fragment (Figure 10). A simple imposter will only be used if the distance to the viewer is larger than 10 and below 1000. Without range specification the behaviour of the LOD will still be good, because of the accuracy curves.

```
LOD
{
    level
    [
    DEF MyCylinder Shape { geometry Cylinder {} }
    SimpleImposter { children [ USE MyCylinder ] }
    MeshedImposter { children [ USE MyCylinder ] }
    ]
    range [
        ConicRange{},
        ConicRange {distance 10 1000},
        ConicRange{} ]
}
```

Figure 10. VRML2 fragment using the proposed nodes.

4. PROTOTYPE IMPLEMENTATION

Mobile Augmented Reality is an interesting concept for offering users information on the spot. Within the UbiCom program at Delft University a mobile augmented reality system is being developed [24, 12] that will consist of a mobile receiver with an augmented reality display that will receive the rendering data over a mobile link from a base station. As the augmented reality is to operate in perfect alignment/registration with the real world, the latency requirements are extremely severe.

Earlier, we presented an approach to enable low-latency rendering on a mobile platform [12]. With the limited CPU power and communication bandwidth available to the mobile unit, our prototype implementation can handle about 350 texture-mapped polygons with a maximum end-to-end latency of 8 ms.

Our prototype implementation is distributed over three machines. For the measurements below, the computers were all connected with 100 Mbit/s ethernet. The first machine is the 'wearable' low-power low-latency frontend [12]. The second machine runs the application and all the code concerning our LOD framework except actual simplification of objects. The third machine does all simplifications and imposter generations.

We implemented the hierarchical LOD framework and the VRML adaptations in our prototype system [24]. Polygon reduction and meshed imposter optimisation is based on Garland's QSlim software [13]. Imposter images are rendered using the lights and the relevant part of the VRML scene. For meshed imposters the depth mesh is generated from the z buffer of the rendered image. We did not restrict accuracy curves to a part of space, to shorten programming time. Instead we regenerate all representations roughly once per second. In our prototype implementation we mapped the resource load to the number of polygons. This is because our low latency AR rendering system [12] can handle approximately 350 texture-mapped polygons, while texture memory and CPU load are less of an issue. These optimisations have no negative effects on the image quality, but will cause extra workload on the backbone and wireless link. Optimal use of conic ranges and the associated caching- and reuse-possibilities will reduce these workloads dramatically when the observer is making non-worst case movements or standing still.

4.1 Demo impressions

Our demo room is quite small with a usable area of about 4 by 3.5 meters. Theoretically this is too small to use meshed imposters or simple imposters, but nevertheless we used them, by setting the estimated maximum observer speed to 0.1 m/s (instead of the usual 1 m/s).

We built several demos, showing primitive objects such as spheres and cylinders, but also large meshes such as Garland's cow, with the various simplification methods used at the same time (Figure 11).

In spite of the (too) close range, the imposters look very good, so good that the switch to a simplified polygon representation causes a dramatic drop in quality. The main cause seems the lack of a texture map on the simplified polygon objects. Recently, Piponi and Borshukov [34] introduced a technique to generate the texture maps that theoretically could be combined very well with Garland's polygon simplifier, but we did not yet have the required manpower and tools to generate appropriate texture maps. Theoretically, the techniques from [Sander01] could be used, but this code is not publicly available.

The spheres and cylinders can be generated and transmitted fast enough. But our prototype system is not yet able to transmit simplified versions of the cow within the planned lifetime of 1 second, mainly because the representations for both the original and simplified cow are in ASCII VRML, which takes a lot of time to parse. Especially the generated texture form a bottleneck, as a typical 256x256 RGBA texture consists of 262000 bytes, each of which has to be converted to and from up to three ASCII digits. Typically this takes 10 seconds, an order of magnitude slower than planned.

4.2 Measurements

A test scenario was set up to estimate the ability of the system to adapt to a changing situation, containing Garland's cow, 2 spheres and a cylinder placed along the sides of the room (Figure 11). A flight path was chosen such that the viewpoint animated closely between two spheres and a cylinder, and then approached a cow model.

It is difficult to measure the reached accuracy directly, our algorithms of course reach the target nearly exactly but a high geometric accuracy does not necessarily imply a good picture. Instead, we opted to measure the signal-to-noise ratio (SNR), using the common formulas:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(s_i - f_i\right)^2$$

$$SNR = 10\ {}_{10}\log\left(VAR/MSE\right)$$

$$VAR = \sum_{i=1}^{N}\frac{\left(s_i - \bar{s}\right)^2}{N} = -\bar{s}^2 + \frac{1}{N}\sum_{i=1}^{N}s_i^2$$

where s is the image using simplified objects, f is the maximum quality image, N is the number of image bytes, which was the image size (640 x 480) times 3 (R, G, and B component), and $\bar{s}$ is the mean of the image bytes s_i. *MSE* is mean square error and *VAR* is the variance within a frame. It is tricky to compare SNR and geometric distortion, although there is some relevance as for instance Watson [35] found high correlations between geometric distortion and naming time of virtual objects. Nevertheless, these measurements should be considered to get an idea about the effectivity of the accuracy setting.

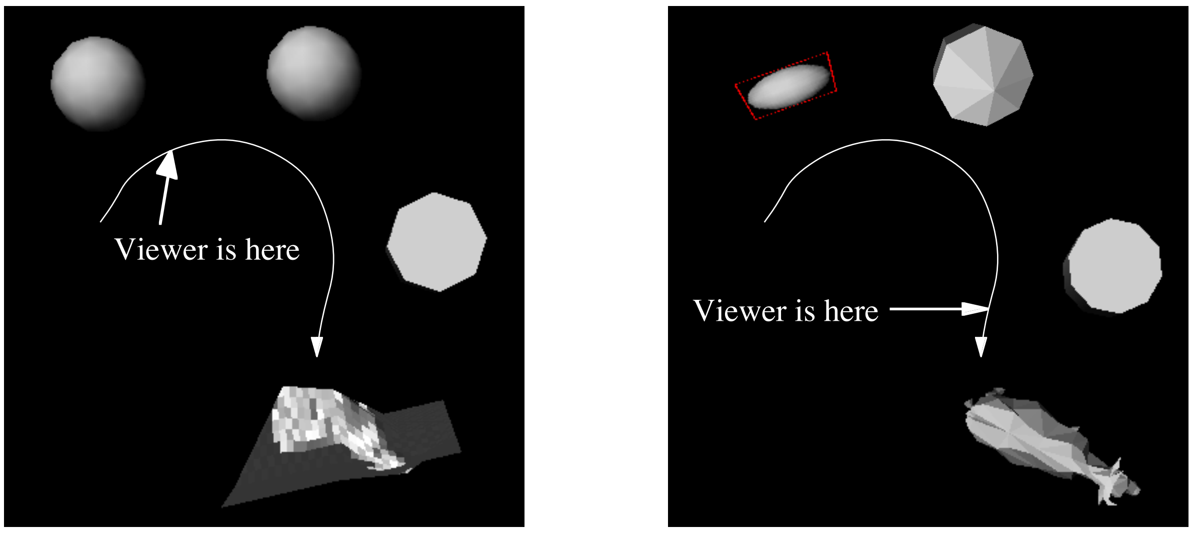

Figure 11 (not to scale). In (a) the viewer is far away from the cow, which can therefore modelled with a meshed imposter. In (b) the observer is close to the cow, but now the spheres can be represented with a simple imposter or simplified (for clarity, a frame was put around the simple imposter, and the background of the meshed imposter was rendered dark grey).

For the first test, we set a target number of triangles of 200 (Figure 12). As can be seen this target is maintained accurately. The first sphere is visible up to frame 5, the second sphere is fully visible between frame 20 and 30, the cylinder between frame 45 and 60, and the cow after frame 65. The SNR varies a lot. The two drops to 0 dB occur when all objects are out of the field of view and the screen is black.

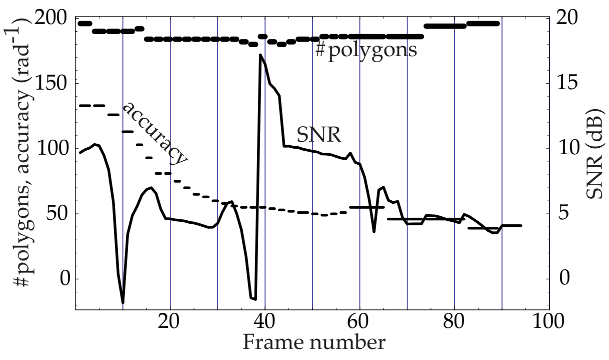

Figure 12. SNR and estimated accuracy over 100 frames. The polygon quotum is fixed to 200. See text.

In frame 30 the second sphere is filling the field of view, and it gives more distortion than the first sphere (frame 15) because the system is already saving polygons for the cow we are approaching. By the time we pass the cylinder (frame 60) nearly all polygons are spent on the cow. The accuracy curve indicates the guaranteed geometric accuracy, and is steadily going down as we get closer to the cow. The cylinder gives a higher than predicted accuracy because our lowest quality cylinder representation available has a higher accuracy than required in this case. Considering these issues, the curves of accuracy and SNR roughly match. Figure 13 shows the results of the second test, where we aim at a constant accuracy instead of a constant polygon usage. Here, the SNR varies between 9 to 13 dB, at the expense of up to 1200 polygons.

Concluding, the system can reach a target resource load very accurately. The accuracy setting is reflected in the SNR, and therefore probably also to the visual quality, but more research is required to determine whether our accuracy measure is sufficient in practice.

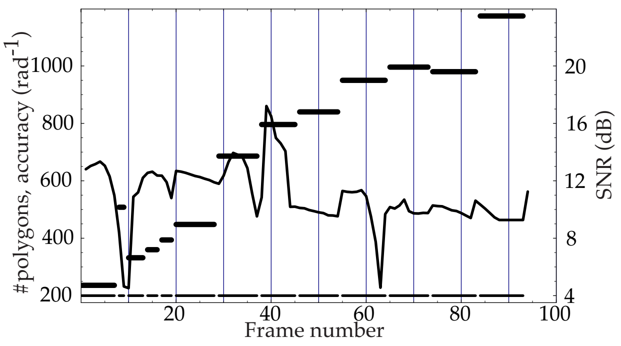

Figure 13. SNR and polygon usage, in a fixed accuracy scenario of 200 rad^{-1}. See text.

5. CONCLUSION, FUTURE WORK

We described a new framework that can handle multiple simplification methods, and can be integrated with a scene graph. The newly proposed accuracy curves are powerful enough to describe the accuracy-resource load relation with enough accuracy to be useful, while being intuitive and easy enough to be used by scene designers and artists.

Incrementally updating the textures seems a useful option both for simple and meshed imposters, and for simplified polygon objects. For the imposters, regular updates are required with relatively small viewpoint displacements, and therefore compression methods like MPEG should be able to reach compression factors in the order of 70. Imposter textures generally will be small, so it seems reasonable to use a single pixel density for the entire texture. For the polygon objects, the viewer will be close to the object, and theoretically it is worthwhile to vary pixel density within the texture, especially if the object is large [11]. However, this may require the use of advanced methods such as clipmapping [14, 15], and therefore seems unfeasible on the typical thin-client platform we are aiming for.

A related problem occurs when many textures need to be refreshed at the same time. This may cause network delays and network delays are only estimated in our prototype system. Currently we assume a constant network delay, which adds to the other delays. The total delay has to be accounted for in the planned lifetime of the simplified object. Therefore, accuracy targets may not be reached in the case of unpredicted latencies.

As all objects behave synchronous with respect to the target accuracy, changing the target accuracy may cause many objects to change representation. In practice, it may be visually preferable to delay changing of objects a little while. This may be possible by keeping part of the resources reserved for this, but of course doing so will reduce the reached accuracy.

There are a lot of possibilities to speed up the simplification mechanism. Our prototype system was built to show the principle, and was hardly optimised at all. To optimise the system, we can implement proper caching of objects, optimise parsing of ASCII VRML, and textures can be compressed before being transmitted. Also we can switch to a binary VRML format, to avoid parsing at all. To speed up simplification, a progressive mesh can be calculated in a pre-processing step, which can give an order of magnitude gain in simplification time. Multiple simplification machines can be used, to update multiple imposters in parallel.

A lot of work has been done on perceptually based LOD handling, considering contours [31], spatial frequencies, visual acuity and drop of acuity in the peripheral field of view [36, 37] and contrast rates to estimate where the user is likely to look at [36]. Some of these techniques require very fast change of rendered LOD level, for instance [37] uses different LODs for different head rotation speeds. A distributed rendering system as in our prototype implementation does not fit well with very fast changing LOD's because LOD changes are made in the backbone and updated via a –maybe wireless– network. More research is required to determine how well our approach fits to perceptually based LOD handling.

Our prototype implementation has basic support for animated objects, but with a low refresh rate, as the animation is done only in the backend. Our low-latency frontend currently can handle only fixed scenes, and does not support animation. To really support animated objects at high animation rates, our framework and the frontend need more research. Accuracy curves will also depend on speeds of objects, and rotating objects are particularly tricky. Fast rotating, distant objects are a difficult case, as these do not fit on imposters, while – depending on the rotation axis – a polygon representation still may be overkill.

The visual appearance of simplified polygon models can be improved dramatically by adding appropriate textures [34, 7] and bump maps [16, 17]. Such improvement is especially needed to make a smooth transition between the imposter models and a simplified polygon model.

6 . ACKNOWLEDGEMENTS

This work was funded by a special interdisciplinary research initiative (DIOC) of Delft University of Technology. Our prototype would have been impossible without Garland's publicly available QSlim source code. Finally an anonymous reviewer did a thorough job and made many valuable remarks on this paper.

7. REFERENCES

[1] Aliaga, D.G., Lastra, A.A. Smooth transitions in texture-based simplification. Computers & Graphics, 1998; 22 (1), 71-81. On Web: http://www.cs.unc.edu/~aliaga/Publications/cg98.pdf

[2] Maciel, P.W.C., Shirley, P. Visual navigation of large environments using textured clusters. In Proceedings of the 1995 Symposium on Interactive 3D Graphics, 1995; 95–102.

[3] Darsa, L., Costa, B., Varshney, A. Walkthroughs of complex environments using image-based simplification, Computers and Graphics 1998; (22) 1, 55-69.

[4] Riegel, T. Coding of PANORAMA 3-D sequences. 1998. On Web: http://uranus.ee.auth.gr/~alex/panorama/deliverables/D031.ps.gz

[5] Schaufler, G., Priglinger, M. Efficient Displacement Mapping by Image Warping. 10th EUROGRAPHICS Workshop on Rendering (Granada, Spain), 2000; 175-186. On Web: http://graphics.lcs.mit.edu/~gs/papers

[6] 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.

[7] Debevec, P.E., Taylor, C.J., Malik, J. Modelling and rendering architecture from photographs: A hybrid geometry- and image-based approach. Proceedings of the SIGGRAPH'96, 1996; 11-20.

[8] 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.

[9] Mark, W.R. Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping. Doctoral dissertation. UNC Computer Science Technical Report TR99-022, University of North Carolina, April 21, 1999. On Web: http://www.cs.unc.edu/~billmark/research.html.

[10] Garland, M. Quadric-based polygonal surface simplification. Doctoral thesis, Carnegie Mellon University, 1999.

[11] Pasman, W., Jansen, F.W. Comparing object simplification methods for thin client 3D rendering systems. Submitted to IEEE Transactions on Visualization and Computer Graphics, 2001.

[12] Pasman, W., Schaaf, A. van der, Lagendijk, R.L., Jansen, F.W. Accurate overlaying for mobile augmented reality. Computers & Graphics, 1999, 23 (6), 875-881. On Web: www.cg.its.tudelft.nl/~wouter

[13] Garland, M. *QSlim 2.0 [Computer Software]*. University of Illinois at Urbana-Champaign, UIUC Computer

Graphics Lab, 1999. On Web: http://graphics.cs.uiuc.edu/~garland/software/qslim.html

[14] Montrym, J.S., Baum, D.R., Dignam, D.L., Migdal, C.J. InfiniteReality: A real-time graphics system. Proceedings of the SIGGRAPH'97, 1997; 293-301

[15] Eckel, G. Cliptextures. Chapter 10 in IRIS Performer Programmer's Guide, SGI Technical Publications, Silicon Graphics Incorporated, Mountain View, CA, 1995. On Web: http://techpubs.sgi.com/library

[16] Cignoni, P., Montani, C., Rocchini, C., Scopigno, R. A general method for preserving attribute values on simplified meshes. Proceedings of the conference on Visualization '98, 1998; 59-66.

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

[18] Schaufler, G. Dynamically Generated Impostors. GI Workshop "Modeling - Virtual Worlds - Distributed Graphics", D.W.Fellner (Ed.), infix Verlag, November, 1995; 129 - 135.

[19] Schaufler, G. Per-Object Image Warping with Layered Impostors. Proc. 9th Eurographics Workshop on Rendering '98 (Vienna, Austria, June 29-July 1), 1998; 145 - 156.

[20] Lengyel, J., Snyder, J. Rendering with coherent layers. Proc. SIGGRAPH'97, 1997, 233-242.

[21] 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.

[22] Funkhouser, T.A., Séquin, C.H., Teller, S.J. Management of large amounts of data in interactive building walkthroughs. Proc. 1992 symposium on Interactive 3D graphics, 1992, 11 - 20.

[23] Funkhouser, T.A., Séquin, C.H. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. Computer Graphics (SIGGRAPH'93 proceedings) 27 (Augustus), 1993, 247-254

[24] UbiCom. Ubiquitous Communications: Aiming at a new generation systems and applications for personal communication. DIOC program at Delft University of Technology. 2000; On Web: http://www.ubicom.tudelft.nl

[25] Web3D Consortium. VRML97 International Standard ISO/IEC 14772-1:1997.

[26] Eckel, G. Active Surface Definition. Chapter 15 in IRIS Performer Programmer's Guide, SGI Technical Publications, Silicon Graphics Incorporated, Mountain View, CA, 1995. On Web: http://techpubs.sgi.com/library

[27] Barber, C.B., Dobkin, D.P., Huhdanpaa, H. The Quickhull algorithm for convex hulls. ACM Trans. on Mathematical Software, 1996, 22 (4), 469-483. On Web: http://www.geom.umn.edu/software/qhull

[28] Mason, A.E.W., Blake, E.H. Automatic Hierarchical Level of Detail Optimization in Computer Animation. Computer Graphics Forum, 1997, 16 (3), 191-200.

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

[30] Hoppe, H. Progressive meshes. SIGGRAPH proceedings, 1996, 99-108.

[31] Luebke, D., Erikson, C. View-dependent simplification of arbitrary polygonal environments. SIGGRAPH proceedings, 1997, 199-208.

[32] Clark, J.H. Hierarchical geometric models for visible surface algorithms. Communications of the ACM, 1976,19 (10), 547 – 554. Also in JC Beatty & KS Booth (Eds, 1982), Tutorial: Computer Graphics (2nd. ed), IEEE computer society, Los Angeles, CA, 296-303.

[33] Scopigno, R. (1998). Surface Simplification: Measuring Error on Simplified Surfaces. Available Internet: http://vcg.iei.pi.cnr.it/metro.html.

[34] Piponi, D., Borshukov, D. Seamless Texture Mapping of Subdivision Surfaces by Model Pelting and Texture Blending. Proc. SIGGRAPH'2000, 471-478.

[35] Watson, B.A., Friedman, A., McGaffey, A. Measuring and predicting visual fidelity. Proc. SIGGRAPH 2001, 213-220. Available Internet: http://www.cs.nwu.edu/~watsonb/school/publications.html.

[36] Luebke, D., Hallen, B. Perceptually Driven Simplification for Interactive Rendering. In Rendering Techniques (S. Gortler & K. Myszkowski, Ed.), Springer-Verlag, London, 2001.

[37] Reddy, M. Perceptually Optimized 3D Graphics. IEEE Computer Graphics and Applications, 2001, 21 (5), 68-75.