

# A Market Model for Level of Detail Control

J.Howell, Y. Chrysanthou, A. Steed, M.Slater

University College London

Gower Street

London, WC1E 6BT

+44 171 387 7050

(J.Howell / Y.Chrysanthou / A.Steed / M.Slater)@cs.ucl.ac.uk

## ABSTRACT

In virtual reality simulations the speed of rendering is vitally important. One of the techniques for controlling the frame rate is the assignment of different levels of detail for each object within a scene. The most well-known level of detail assignment algorithms are the Funkhouser[1] algorithm and the algorithm where the level of detail is assigned with respect to the distance of the object from the viewer.

We propose an algorithm based on an analogy to a market system where each object does not have an assigned level of detail but has the ownership of a certain amount of time which it can use to be rendered with. The optimization of the levels of detail then becomes a simplistic trading process where objects with large amounts of time that they don't need will trade with objects who have need of extra time.

The new algorithm has been implemented to run on the DIVE[2] virtual environment system. This system was then used to perform experiments with the aim of comparing the performance of the algorithm against the other two methods mentioned above.

## Keywords

Level of detail, rendering, framerate, DIVE.

## 1. INTRODUCTION

In virtual reality simulations the speed of rendering is vitally important especially in applications where realtime interaction with the simulation is required [3]. One of the techniques that is used for controlling the frame rate is that of having several different geometrical representations for each object and then assigning a level of detail to each object in such a way as to keep the frame rate high whilst maximizing the visual quality of the images that are rendered. The most well-known level of detail assignment algorithms are the Funkhouser[1] algorithm and the algorithm where the level of detail is assigned with respect to the distance of the object from the viewer which we will call the

*distance method*. The DIVE[2] virtual environment system originally only had this distance method implemented to control the scene complexity when rendering a scene. In order to improve on this a version of the Funkhouser algorithm was implemented in DIVE whereupon it became apparent, using small scenes that could be solved recursively, that there was room for improvement with the assignment of the level of detail for each object. This led to the concept of using a market model.

In the next section we briefly review some background work on level-of-detail control, concentrating mainly on the Funkhouser method. In Section 3 we present the principles of the new method followed by some implementation specifics in Section 4. The experiment to evaluate the new method and the results are described in Sections 5 and 6 respectively.

## 2. PREVIOUS WORK

The representation of objects using different several levels of detail (*lod*) was first proposed by Clark [4] in 1976. Since then a huge amount of research has gone into techniques for creating lods and using them to reduce the complexity of the rendered geometry [5, 6]. However, most of the common lod methods are static. They use certain unchanging thresholds, such as the distance of object from the viewpoint or the projected area on the screen, applied to each object individually, for deciding what lod to use. They merely reduce the geometry rather than regulate it since they make the complexity of rendering dependent on the complexity of the visible part of the scene at each frame.

It wasn't until much more recently that dynamic methods (predictive [1] or adaptive [9]) appeared. Here the thresholds for selecting the lods change at each frame depending on the amount of visible geometry so that the frame rate remains within the desired limits and ideally is constant.

Hierarchical methods were proposed by Maciel [7] and Mason [8]. A hybrid approach using static techniques of distance and load balancing is used by Iris Performer [9].

### 2.1 FUNKHOUSER'S ALGORITHM

The market model is a dynamic method so for comparison we will use the most well known method from this category, Funkhouser's algorithm. In order to do that it is necessary to briefly state how that particular algorithm works.

For each frame the initial level of detail for each object is the level of detail that the object had for the previous frame (or the lowest level of detail if the object was not rendered in the last frame). Then in order to try to reach an optimal solution the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

VRST 99 London UK

Copyright ACM 1999 1-58113-141-0/99/12...\$5.00

object that has the largest Bang-for-Buck<sup>1</sup> [1,11] will have its level of detail increased. The total time taken for the frame will now have increased. In order to get the time below the required frame rate limit the object with the lowest Bang-for-Buck for its next level of detail downwards will have its detail reduced by one.

This increase/decrease stepping is performed until the object that has its level of detail increased is the same object who then has its level of detail reduced again. The algorithm then terminates for that frame. It is in essence an iterative “greedy” algorithm. For more on optimization algorithms including “greedy” algorithms see [10].

One of the biggest problems with the algorithm comes with choosing sufficiently accurate cost and benefit functions for the Bang-for-Buck measure. As shown later by the experiment (the results of which are shown in Table 2) both the version of the Funkhouser algorithm and market model algorithms overrun the time limit for rendering a single frame. If the cost function was accurate then there would be no overrunning which shows the cost function is not as accurate as it needs to be. The problem of making the cost function more accurate is difficult to solve when trying to profile a system as complex as a virtual environment system.

The other major problem with choosing a suitable benefit function is that Funkhouser does not in [1] or [11] describe the benefit function in enough detail to make it readily implemented so that duplicating Funkhouser’s actual algorithm is not possible with further information. The thesis [11] defines the benefit function as  $\text{Benefit}(O,L,R) = \text{Size}(O) * \text{Accuracy}(O,L,R) * \text{Importance}(O) * \text{Focus}(O) * \text{Motion}(O) * \text{Hysteresis}(O,L,R)$ , where O is the object being rendered, L is the level of detail and R is the rendering algorithm being used. The only term mentioned in any detail in either [1] or [11] is the accuracy term. It is also unclear as to whether the choices that Funkhouser made were the best choices as the terms are not backed up with evidence of their appropriateness for the task at hand. Since each different choice of benefit function alters the optimization problem to be solved it should not matter greatly how complex the benefit function chosen to test the algorithms is as long as the limitation that the results only apply to that choice of function is accepted.

### 3. THE MARKET MODEL CONCEPT

#### 3.1 The Market model

The inspiration for the new level of detail control algorithm was that of the concept of a trading market. The central idea was to take the conventional way of having a central algorithm that performs an optimisation algorithm to decide how much time to give to each object and turn it around. Now, just like in a trading market, each player (object) is given an initial allotment of resources (time) to trade with. For each frame the assignment of the levels of detail becomes a session of trading where each

object will try to “buy” or “sell” the time it has depending on the “need” it has for the time it currently owns.

At the beginning of each frame all the objects will calculate their “need” which will be positive if they want more time (which might occur when, for example, they have suddenly become closer to the viewer) or negative if they have spare time which they would like to “sell” to other objects (for example if they have dropped out of view).

In pseudocode the algorithm is as follows:

Correct LOD’s with respect to their new positions (with respect to their allotted times)

**For** all objects

    Calculate market need for object

**While** not finished

    Pick a trading pair

**If** no pair found **then** finished

**If** pair has object trading with itself **then** finished

    Allow the pair to make a trade

    Update LOD’s with respect to new times

**Endwhile**

**Figure 1: Market model pseudocode**

Simply put the objects which are not in view have surplus time which they sell to objects that are in view. Those with the greatest need (which to compare with Funkhouser’s “greedy” approach is the benefit divided by cost) are first to try to buy the time offered by the non-visible objects. It is possible for objects with a large need to buy from objects with a smaller need if all the objects who are not in view have already sold all their time.

So far the algorithm does not take into account the possibility of going bankrupt and trading is allowed to continue until all the trading that can happen has been performed. In practice trading mostly finishes with two objects trading the same segment of time back and forth and this is taken as an end to trading. The other end condition is when an object looking for a trade cannot find anyone else to trade with. If trading is taking a long time for any reason it would be possible to suspend trading using a time limit added to the “while” condition of trading.

The algorithm also still only runs with only one trade allowed at any one time. This makes trading is a serial operation and not a parallel one. If more than one processor was available then parallel trades could be allowed as long as some locking mechanism was added to prevent the same time being traded independently to two different objects at the same moment (which would forbid the objects to perform “confidence tricks”).

It is also the case that the algorithm let’s the trading occur only in the order where the object with the most need gets to trade first. It would certainly be possible to let the trading happen in a “random” order or choosing trading partners via a probability distribution (which could take into account things such as an analogy to customer loyalty) but it is not clear as to whether this would make the model any more successful at finding the optimal solution and, in the case of using probability

<sup>1</sup> The Bang-for-Buck for an object is the predicted perceptual benefit of that object divided by the predicted cost of rendering that object.

distributions, might lead to an algorithm that would be too slow for a real-time simulation. It would be an extension to the current work to ascertain what the effect of a probability distribution might be and whether the source scene can be used to create a distribution that would improve the results from the algorithm.

### 3.2 Funkhouser and Market compared

It is important at this point to explicitly state the differences between the market model algorithm compared with the Funkhouser method. Both methods can use the same benefit and cost functions so they can be made equivalent in that respect.

It is perhaps easiest to compare the two algorithms by showing that the Funkhouser algorithm can be reformulated as a type of market model in its own right. The algorithm has the current level of detail being the resource which the objects own and trade. When they trade they have transactions that can be grouped together as an increase in lod for one object ("buying") and the decreases needed to bring the time back under the constraint ("selling"). The difference here is that one buying can have multiple sellings associated with it. The market model, however, uses time as the resource that the objects own and the trading occurs in buy/sell pairs between objects.

In order to illustrate the difference this makes imagine a gridded benefit surface for two objects (the grid density is reliant on the number of levels of detail for the object associated with that particular axis) and a scenario part way through a run of rendered frames. Both algorithms have a solution from the previous frame. The Funkhouser solution can be visualized as a grid point on the surface whereas the market model has a point that does not need to lie on the grid that only becomes approximated to a grid point when levels of detail need to be assigned at rendering time.

As the viewpoint moves for the next frame the benefit surface changes shape and the algorithms now try to reach an optimal solution for this new frame. The Funkhouser method will start from the old grid position and optimize from that point. The market model will first adjust the lod's for the objects given the time that they have with respect to the new cost (which will have changed because of the alteration in the viewpoint). The market model will then optimize from that point.

The other major difference is that the Funkhouser algorithm viewed as a market formulation only allows trades that are full levels of detail whereas the market model trades in time (and so can effectively trade in partial levels of detail). This can be visualized as the Funkhouser method having to jump from gridpoint to gridpoint whereas the market model can move over the surface without being restricted to the grid at all.

To summarize the market model allows the start point of optimization to be time dependent not lod dependent and the optimization path up the surface need not be constrained to only the gridpoints on the surface.

The analogy of the market model also leads to the possibility that probability distributions based on market forces might be applicable in providing biases for choosing the optimal path "up" the benefit surface. These distributions could be added to the Funkhouser algorithm as well as the market model.

## 4. INTEGRATION INTO DIVE

In order to test and compare the algorithms, the "greedy" algorithm portion of the Funkhouser[1] algorithm and the market model algorithm were implemented as part of the renderer of the DIVE virtual environment system. Within DIVE the level of detail is currently chosen using the distance of the object from the viewer to determine what level to assign.

It was found when implementing the Funkhouser algorithm that there were some decisions that needed to be made above and beyond the definition of the algorithm in Funkhouser's thesis [11]. In particular there seemed to be no indication of whether objects could be dropped from the scene in order to increase the level of detail for those remaining objects. In order to prevent objects from "popping" on and off it was decided to implement the algorithm in such a way as objects could not be dropped from a scene in such a way.

Another problem that was encountered was that of trying to choose the cost and benefit functions that the Funkhouser and market algorithms rely upon. The cost function was implemented as being a direct proportion to the number of polygons for the representation of the object. This was done by running the renderer to see how long, on average, each different representation for the sphere took to render. The difficulty in choosing the cost function is exemplified by the erratic timings for the object when stationary. The problem becomes one of how much processing should be spent trying to accurately predict the rendering time. It was decided that the rough approximation used would suffice for comparison purposes between the algorithms.

The benefit function was chosen as being comprised of a term that was an approximation of the screen area taken by the object along with a scaling term which would be proportional to the current level of detail. The screen size was approximated by calculation the area that the bounding box for that object would project onto. The short-falling of this is that all the levels of detail would share the same bounding box meaning that there would be no benefit in increasing to a higher level of detail. A simple function was therefore used to scale the benefit with respect to the current level of detail after calculating the screen area that the bounding box would project onto. The function took the level of detail (lod) and scaled it with respect to the number of levels of detail available to that object (i.e. scaled lod is the current lod divided by the number of levels of detail that the object has defined for it) and then it would use that new scaled lod in the function below:

$$factor = 1 + \frac{(1 - scaled\_lod)^2}{10}$$

It could be argued that because the benefit and cost functions are not the same as those used by Funkhouser that the comparison using the chosen benefit and cost functions are not "fair". Firstly it is important to note that Funkhouser's thesis [11] does not describe in detail many of the terms he puts into the benefit function that he uses and without that knowledge there is no way to duplicate his algorithm exactly. It is also true that with each different choice of cost and benefit function the surface upon which we are trying to pick an optimal point will change shape. This leads to the obvious conclusion that any comparison made

between level of detail algorithms is only valid in comparing the algorithms with respect to the choice of benefit and cost functions along with the scene (number of objects and numbers of levels of detail for each and their positions) and the path taken through that scene. Any change in the functions or the scene and the path taken through it could lead to a different algorithm being the best.

## 5. THE EXPERIMENT

In order to test the market method for level of detail assignment against the Funkhouser and distance algorithms an experiment was run. The renderer was set to run each of the algorithms by moving the viewpoint for the renderer along a predetermined path between 10 points. These points were chosen to make sure that the path would view all sides of the scene and have sections where no objects would be visible and some sections where all the objects would be visible.

The scene used was composed of 32 spheres in a matrix formation of 4x4x2 (X by Z by Y). Each of the spheres was 2m in size and the translation between each sphere in any axis direction was 5m. The spheres themselves were comprised of 5 different levels of detail each of which would have a different colour to allow easy identification of the current level of detail for an object. The table below shows the numbers of polygons for each representation and also the colour that the representation was given along with the distance beyond which that particular level of detail would be used (for the distance method only).

Level of Detail	Number of	Distance	Colour
-----------------	-----------	----------	--------

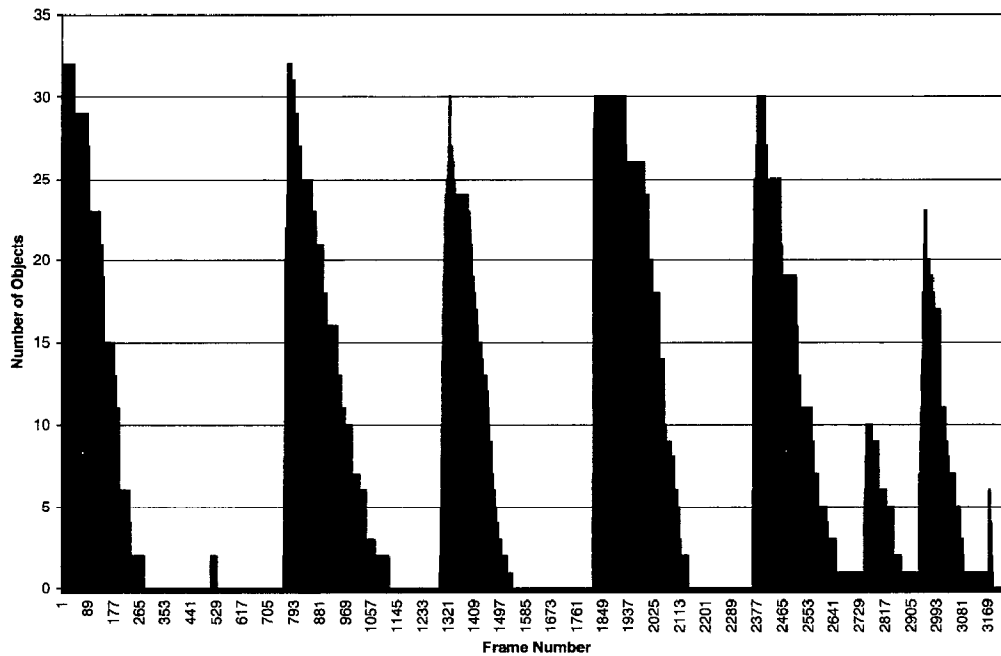
	Polygons	(m)	
0	256	16.4	Red
1	216	29.3	Orange
2	191	55.1	Yellow
3	85	106.8	Green
4	12	209.5	Blue

**Table 1: Sphere details**

The five levels of detail for the sphere object and the distances at which they should be used were created using a program called LodeStar [12].

The time to render each frame and the time to perform the level of detail assignment algorithm was recorded. To show that all the level of detail control algorithms were an improvement over no level of detail control another run was made where all the objects were assigned their highest level of detail.

The test would be to see whether the algorithms could keep a steady frame rate and what the visual quality of the scenes would be. Even though the scene was simplistic and only used objects of one type it was hoped that the path through the spheres would still give a wide variety in the number of objects viewed. This included times where no objects could be seen and times when lots of objects would appear at once (see Figure 2). Obviously further work would be needed to compare the algorithms for different size scenes with different types of objects to produce a result more representative for an “average” scene.



**Figure 2: The number of objects throughout the rendering sequence**

Algorithm	No. overrun frames	Total overrun time (secs)	Average overrun (secs)	Max. overrun time (secs)	Total algorithm time (secs)	Benefit total
Highest	775	31.98	0.0413	0.095	0.00	16776
Distance	679	17.35	0.0256	0.081	0.65	16770
Funkhouser	41	0.25	0.0061	0.042	1.00	14016
Market model	116	0.73	0.0063	0.040	2.30	16753

Table 2: Experimental results

## 6. RESULTS

The four different level of detail control choices were run for the predefined route. The rendering for that route produced a sequence of 3246 frames with each frame being given 0.1 seconds to render in. This meant that the experiment was almost five and a half minutes of rendering.

The number of objects on the screen during this time is shown in Figure 2. Out of the 3246 frames 1197 frames had no objects to be rendered.

As can be seen from the shape of Figure 2 the numbers of objects tends to rise sharply then drop off more slowly but this is probably because of the peculiar shape of the scene. The average number of objects in view over the 3246 frames is 8.5 which is about 25% of the objects in the scene.

In order to compare the respective algorithms' performances the time taken to render each frame, as well as how much time the algorithm took to select the levels of detail was recorded along with the total "benefit" for that frame. Table 2 shows the results from the runs with each algorithm

As can be seen from the table the amount of time that the renderer would overrun, if the highest level of detail was used for all the objects all the time, would be almost 32 seconds. Using the distance algorithm would reduce this to just over 17

seconds whilst both of the other algorithms reduce the time overrun to under 1 second.

The table clearly shows that the Funkhouser algorithm and the market model algorithm are both far superior level of detail allocation algorithms compared to using the distance method. It is of course possible that the choices for the distances that the LodeStar program chose for the different levels of detail were not optimal but it is beyond the scope of this paper to judge and profile LodeStar.

In comparing the Funkhouser algorithm with the market model it is hard to make a judgement as to which is more successful at assigning levels of detail. The total benefit from the market model is much higher than that of the Funkhouser results but it is at the expense of a greater overrun in time. It is clear that the market model is more sensitive to the inaccuracies of the cost function as neither algorithm should overrun on any of the frames if the cost function had been an accurate predictor of rendering time.

Figure 3 shows the rendering times of each of the four algorithms over the whole period of the experiment. The Funkhouser and market model both perform much better than either of the other two methods but it is not clear from this run which is superior when comparing them to each other.

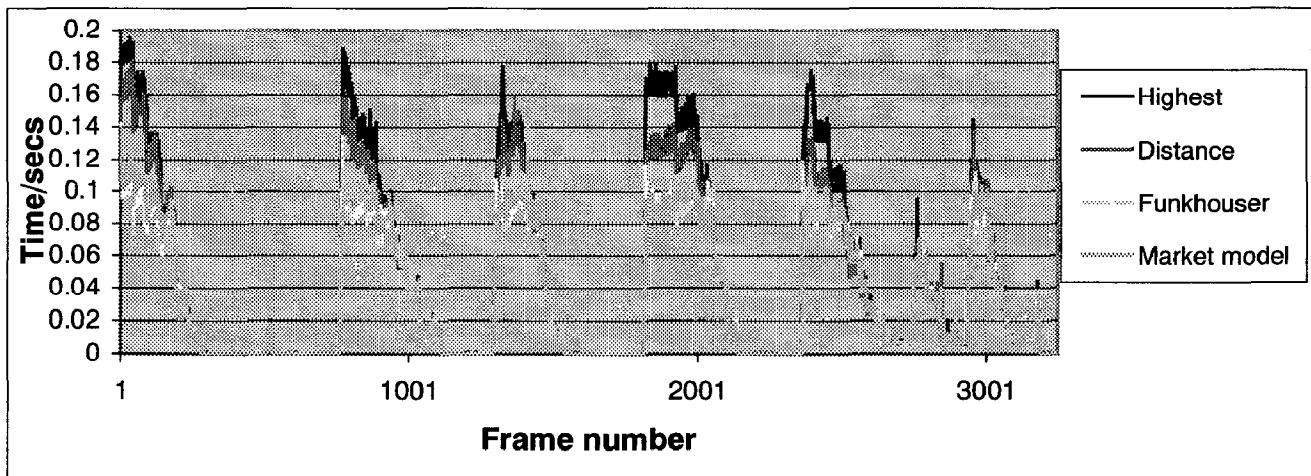
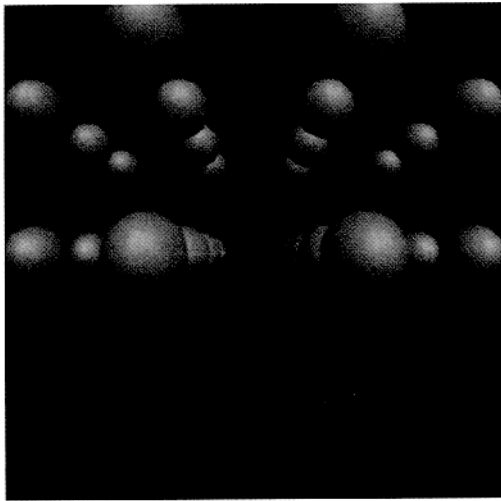
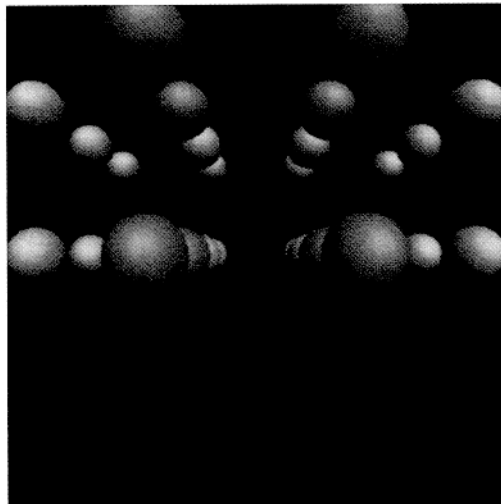


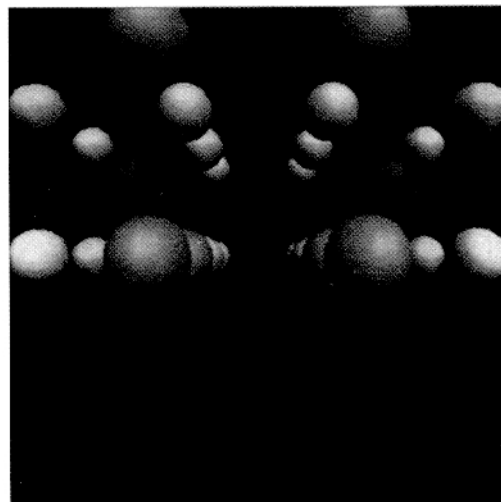
Figure 3: Rendering times for the whole experiment



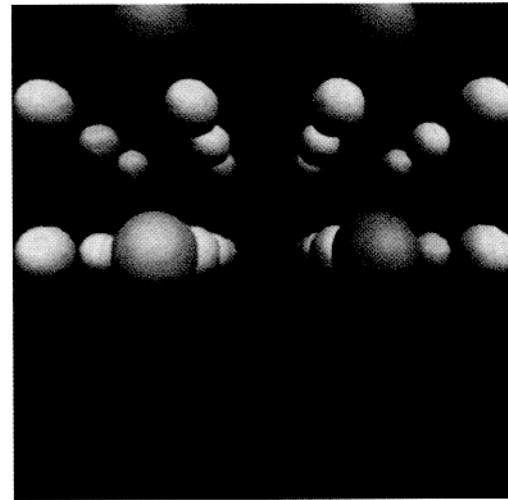
**Figure 4: Highest levels of detail**



**Figure 5: Distance algorithm**



**Figure 6: Funkhouser algorithm**



**Figure 7: Market model algorithm**

Another way to compare the two algorithms is to see the variance that the overrunning times have. If the variance is low then the overrunning time could be taken into account by just reducing the allotted amount of frame-time given to each frame. The standard deviation for the two algorithms (taking only those points over the 0.1 time allotted to each frame) is 0.0103 for the Funkhouser algorithm and 0.0086 for the market model.

Perhaps the best way to interpret the results is to actually look at an example of a frame rendered in the experiment. The four images (Figures 4 to 7) show each of the four different choices of level of detail assignment at exactly the same frame in the walkthrough route of the rendering. The first image shows the frame with all the objects at their highest level of detail. The next image shows the result of assigning the levels of detail with respect to the object's distance from the viewpoint. The third image shows the result of using the Funkhouser algorithm with the fourth image showing the results of the market model algorithm.

As can be seen the market model frame has a much more even spread of time allocated to the objects although the only big noticeable difference in quality comes with the spheres at the back of the frame which in the Funkhouser rendered frame have blue spheres (which are so simply represented that they fail to even look spherical at all).

Other visual results include the "popping" of objects from one level of detail to another. In all of the algorithms there was a problem with objects that were new to the frame "popping" but this is due to the placing of the level of detail control algorithm within the graphics pipeline. The trouble is caused by the pipeline using the renderer's last frame to determine the visibility for the level of detail control algorithms. This means that objects can appear for one frame (with the level of detail they were last assigned before they "dropped" from being rendered) before the level of detail control algorithm knows about them. The renderer will therefore take one frame before new objects appear at a level of detail that is actually chosen by the algorithm for that current frame.

More seriously there are cases in the Funkhouser and market models where objects that have the same benefit and cost as each

other can, for a few frames, “pop” between levels of detail. This was avoided by Funkhouser in his thesis[11] by using a term in the benefit measure that gave extra benefit for objects that stayed at the same level of detail the same for several frames. This was not implemented for this experiment but that will not affect the implications of the results obtained within this paper.

## 7. CONCLUSIONS AND FUTURE WORK

In this paper a new level of detail control algorithm has been suggested. The algorithm has been based on the principals of a trading market and has been implemented to work within the DIVE virtual environment system. The algorithm works using a finite number of levels of detail but could be extended to work with “continuous” levels of detail such as progressive mesh techniques [13], [14]. There could well be difficulties in calibrating the benefit functions for objects with so many different possible visible representations. Certainly further work would be needed to see whether the market method could be realistically applied to systems that allow a “continuous” level of detail to be created for objects.

A version of the Funkhouser algorithm has also been implemented within the DIVE system but the benefit and cost functions used were not as complex as those used by Funkhouser himself [11]. It was also suggested that the Funkhouser algorithm itself could be viewed as a type of market model.

An experiment was run to show the potential of the new algorithm which compared favorably with the Funkhouser algorithm. The experiment that was run used a fairly unrealistic scene but this was primarily due to the shortage of readily available virtual reality objects that have multiple levels of detail. It does, however, suggest that the market model displays some useful behaviors under practical conditions, such as having a low variance when overrunning frame time limits. It also appears to give a good trade-off between time overrun and the total benefit obtained.

The implementation is currently under revision to perform some necessary alterations as well as test other conditions. The market transactions will be made between random pairs of objects with a probability distribution based on the market needs they have. It is also necessary to extend to code to handle scenes where there are more objects in the hierarchy than the renderer can even traverse in one frame (let alone render). A version of code to limit the number of objects that are sent to the renderer has been implemented on a rendering simulator but not in the DIVE renderer as yet.

## 8. ACKNOWLEDGMENTS

Thanks must go to the Digital VCE whose funding has made this work possible.

## REFERENCES

- [1] Thomas A. Funkhouser and Carlo H. Sequin. Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments. *In ACM Computer Graphics Proceedings Annual Conference Series*, volume 27, August 1993, pages 247-254.
- [2] Emmanuel Frécon and Mårten Stenius. DIVE: A Scaleable Network Architecture for Distributed Virtual Environments. *Distributed Systems Engineering Journal*, Vol. 5, No. 3, 1998, pages 91-100.
- [3] G. J. F. Smets and K. J. Overbeeke. Trade-off Between Resolution and Interactivity in Spatial Task Performance. *IEEE Computer Graphics and Applications*, September 1995, pages 46-51.
- [4] J. Clark. Hierarchical Geometric Models for Visible Surface Algorithms. *Communications of the ACM*, 19 (10): 547-554, 1976.
- [5] E. Puppo and R. Scopigno. Simplification, LOD and Multiresolution: Principles and Applications. *Eurographics '97*, Tutorial T4, Budapest, Hungary, September 97.
- [6] P. S. Heckbert and M. Garland. Multiresolution Modeling for Fat Rendering. *In Proceedings of Graphics Interface '94*, pages 43-50, 1994.
- [7] P. W. Maciel and P. Shirley. Visual Navigation of Large Environments Using Textured Clusters. *In 1995 Symposium on Interactive 3D Graphics*, pages 95-102, April 1995.
- [8] A. E. Mason and E. H. Blake. Automatic Hierarchical Level of Detail Optimization in Computer Animation. *Computer Graphics Forum*, Vol. 16, 1997, pages 191-199.
- [9] J. Rohlf and J. Helman. Iris Performer: A High Performance Multiprocessing Toolkit for Real-time 3D Graphics. *SIGGRAPH 94*, pages 381-394, July 1994.
- [10] Silvano Martello and Paolo Toth. Knapsack Problems: Algorithms and Computer Implementations. John Wiley and Sons Ltd, 1990.
- [11] Tom Funkhouser. Database and Display Algorithms for Interactive Visualization of Architectural Models. University of California at Berkeley, EECS Computer Science Division, September 1993.
- [12] Dieter Schmalstieg. LODESTAR: An Octree-Based Level of Detail Generator for VRML. *In the Proceedings of VRML 97*.
- [13] Hugues Hoppe. View-Dependent Refinement of Progressive Meshes. *SIGGRAPH 97*, pages 189-198, August 1997.
- [14] David Luebke and Carl Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. *SIGGRAPH 97*, pages 199-208, August 1997.

## APPENDIX

A more detailed pseudocode version of the market model algorithm:

```
For all objects //adjust lod of objects to max within their time
    While time for current_lod > allotted time
        Reduce lod
    While time for current_lod < allotted time
        Increase lod
Endfor
For all objects // set the market need of each object
    If object is at highest lod
        Set market_need to 0
    Else
        Set market_need to Bang-for-Buck
    If object not visible
        Set market_need to -1.0*alloted time
    Else if without enough time for lowest lod
        Set market_need to a very large number
Endfor
While not finished // do the trading
    maxneed, minneed = need for first object in array
```

```
For all objects // find objects with max/min need
    If need for current object > maxneed
        maxneed = current need
    If need for current object < minneed and there's
        spare time to render object at current lod
        minneed = current need
Endfor
// find how much time is required and on offer
required = time to increase object with maxneed to
its next lod
offer = spare time that the object with minneed has without
dropping an lod
If offer is 0 // no time left to offer so finish
    Finished trading
Else if offer > required // object can take all it needs
    Trade 'required' time
Else // object can only take what is on offer
    Trade 'offer' time
// update the market needs and lod's for the 2 objects
Update market need for the two objects that have traded
Correct lod's for the two objects using their new times
Endwhile
```