LODManager: a framework for rendering multiresolution models in real-time applications

J. Gumbau
Universitat Jaume I,
Castellón, Spain
jgumbau@uji.es

O. Ripolles
Universitat Jaume I,
Castellón, Spain
oripolle@uji.es

M. Chover
Universitat Jaume I,
Castellón, Spain
chover@uji.es

ABSTRACT

Many papers have addressed the problem of achieving real time visualization in interactive applications where millions of polygons are rendered and many objects are visualized. Multiresolution modeling has proven to be a good solution, as it diminishes the quantity of geometry to render. But this solution is not widely used because it presents inefficient level of detail update routines that lower the overall performance. We are introducing a set of techniques to adapt the level of detail while meeting time constraints and maintaining image quality. In order to fulfil the requirements of current game engines, the LODManager considers exploiting graphics hardware and reuses as possible those levels of detail already calculated. Finally, we will show the integration of our LODManager in a game engine and we will prove the validity of our solution in an interactive application.

Keywords: Real-time rendering, level of detail, scene management

1 INTRODUCTION

In recent years, computer graphics have experienced an intense evolution as new graphics hardware offers a final image quality that was totally impossible to imagine a few years before. This way, interactive graphics applications, such as computer games, virtual reality environments or CAD applications, include more complex scenes to offer very detailed environments.

The necessity of highly realistic scenarios often involves including many polygonal meshes made up of a high number of triangles, which poses a problem for maintaining interactivity. In these applications, it is important to guarantee stable frame rates while reducing perceived lag [15]. The lag, which is the delay between performing an action and seeing the result of that action, is as important as the frame rate to perceive interactivity in an application.

One of the possible solutions to this problem is the use of continuous level-of-detail techniques to maintain a balance between image quality and rendering speed. Nowadays, multiresolution modeling can be considered as a compulsory feature of libraries and game engines. In this sense, graphics libraries like OpenInventor or OSG, and game engines such as Torque or Ogre, introduce multiresolution models to easily alleviate the amount of geometry that must be rendered in a scene, thus resulting in an improvement in performance. Most of them use static heuristics, like the distance or the screen-space area, as the metric to select the suitable level of detail. Other works like [1] add a criterion based on the occlusion information to obtain a tighter estimation of the contribution of each object to the scene. These heuristics, despite improving frame rates, are usually not enough. They cannot guarantee stable frame rates and often present jerky frame rates, as they are not adaptive and cannot work correctly in scenarios where objects are moving in and out of the scene or where the objects become bigger or smaller quickly.

In order to improve the results of the static heuristics, some authors have introduced the use of feedback algorithms, which take into account the past rendering times. These algorithms, even though are more adapted to the rendering conditions, also suffer from oscillation and unavoidable overshoot when rendering discontinuous environments. They present a good alternative for scenarios where there's a large amount of coherence between frames, as it happens with flight simulators. This is the case of the solution presented in [6], which provides temporal coherence through the runtime creation of geomorphs to control de level of detail.

Funkhouser and Séquin [4] demonstrated that it is necessary to use a predictive selection scheme, based mainly on the complexity of the current frame, rather than a reactive framework, based on the feedback obtained. They formulated this problem as an optimization task which is equivalent to a constrained version of the Multiple Choice Knapsack Problem. Even though this problem is NP-complete, some authors like [4] or [9] obtained several techniques that could only guarantee a solution that is at least half as good as the optimum one. [14] reconsidered this problem for the special case

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, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency – Science Press, Plzen, Czech Republic.

of continuous multiresolution models, obtaining a non-iterative closed form solution which was cheap to evaluate for every frame.

This way, the problem of the time-critical multiresolution rendering can be presented as an optimization problem for finding the LOD that maximizes the scene quality under timing constraints. Funkhouser and Séquin [4] developed a generalization of the predictive approach, using approximate heuristics of the cost and the quality obtained that were efficient and accurate enough to obtain the best image possible within the target frame time. The work in [5] extended the use of predictive techniques with more precise heuristics for the cost and the benefit of the resolution of the objects. It also considers temporal coherence to minimize sudden changes, although the authors did not include it in their tests. These optimizations are very accurate but costly, and as they assign one variable for each object, rendering scenes with a large number of objects tends to be a slow solution.

All the previous approaches apply different kinds of heuristics: static, feedback or predictive. But, in all cases, it is necessary to use a criterion to select the most adequate level of detail. This way, it is possible to use the size, the speed, the position in the scene, etc. Many authors have addressed the necessity of investigating how the human perception system works. [11] considers the necessity of including an analysis of the human visual system to understand how it works and to offer more adequate results, extending his results in his subsequent publications. In this sense, several authors have included biometrics into their heuristics, considering spatiotemporal sensitivity [17] or developing frameworks with eye tracking as the basis [2].

Other authors have addressed this problem from different points of view. The approach presented in [13] uses a multiresolution hierarchy based on bounding spheres with a rendering system based on points specially designed for 3D scanned models with a great geometric complexity. They perform the LOD selection based on the projected size in the screen, and adjust the threshold from frame to frame. They also gradually refine the model when the viewpoint is not moved for a period of time. In [3] it is presented a hierarchical solution which represents the environment with a scene graph and automatically calculates the different approximations of portions of the scene graph. Different researchers have presented architectures to solve this problem, like [7], which use a distributed rendering architecture to obtain a stable frame-rate, or [8], which proposes a parallel architecture combined with levels-of-detail and occlusion culling techniques. The most novel aspect of [16] is the concept of interruptible rendering, which finds a rational compromise between spatial and temporal detail. They produce a complete image in the back buffer almost immediately and then incrementally refine it so that the refinement can be interrupted at any time. Zach [18] presents a solution based on geomorphing where the LOD management is achieved by distributing the LOD selection and calculation between several frames, reusing the old resolution until the new one is ready. As the new LODs will appear in future frames, they need a path prediction process to obtain future viewpoints an directions. They also use cost and benefits computation, but include some feedback strategy to compensate for some assumptions they make. These authors extended their work in [19], presenting an approach for discrete and continuous models where the time spent for LOD selection is amortized over several frames.

This paper presents the following structure. Section 2 presents the motivation for developing this LODManager. Section 3 contains an overview of the approach we are presenting. Section 4 discusses the architectural design of the LODManager. Section 5 presents the results obtained and sketches briefly the framework where this LODManager was tested. Lastly, Section 6 contains comments on the results and outlines the future work.

2 MOTIVATION

Many of these articles were written in the early days of the GPUs (or even in earlier times [4]) when it was advisable to spend some CPU processing time to optimize the GPU rendering process. Nowadays, due to the great scalability of the graphics cards, we must revise all that previous work to provide an updated and practical viewpoint of that situation: overloading the CPU is a delicate task that in most cases will cause it to be a bottleneck for the graphics hardware.

All the previous works have in common that, to optimize the GPU usage, they apply complex heuristics that have an important CPU penalty. This issue is specialy problematic when dealing with scenes with lots (some hundreds or even thousands) of LOD objects. This way, these solutions tend to be CPU bounded, limiting the gross horsepower of the GPUs. In addition, many of them present high memory overheads, while others guarantee image quality but not a stable frame rate. Furthermore, many of the papers which present hierarchies or pre-calculate LODs are not suitable for dynamic scenarios as they are aimed at environments with infrequent motion.

Therefore, the aim of this method is to offer real-time rendering of dynamic scenes, by means of a LOD manager with very low CPU requirements, freeing the CPU by minimizing the number of real changes in levels of detail. We provide a simple while effective method that lowers the CPU usage in order to keep the bottleneck on the GPU. This work also uses the concept of frame rate feedback to automatically adapt the level of detail of the scene to achieve a target user-defined frame rate.

Figure 1: Left: A LOD scene composed by 3000 LOD objects. Right: A LOD forest populated with 100 highly detailed trees.

This approach (explained in section 4.2) offers more interesting results as it is an adaptive solution. Adaptive display has been lately presented as the most suitable solution to maintain a balance between accuracy and interactivity, while minimizing the CPU usage as much as possible.

3 OVERVIEW OF THE APPROACH

Continuous multiresolution LOD models have always had an associated extraction time. This is defined as the time needed to compute the new level of detail and to make it ready to be rendered. For this reason, changing the level of detail of a high amount of LOD objects independently tends to be completely unefficient.

The aim of the work presented here is to provide a framework on which a scene populated with a large number of multiresolution objects (like a crowd in a scene or the vegetation in a forest) can be managed efficiently. This management is based in the fact that a multiresolution scene can contain a number N_T of multiresolution objects of the same type T. When this occurs, there is a certain possibility that two or more objects can share a similar level of detail. This similarity S factor will be explained later. The objective of the LODManager is to minimize the number of changes in levels of detail to avoid recalculations. Therefore, the LODManager must be able to reassign previously calculated levels of detail.

To make this feature effective, the LOD objects must implement a *fast LOD switching* functionality to allow a low-cost update of their active rendering geometry. For example, this can be accomplished in a real game engine by changing the object's active index buffer by the one supplied by the LODManager. Thus, a LOD object can hold its own level of detail or a *borrowed* one.

Using this technique the number of changes in levels of detail can be minimized depending on the similarity factor and on the quantity of objects that need to change to a particular precalculated LOD, so that they can share it.

The main contribution of this approach is that the heuristics proposed in this article, despite being simpler, are faster and effective to manage big scenes with lots of several LOD objects virtually changing its LOD at the same time. Moreover, the method of sharing already calculated LOD data is completely GPU-compliant (because in practice LOD objects share precalculated index buffers) which lowers the CPU usage for LOD calculations in this kind of scenes.

The core technique of this method is the discretization of continuous LOD data on-the-fly by maintaining a user-defined number of different discretizations. These discrete levels of detail are recalculated in real-time using a continuous LOD algorithm. The objects in the scene can decide if they should use one of the available levels of detail or if it is better to calculate their own LOD. Therefore, this technique offers a continuous range of approximations while exploiting the similarity between the objects sharing levels of detail. This method will be explained more deeply in section 4.

4 ARCHITECTURE

Let V be an array where the LODManager stores references to all LOD objects in the scene. These objects might be of different types. Two objects are said to be of the same type T if they use the same geometry. We create an array D_T of a user defined length N_T for each type of object. These arrays will store the discretizations at different levels of detail of that type of object. We assume that the level of detail is defined by a LOD factor between 0 and 1, where 0 and 1 represents the minimum and maximum levels of detail, respectively.

More precisely, a position i in the array D_T contains a discrete LOD associated to level of detail $i/N_T \in [0,1]$. For example, assume an array D_T of length 2. The first position in the array would represent a discrete snapshot for the range $[0,0.5)$ and the second position an snapshot for the range $[0.5,1]$.

 ISBN 978-80-86943-02-2

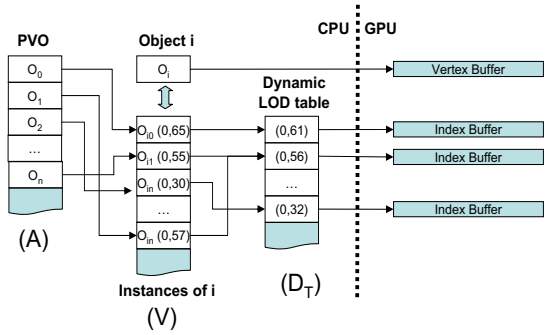

Figure 2: The LODManager architecture.

This discretization of levels of detail defined by N_T also defines the similarity factor the algorithm will use to decide when two objects are similar enough to share the same level of detail. Two objects at two different LODs are *similar* when they can be stored in the same position of D_T. More formally, we define a similarity boolean function S as:

$$S(O_1^T, O_2^T) \leftrightarrow trunc[lod(O_1^T) \cdot N_T] = trunc[lod(O_2^T) \cdot N_T]$$

where $lod(O) \in [0,1]$ returns the LOD factor at which an object O is represented.

We can also define the similarity between two LOD factors in the same way:

$$S(lodf_1, lodf_2, T) \leftrightarrow trunc[lodf_1 \cdot N_T] = trunc[lodf_2 \cdot N_T] \quad (1)$$

Figure 2 presents the LODManager architecture. It shows a snapshot state of the LOD managing system: some objects in the scene (V) use some discretizations (from D_T). It can be seen that only those objects belonging to the Potential Visible Objects list (A) are being referenced as current active objects. Each element of D_T points to an active index buffer in GPU memory that represents the object at a certain LOD.

4.1 Algorithm

First of all, an array D_T is created for each different type of objects in V. Each position of each array D_T is initialized to $\emptyset$.

At each Update step, the LODManager iterates $V = (V_0, V_1, ..., V_n)$. Each object V_i has associated a type T and a desired target LOD factor ($dlod(V_i)$). The way this desired LOD factor is calculated is explained in section 4.2. Thus, for each element V_i:

1. Discard any LOD change if the object has a *similar* LOD factor compared to the desired one. This comparison is performed using the similarity function described in equation (1).
2. If the LOD must be changed:
   (a) Find an object F_T which has a similar level of detail. This is done by accessing at the position $trunc[dlod(V_i) \cdot N_T]$ at the array D_T (where T is the type of the object V_i).
   (b) Compare the similarity of lod factor of F_T to $dlod(V_i)$. This can be done using equation (1).
   (c) If they are NOT similar enough:
      i. Change the level of detail of the object V_i
      ii. Update $lodf(V_i)$ with the new LOD factor.
      iii. Calculate the position in D_T depending on $lodf(V_i)$ (using this formula: $trunc[lodf(V_i) \cdot N_T]$). After this step, that position of D_T points to the object V_i.
   (d) If they are similar enough:
      i. Make V_i to use the level of detail already calculated by F_T. In practical terms it means to make V_i use the *index buffer* of F_T.
      ii. if $V_i \in D_T \rightarrow$ Remove V_i from D_T.

It is important to note that the implementation must be aware of sharing index buffers. When an object borrows the index buffer from another it is important that the original object stores a reference to its own index buffer. That's because when an object changes its level of detail, it must update its own index buffer, not the borrowed one.

The number of elements in D_T affects the performance as it represents the number of discretizations available, and therefore affects the number of LOD recalculations. The user can freely increase the number of elements in D_T for finer granularity of the similarity comparisons, as the spatial cost can be negligible as it only stores references to objects.

4.2 Heuristics

The active objects list. In order to simplify the algorithm explanation, we have assumed that the algorithm iterates through the whole list of objects in the scene (V). However, when dealing with a high number of LOD objects, iterating through all those elements tends to be unefficient. To solve this problem, a list of active (or *more important*) objects (A) can be maintained. For deciding which objects must be included or excluded from the active objects list, we create an initially empty list A of a user-defined constant size N_a. Every object V_i has a value assigned depending on the difference between the ideal LOD and the real LOD $lodf(V_i)$. Objects outside the frustum are given a penalization factor to ensure that the algorithm potentially discards them. Thus, objects outside the frustum will only have the opportunity to be included it they are near enough to the camera, or in other words, if they are about to be included in the frustum.

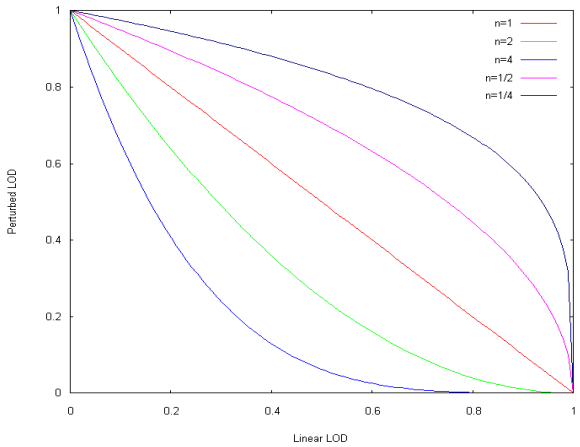

Figure 3: Perturbation function to calculate the desired LOD factor to adapt it to the current frame rate. For example, $n = 2$ will cause a global reduction in the desired LOD factors.

The desired LOD factor. We define the *desired LOD factor* (or $dlodf(V_i)$) associated to a LOD object as the ideal LOD the object needs to change to, depending on certain heuristics. We define it as ideal because an object V_a can takes an already precalculated LOD, by similarity, from another object V_b, that may have not exactly the same LOD factor.

To calculate $dlodf(V_i)$ we use an heuristic that takes as input the distance of the object to the camera and the current application frame rate. The distance to the camera defines a linear function that is mapped to the range $[0,1]$, as shown below:

$$lod = -\frac{dist_{cam} - range_{near}}{range_{far} - range_{near}}$$

This value is clamped to the range $[0,1]$ and is perturbed depending on the current frame. The aim is to use the frame rate as a feedback parameter to alter the linearity of the LOD function to fulfil that: 1) if we are running under the desired frame rate, we must use coarser levels of detail; and 2) if we are running above, more objects will increase their LOD. We use the two perturbing functions $dlodf = lod^n$ and $dlodf = lod^{1/n}$ to increase or decrease, respectively, the global desired level of detail. The higher the parameter n, the faster the objects will increase or decrease their LOD. An illustrative picture is shown in Figure 3.

4.3 Non-linear precalculated LOD intervals

We have assumed for clarity that the vector D_T, which stores snapshots of previously calculated levels of detail, has a linear distribution. However, more real applications will prefer to use a non-linear distribution to allow for much finer LOD changes for closer models and much coarser LOD changes for objects that are far away from the viewer.

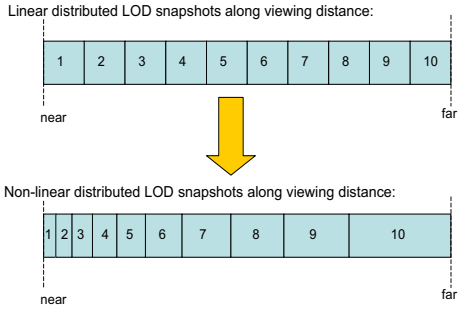

Figure 4: Linear vs non-linear LOD snapshots distribution

This distribution function can be customized by the user so that it can be used in very different client applications and situations. An illustration can be seen in Figure 4.

This optimization will reduce popping effects because queries for closer models will be classified with less granularity. It is important to notice that equation 1 should be adapted to the new snapshot distribution.

5 IMPLEMENTATION AND RESULTS

5.1 Library usage

We have implemented this method as a library which is independent of the underlying multiresolution model used to represent the objects. There are only some requirements that the multiresolution model must fulfil. These requirements are:

- The objects must provide an interface to change their level of detail. This interface must be implemented using the range $[0,1]$ as the active LOD range.
- The objects must be able to implement a *fast LOD switching* functionality. In practice, this can be done by *borrowing* index buffers from other objects while keeping the original index buffer for further LOD calculations.

Our implementation provides a `LodObject` class interface that provides some virtual functions the multiresolution that the models must implement. Thus, it is really simple to handle several types of different multiresolution models inside the same scene.

5.2 LOD Models

In our implementation we have used two different multiresolution models: one for general meshes called LodStrips [10] and another one designed to handle the foliage of plants and trees, which we will call LodTrees [12].

 ISBN 978-80-86943-02-2

LodStrips is a multiresolution model based on triangle strips. It efficiently defines a continuous sequence of level of detail changes from a base mesh. It is a index-based multiresolution model, i.e. it calculates the current index set for a defined level of detail, without affecting to the vertex list.

LodTrees is a multiresolution model used to handle a continuous level of detail management for the foliage of trees and plants. It is based on a leaf-collapse operation in which each simplification step removes two leaves and replaces them by a new single leaf that keeps the appearance.

Both models require a certain amount of time to change the level of detail, depending on how much change must be accomplished, and they can easily implement the *fast LOD switching* functionality described in section 3. Therefore, they are valid base multiresolution models to demonstrate the usefulness of our manager.

5.3 Tests and results

We have used two different polygonal models for the tests. The Ogre mesh features 1960 triangles and its minimum level of detail reduces the triangle count to a 10%. It implements the LodStrips algorithm (briefly described in section 5.2). The Tree mesh represents an *Olea europaea* with 97133 triangles at full level of detail; it uses the LodTree algorithm to reduce its triangle count to 10% at its minimum level of detail.

Two different tests are proposed: a performance test which measures the performance boost when using the LODManager, and a visual quality test that will prove the visual acceptability of the method. The test machine has been an Athlon 64 3500+ with 1 Gb RAM and a GeForce 6800 Ultra.

Performance test We have used two different test scenes. The first scene has been populated with 3000 independent LOD objects of the Ogre mesh, shown in Figure 1. The second scene adds 100 tree LOD objects to the previous one to show how the algorithm can deal with heterogeneous scenes.

All demos move the camera through a predefined path. Figure 6 shows the frame rate comparison enabling and disabling the LODManager in both scenes. Thus, the improvements in performance offered by the LODManager can be easily measured. In addition, this Figure offers the number of triangles rendered during the walkthrough. These graphs are a good help to understand the frame rate obtained, and also proves how the number of triangles rendered with and without the LODManager is nearly the same, proving that our solution offers higher frame rates while maintaining a similar visual quality.

The two graphs on top of Figure 6 show a similar pattern: the LODManager efficiently manages level of detail changes and minimizes the CPU consumption due to the LOD management. In fact, when dealing with scenes with a high count of independent LOD objects (like in the scene of the 3000 ogres), the CPU consumption dedicated to LOD changes can become the bottle neck of the application reducing the performance to make it unsuitable for interactive content.

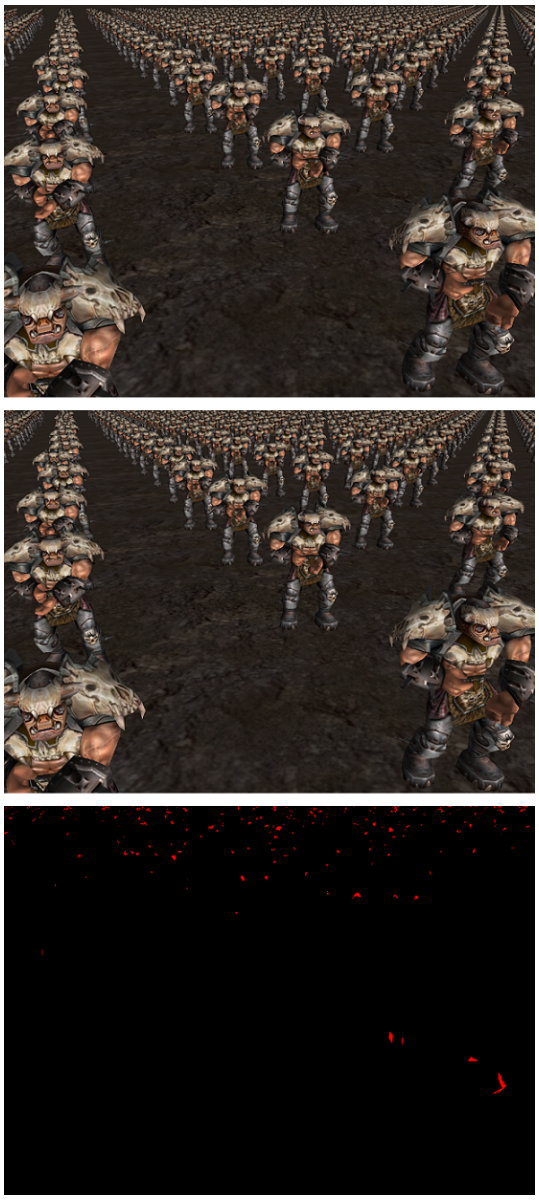

Figure 5: Top: screenshot of a scene using the LODManager. Middle: screenshot of the scene with the LODManager disabled. Bottom: per-pixel differences between the other two pictures

Visual quality test We have provided some performance tests where our LODManager proves its usefulness in Lod scenes populated with lots of independent LOD objects. Now we will show that our heuristics do not affect the visual quality of the models in a significant manner. The top image in Figure 5 shows the scene populated with ogres using our technique to man-

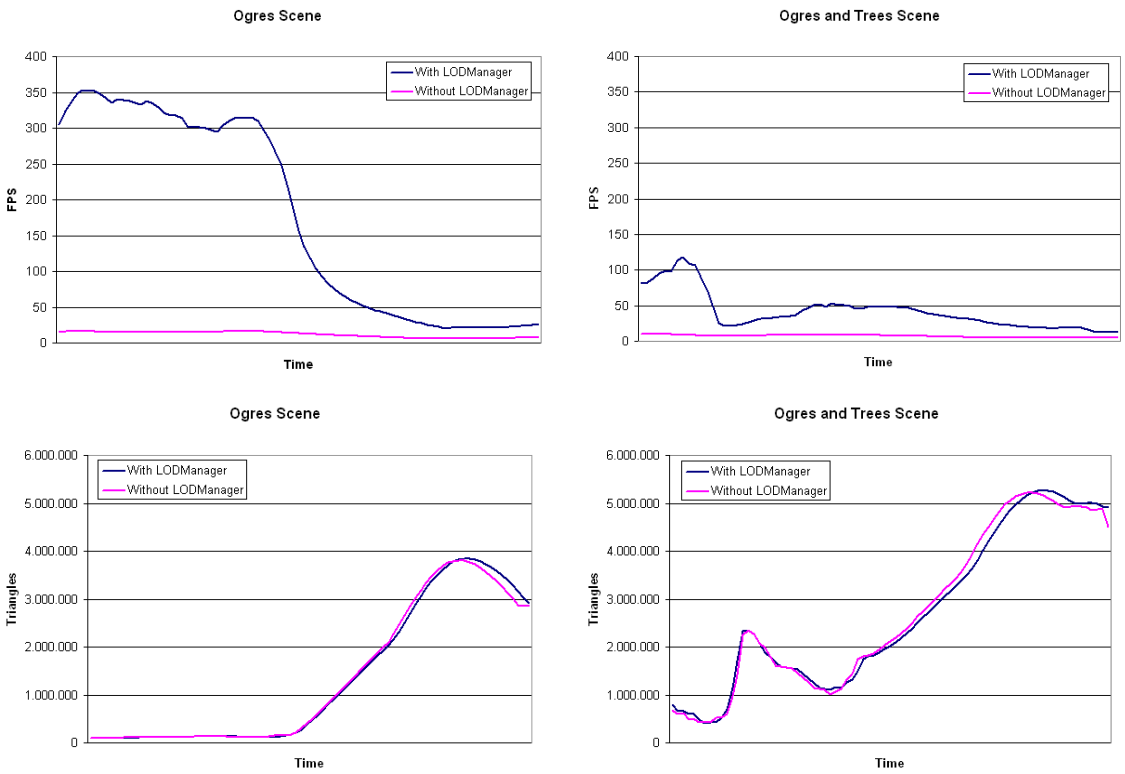

Figure 6: Performance comparison with and without LODManager in two different scenes, showing the FPS and triangles rendered throughout the scenes.

age the level of detail of the whole scene. The middle image shows the same scene without any LOD management approach active, i.e. each independent object treats its own level of detail independently. The differences caused by our method are shown in the Figure 5, where a red pixel shows a difference between the two images. We can see that the visual differences are almost imperceptible.

6 CONCLUSIONS

We have introduced a new technique to minimize the number of level of detail changes of a scene populated with a high count of LOD objects. This technique allows the reuse of LOD calculations to minimize the CPU computation time. In section 1 we have analyzed some methods which use more complicated heuristics than ours, and thus, require more computation time. Our algorithm also features a feedback heuristic that is able to globally reduce or increase the LOD of the scene to achieve a user defined frame rate.

Nowadays, the great scalability of the graphics processor units has contributed to make them far more powerful than the CPUs. Thus, real world applications tend to be CPU bound and the GPU becomes limited by the CPU power. It is more useful a technique that saves CPU time as well as providing an real world acceptable LOD management, rather than more sophisticated techniques that consumes CPU to save GPU cycles. This is specially true when dealing with scenes with a high number of LOD objects, where predictive methods tend to be completely unsuitable for real time applications.

Even though our technique has been designed to be much simpler than predictive heuristics, it has proved to be simple to implement and effective to minimize CPU consumption, to manage heterogeneous LOD scenes and to help maintain a target user-defined frame rate.

ACKNOWLEDGEMENTS

This work has been supported by the Spanish Ministry of Science and Technology (TIN2004-07451-C03-03), the Spanish Ministry of Science and Education (FPU grants), the European Union (IST-2-004363) and FEDER funds.

REFERENCES

[1] Carlos Andújar, Carlos Saona-Vázquez, Isabel Navazo, and Pere Brunet. Integrating occlusion culling with levels of detail through hardly-visible sets. *Computer Graphics Forum (Proceedings of Eurographics '00)*, 3:499–506, 2000.

[2] R. Danforth, A. Duchowski, R. Geist, and E. McAliley. A platform for gaze-contingent virtual environments, 2000.

[3] Carl Erikson and Dinesh Manocha. Hierarchical levels of detail for fast display of large static and

dynamic environments. Technical report, Chapel Hill, NC, USA, 2000.

[4] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics*, 27(Annual Conference Series):247–254, 1993.

[5] Enrico Gobbetti and Eric Bouvier. Time-critical multiresolution scene rendering. In *Proceedings IEEE Visualization*, pages 123–130, Conference held in San Francisco, CA, USA, October 1999. IEEE Computer Society Press.

[6] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 35–42, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[7] J. Edward Swan II, Jesus Arango, and Bala Krishna Nakshatrala. Interactive distributed hardware-accelerated LOD-sprite terrain rendering with stable frame rates. In *Proc. SPIE Vol. 4665, p. 177-188, Visualization and Data Analysis 2002*, pages 177–188, March 2002.

[8] William V. Baxter III, Avneesh Sud, Naga K. Govindaraju, and Dinesh Manocha. Gigawalk: interactive walkthrough of complex environments. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 203–214, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[9] Ashton E. W. Mason and Edwin H. Blake. A graphical representation of the state spaces of hierarchical level-of-detail scene descriptions. *IEEE Transactions on Visualization and Computer Graphics*, 7(1):70–75, 2001.

[10] J. F. Ramos and M. Chover. Lodstrips: Level of detail strips. In *International Conference on Computational Science*, pages 107–114, 2004.

[11] Martin Reddy. Reducing lags in virtual reality systems using motion-sensitive level of detail. In *Proceedings of the second UK VR-SIG Conference*, 1994.

[12] I. Remolar, M. Chover, J. Ribelles, and O. Belmonte. View-dependent multiresolution model for foliage. *Journal of WSCG'03*, 11(2):370–378, 2003.

[13] Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 343–352. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[14] Michael Wimmer and Dieter Schmalstieg. Load balancing for smooth lods. Technical Report TR-186-2-98-31, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, December 1998. human contact: technical-report@cg.tuwien.ac.at.

[15] M. Wloka. Lag in multiprocessor virtual reality. *Presence*, 4(1):50–63, 1995.

[16] Cliff Woolley, David Luebke, Benjamin Watson, and Abhinav Dayal. Interruptible rendering. In *SI3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 143–151, New York, NY, USA, 2003. ACM Press.

[17] Hector Yee, Sumanita Pattanaik, and Donald P. Greenberg. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. In *ACM Transactions on Graphics*, pages 39–65. ACM Press, 2001.

[18] Christopher Zach. Integration of geomorphing into level of detail management for realtime rendering. In *SCCG '02: Proceedings of the 18th spring conference on Computer graphics*, pages 115–122, New York, NY, USA, 2002. ACM Press.

[19] Christopher Zach, Stephan Mantler, and Konrad Karner. Time-critical rendering of discrete and continuous levels of detail. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 1–8, New York, NY, USA, 2002. ACM Press.