Robin Hood's Algorithm for Time-Critical Level of Detail

Eduardo Hernández* and Bedřich Beneš†
Department of Computer Science
ITESM CCM, Mexico City, Mexico

Abstract

We describe a novel method for time-critical rendering of complex scenes populated by numerous distinct objects having an intricate geometry. During the preprocessing step the time consumed by logic and rendering processes is measured. For each frame we estimate the rendering time and distribute it between each potentially visible object which chooses its best representation within the time restrictions. Finally we take advantage of the unused time to improve the rendering process of the remaining objects. This method adjusts image quality adaptively and aims to maintain a constant bounded frame rate even for scenes which complexity may change drastically between frames. In contrast to previous solutions our algorithm can be used in combination with discrete and continuous LOD techniques and it does not impose limitations on the scene conditions. We demonstrate the benefits of our method in a virtual ecosystem composed by trees.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality, Animation; I.3.4 [Computer Graphics]: Picture/Image Generation—Display algorithms.

Keywords: time critical, constant time, level of detail

1 Introduction

One of the most important goals of real-time 3D programming is to provide interactivity with the virtual world. To gain credibility, these virtual worlds are usually made of a large number of complex objects, the scenes simply cannot be displayed in the desired frame-rate because the bus bandwidth is incapable of transferring such huge amounts of data, and at the same time the GPU is unable to process it. Both the interactivity and credibility of real-time 3D programs depend on acceptable frame-rates to preserve the illusion of real-time exploration of the virtual world.

A suggested approach to solve this problem has been the use of level of detail (LOD). We do not have enough space to make an overview of existing techniques, we refer the reader to relevant literature on the topic [Luebke et al. 2002]. These techniques are highly valuable because they allow us to increase the speed of rendering or significantly save the used memory to display the entire environment. However it is important to note that most of these techniques focus on the quality of display and do not solve the problem, in other words they do not offer a constant bounded frame-rate but only shifts the problem to a larger scale.

*e-mail: A00716539@itesm.mx
†e-mail: bedrich.benes@itesm.mx

Another group of techniques is called time-critical level of detail (TCLOD). These techniques adjust the image quality adaptively to maintain a uniform, user-specified target frame rate. Funkhouser and Séquin formulated a solution for the TCLOD [Funkhouser and Séquin 1993] in terms of a problem called "multiple choice knapsack problem" (MCKP) [Armstrong et al. 1983], which belongs to the NP-hard category. From this solution there have been many other proposals trying to find closer results to the optimum for this NP-hard problem but most of them impose some type of restriction or limit their applicability to certain types of scenes.

Instead of trying to adjust the time budget to the available representations of each object in the scene using the MCKP, our method adjusts the representation of each object to its allocated time which is proportional to the visual importance of the object in the resulting image. Therefore our method does not try to solve the NP-hard problem and turns the focus to adjust objects representations which is a field that has been broadly researched in CG with techniques like LOD. The algorithm presented in this paper overcomes many restrictions encountered in the current approaches and improves in the following areas:

- It ensures that each potentially visible object receives time for its rendering. The object is rendered if this time is at least big enough for its simplest acceptable representation.
- It does not impose restriction on the type of scene. It can be used in outdoor or indoor environments, with static or dynamic objects.
- Its predictive ability makes it useful in any environment regardless of its amount of coherence, which refers to the variation in the scene's complexity from frame to frame.
- It is fast and does not require iterative optimizations.
- It can be combined with existing acceleration techniques such as LOD or visibility determination [Teller and Séquin 1991].
- It takes into account the time consumed not only by rendering, but also by logic processes, such as simulations, CPU delays, AI, etc.

Our testing application is a virtual ecosystem. Most of the current solutions show numeric results from scenes composed by only one type of objects. We opt to present results from scenes composed by different types of trees with different degrees of complexity.

2 Previous Work

2.1 Time-Critical Rendering

A first approach was feedback algorithms that adjust LOD selection based on the time required to render previous frames [Schachter 1983]. However these algorithms are inappropriate for discontinuous environments. In contrast, [Funkhouser and Séquin 1993] proposed a predictive algorithm that estimates the time required to render the current frame. Their approach is formulated in terms of the

MCKP and approximates the optimum solution with a greedy algorithm. Unfortunately their result only guarantees to be half as good as the optimum solution and there is the risk that low valued visible objects may not be displayed.

[Maciel and Shirley 1995] used a single hierarchy of impostors that ensures that each visible object is displayed in each frame. This method is only appropriate for outdoor environments with static objects and it can exhaust the texture memory if the hierarchy is big.

[Mason and Blake 1997] proposed a hybrid method of [Funkhouser and Séquin 1993] and [Maciel and Shirley 1995]. LOD is organized in a hierarchal order in which nodes represent impostors of one or more objects in the scene. For each frame a top down greedy algorithm traverses the hierarchy and finds the set of nodes that represents the visible portion of the scene. However, like with [Funkhouser and Séquin 1993] this solution guarantees to be only half as good as the optimum solution and like [Maciel and Shirley 1995] is useless in scenes with moving objects.

[Wimmer and Schmalstieg 1998] used Lagrange multipliers and described a non iterative solution for cases where the maximum triangles count of each object in the scene is at least equal to the maximum triangles that can be rendered with the time budget, other cases may require iterations. However this solution assumes all triangles have the same cost without considering rendering configuration or projected area.

In the case of continuous LOD, discrete variables from the MCKP can be replaced by continuous ones. [Gobbetti and Bouvier 1999] proposed a solution that transforms the original problem into an unconstrained problem which can be solved with iterative optimizations until it produces a result with an acceptable error, otherwise the optimization time is consumed and the last result is used. This method offers several advantages over previous techniques but the optimization can be expensive when the scene has multiple objects and the time budget is tight.

[Zach et al. 2002] combined the discrete LOD hierarchy of [Mason and Blake 1997] with gradient ascent methods that points a search direction for selecting continuous LOD. This method also uses iterative optimizations until the entire time budget is distributed between the visible objects or the optimization time is exhausted.

2.2 LOD for Trees and Plants

[Marshall et al. 1997] used hierarchical tetrahedron impostors for virtual plant ecosystems simplification. The hierarchical scene representation is traversed and, if the cluster is sufficiently far, the corresponding impostor or impostors are displayed.

[Deussen et al. 1998] faced out-of-core rendering of huge plant ecosystems. They used instancing and clustering of plants. Each cluster has a single representative when rendered. This does not increase the speed of rendering but it significantly saves the used memory allowing to display the entire ecosystem.

[Stamminger and Drettakis 2001] described a point-based modeling and rendering technique. Choosing the appropriate point density they are able to produce several view dependent LODs and to reduce the complexity of trees.

[Deussen et al. 2002] extended the point-based rendering to point-based and line-based rendering. Plant foliage is simplified by different LODs, starting with the exact representation and simplifying to more and more coarse point representation. Branches are simplified from the generalized cylinders to poly-lines in the same way.

An automatic LOD generation for an individual plant was described by [Lluch et al. 2003]. Formal plant representation by means of the Lindenmayer's systems is enriched by multi-resolution information that allows extraction of different LODs.

3 Overview of our Approach

The idea behind the algorithm is to adjust the image quality adaptively to maintain a constant desired frame rate. Two benchmarks take place during a preprocessing step: a) the first one measures the time consumed by the rendering of different primitives in various configurations; and b) the second one measures the time consumed by the logic of the program.

The scene is composed by several instances which belong to different types of objects. When the scene is loaded, all types of objects are ordered according to their degree of complexity from simple-to-complex.

For each frame we determine all the potentially visible instances. A first cycle traverses each visible instance which receives a value based on its projected area and importance factor. When the first cycle is done we obtain the total value of the frame which is the sum of all the visible instances values. At this time it is also possible to estimate the time consumed by the logic of the program. Using the estimated time for the logic and the target time for the frame we compute the total time for the rendering.

A second cycle traverses all the potentially visible instances in the preprocessed simple-to-complex order. Each instance receives a rendering time proportional to its value, both the value and the time of the instance are subtracted form the total value of the frame and the total time for the rendering respectively. The instance searches the best representation that can be displayed within the received time. Any remaining time is added to the total time for the rendering thus following instances take advantage of it.

4 Benchmarks

4.1 Rendering

The rendering benchmark takes place during a preprocessing step and measures the $time(P,A,R)$ consumed by the rendering of each primitive, denoted by P, projecting an area A and using a rendering configuration R. To measure each tuple we display n numbers of primitives and the resulting time is calculated as the average time.

For each primitive P in a certain rendering configuration R we take several samples changing the projected area A in constant intervals. In contrast with other type of benchmarks like those presented in [Funkhouser and Séquin 1993] and [Gobbetti and Bouvier 1999] we opt to use this kind of benchmark because several test showed that the change of the projected area affects the rendering time in a non-linear way.

4.2 Logic

The logic benchmark takes place during a preprocessing step and measures the required time by those processes distinct to the rendering. In contrast to the rendering benchmark the measures are done over the same scene used in the interactive application. This benchmark modifies the camera position and orientation to measure

those cases when the frustum encloses the maximum and minimum number of visible instances and it also takes samples changing the number of visible instances in constant intervals. Those samples include the time required for visibility determination, searching of the proper representation of each visible instance and computations related to the time-critical like value assignation, time estimation and time distribution.

This benchmark also includes some constant rendering tasks which are the initial setup, buffer clearing and swapping, and the setup time for each visible instance (rendering configuration, position, orientation and scale). Other types of processes (physics simulation, character movement, AI, etc.) that may affect the time consumed by the frame generation should be also included in this benchmark.

5 Robin Hood's Algorithm for TCLOD

In this section we present the main idea of this paper. Robin Hood's algorithm for TCLOD distributes the rendering time between all the visible instances and its name comes from its objective: *Take the remainder from those instances that received resources in excess and distribute it among the poor ones*. This algorithm has two stages: initialization and display.

5.1 Initialization Stage

It takes place just after the scene is loaded into the memory. It is described by the following tasks:

- Find the complexity of each type of object in the scene.
- Organize the types of objects based on their complexity in a simple-to-complex order.

The $complexity(O)$ heuristic is equal to the $time(O, l_0, A)$ required by the rendering of a type of object O using its best representation l_0 and projecting an area A. The area projected by each type of object can vary since each of its instances has a different position and scale, for this reason A is a constant during this stage and we assume that all types of objects project the same area.

Each type of object has its representations ordered in a set $L = \{l_0, l_1, \ldots, l_{n-1}\}$ where l_0 is the best representation without LOD and l_{n-1} is the simplest acceptable representation. The time required by the representations decreases monotonically as their LOD increases. Each representation has its own rendering configuration. Time is estimated from the number of primitives composing l_0, their rendering configuration, and the hypothetical area A using the data form the rendering benchmark. Once we have computed the complexity of each type of object we organize them in a simple-to-complex order.

5.2 Display Stage

It takes place each time the interactive application generates a new frame and it is described by the following algorithm:

1	Allocate a value to each visible instance.
2	Compute the total value of the frame.
3	Estimate the total time for the logic.
4	Estimate the total time for the rendering.
5	Traverse the visible instances in simple-to-complex order.
6	For each visible instance:
7	Allocate a rendering time proportional to its value.
8	Subtract the value of the instance from the total value of the frame.
9	Subtract the time of the instance from the total time for rendering.
10	Select the best LOD within the allocated time.
11	Add the remaining time if any to the total time for rendering.

The algorithm traverses the visible instances during two cycles. The first cycle (step 1) allocates a value v_i to each of the visible instances by processing the area projected by the instance with a value function described in section 5.4. When the first cycle is finished we can compute the total value of the frame v_f by accumulating the value of the n visible instances using equation 1.

$$v_f = \sum_{i=0}^{n-1} v_i \tag{1}$$

The first cycle is also used to estimate the total time required by the logic t_l using the data from the benchmark of logic. The benchmark takes those processes which are different to the rendering required by each instance and the number of visible instances as parameters and returns the estimated value for t_l. Using t_l and the target time t_f that the frame should spend, we estimate the total time for the rendering t_r, this estimation is described in section 5.3.

The second cycle starts in step 5, in this case instances are traversed based on the complexity of their object type in the simple-to-complex order. The rendering time allocated to each instance t_i can be computed with the equation 2.

$$t_i = \frac{v_i * t_r}{v_f} \tag{2}$$

Finally we subtract the allocated resources from the total values in the way described by equations 3 and 4.

$$t'_r = t_r - t_i \tag{3}$$

$$v'_f = v_f - v_i \tag{4}$$

The values of t'_r and v'_f are always positive and they reach zero after resources have been allocated to the last visible instance. The instance selects one representation which rendering time does not surpass t_i and minimizes the remaining time, this pre-ordered selection is $O(logn)$ but can use the coherence between successive frames to reduce the number of iterations. The rendering time of each representation is estimated considering the actual projected area by the instance, the rendering configuration and number of primitives in the representation. Any remaining time is added to t'_r and therefore can be useful for the following instances.

The second cycle starts with instances from the simplest objects because the algorithm tries to make use of the remaining time of each

instance. The probability that instances from simple objects produce remaining time is greater than for complex objects. Instances from complex objects usually require more time to obtain a good appearance. The purpose of this algorithm is exemplified by two extreme cases:

1) Simple objects project an area bigger than the one projected by complex objects. Instances from simple objects receive a big value and rendering time, since they are simple they will probably choose representation l_0 and will generate a big amount of remaining time which will be used by instances from complex objects.

2) Complex objects project an area bigger than the one projected by simple objects. Instances from simple objects receive a small value and rendering time; probably they will have to use some kind of LOD and will generate a small or null remaining time. Instances from complex objects receive a big value and rendering time, since they are complex they will use most of their rendering time and minimize the remaining time.

5.3 Estimation of Rendering Time

Today's computers have a hardware configuration in which the CPU and the GPU carry out the requested tasks in parallel. Most logic processes are done by the CPU while rendering ones are done by the GPU, during several test we observed the following behavior: a) when the time required by one type of process (t_l or t_r) is much more bigger than the other, the total time of the frame t_f is equal to the biggest time; and b) when both times are equal, t_f is approximately $\frac{2}{3}(t_l + t_r)$ with certain variations depending on the machine.

To simulate this behavior we formulate the following equations. The variable ℓ in equation 5 can be thought as the percent of t_f occupied by t_l. Our method restricts the minimum value of t_f to be t_l, for this reason the maximum value of ℓ is 1. The square minimizes the result when the percent is close to zero, when the percents are close to 1 it has less effect. The variable ρ in equation 6 is the percent of t_f occupied by t_r. Finally t_r is estimated using equation 7 where C is a constant coefficient for a certain machine and ρ is also squared to reduce its value when it is smaller than 1.

$$\ell = \left(\frac{t_l}{t_f}\right)^2 \tag{5}$$

$$\rho = 1 - \ell \tag{6}$$

$$t_r = \frac{\rho^2 t_f}{C} \tag{7}$$

These equations are particularly useful for imposing tight restrictions to the target time t_f. Most available solutions ignore the time consumed by the logic and avoid results with tight time restrictions or tests in single-processor computers.

5.4 Value Function, Remaining Time and Importance

The value function allocates a value to each visible instance. This function is flexible and can be changed depending on the application's objective. Equations 8, 9 and 10 are functions for computing the value v_i where z_i is the depth from the camera to the instance, a_i is the un-projected area of the instance and I is the importance factor for the type of object to which the instance belongs. The impact that these functions have over the scene is exemplified by figure 1 using the implementation of time-critical without LOD from section 7.1. Images in the upper row show a top view of the scene, although the target time t_f is the same for the three images, the resources distribution is different. Note how this distribution also affects the final image in the middle row.

$$v_i = \frac{a_i}{\sqrt{z_i}} I \tag{8}$$

$$v_i = \frac{a_i}{z_i} I \tag{9}$$

$$v_i = \frac{a_i}{z_i^2} I \tag{10}$$

Functions 8, 9 and 10 assign more value to instances closer to the camera but the difference between the value of closer and distant instances is bigger in function 10 than in function 8. In practice, function 8 distributes the resources very well and minimizes the remaining unused time, it can be useful for application where distant objects are visually important (golf, shooting, etc.). On the other hand, function 10 assures high quality to close objects even when the distant objects are visually affected. Function 10 generates big amounts of remaining time, examples of this situation and possible solutions are described in section 7.2.

The importance factor adjusts the value to agree with certain context restrictions or geometry characteristics. For example, our testing application uses an important factor to reduce the value of leaves for two reasons: a) leaves are visually less important than the trunk to recognize the object as a tree; and b) leaves in the front usually cover leaves located behind them, this occlusion reduces the covered screen area.

6 Level of Detail

The technique described in this section is an extension of the work presented by [Deussen et al. 2002]. We use points and lines to simplify the leaves and the trunk respectively. For each element of the tree, leaves and trunk, there is a preprocessing step and a rendering step. We provide a continuous, non-recursive, view dependent LOD and our algorithm improves the previously published methods in the following aspects:

- Different line widths can be used simultaneously.
- Preprocessing steps are done automatically based on the trunk hierarchy. This provides an automatic partition of the tree and generation of bounding volumes.
- Each part of the tree has its own transition resulting into a global smooth transition.

6.1 Trunk

During a preprocessing step the trunk is treated as a hierarchy of generalized cylinders. For each cylinder we identify its triangles and the centers of its two circumferences. We assume the *diameter of the cylinder* to be the largest diameter of its two circumferences. Cylinders are organized based on their diameter length in ascending order. Their diameters, centers of their two circumferences and triangles indices are stored in parallel arrays. The number of triangles that compose each cylinder must be the same for all cylinders in the hierarchy.

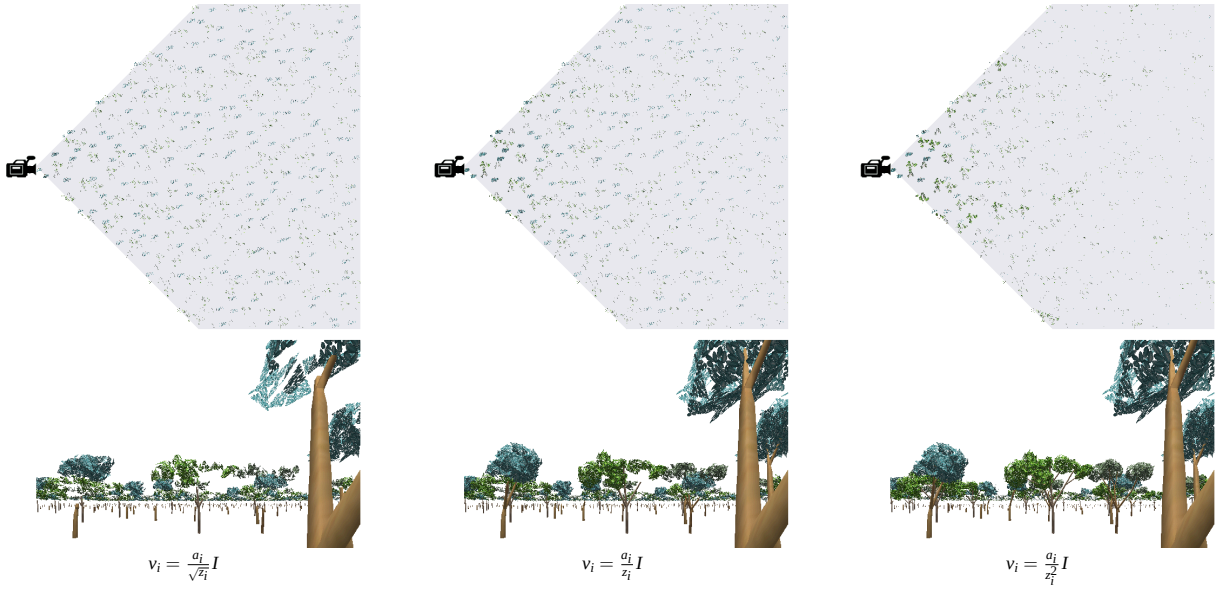

Figure 1: Impact of the different value functions v_i on the scene.

In the rendering step we define a range of available lines widths $[w_{min}, w_{max}]$. The rendering starts traversing lines widths in thin-to-thick order. For each cylinder we compute its projected diameter, if this value is in the range $(\frac{1}{2}w_i, \frac{3}{2}w_i)$ where w_i is the actual line width, it is possible to render the cylinder as a line joining the centers of its two circumferences, in other case we try with the next line width. The rendering using lines finish when: a) all cylinders were rendered; or b) when we found a cylinder which projected diameter is bigger than $\frac{3}{2}w_{max}$, this is also true for the rest of the cylinders because they were organized in thin-to-thick order. In case (b) the remaining cylinders are drawn using triangles, since diameters, centers and triangles indices were stored in parallel arrays it is easy to find the first triangle to render.

6.2 Leaves

In a preprocessing step groups of leaves are formed based on their distances to the terminal cylinders which are the ones that are farthest from the root cylinder. Each group organizes its leaves according to the distance of each leaf to the center of the group in a close-to-far order. We compute the center of each triangle forming the leaves. Triangles indices and centers are stored in arrays parallel to the leaves order inside the groups.

During the rendering step we define the point size. The projected area of each group of leaves is approximated by using the average area of its leaves divided by a constant factor. This factor represents the angle between the normal vector of the leaves and the viewing direction. The number of required points, denoted by p, is the area projected by the group divided by the area projected by the point size. If p is smaller than the number of leaves n we render p points. These points correspond to the centers of the leaves' triangles farthest to the center of the group.

In other case, if $p \leq 2n$ we render $2n - p$ points corresponding to the centers of the leaves' triangles closest to the center of the group, the rest of the leaves are rendered with triangles. This second case enables a smooth transition between points and triangles. Note that in the first case we use the farthest leaves because they preserve the outline of the group while in the second case we use the closest leaves because they are usually covered by the leaves rendered with triangles. Both cases try to reduce the visual impact of the transition. Finally if $p > 2n$ all leaves are rendered with triangles.

7 Results and Discussion

We present two versions of our testing application. The first one implements time critical without using LOD, this simplification clearly illustrates the effect of Robin Hood's algorithm on the scene. The second one implements TCLOD with the Robin Hood's algorithm. We tested a scene composed by 600 instances of trees. There are three types of trees with different degree of complexity as showed in table 1. The trunk and the leaves are considered to be two independent objects. Visibility is determined with a hierarchy of bounding boxes, which encloses the entire scene. Instances inside a box of the hierarchy which is partially visible require some extra tests to cull visible parts of the tree. Figure 2 shows the camera paths through the scene during the tests.

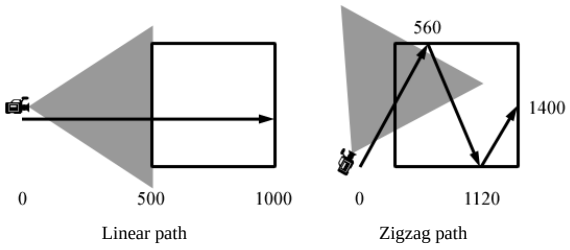

Figure 2: Camera paths through the scene. The square represents the scene dimensions, gray area represents the viewing frustum and frame number has been written at key positions.

7.1 Time-Critical Test without LOD

In this implementation there is no LOD, each instance simply renders the maximum number of triangles allowed by its allocated time

Tree	Trunk △	Leaves △
Type 1	620	480
Type 2	890	720
Type 3	1740	1800

Table 1: Number of triangles composing each part of the trees.

t_i. Figures 3 and 4 show the results of linear and zigzag camera paths respectively using equation 7 and value function 8. The time consumed by each frame without restriction is represented with the solid bold line. Stripped lines crossing the graph represent the target time t_f for each test. Finally solid thin lines show the real time consumed by each frame during the test. Due to its predictive property our method presents an acceptable behavior in all tests even when the scene complexity changes drastically in frames 560 and 1120 from zigzag camera path.

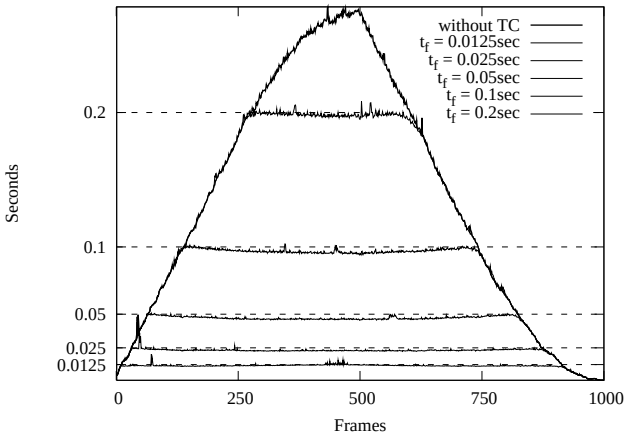

Figure 3: Time consumed by the frame during the linear camera path in the time critical test without LOD.

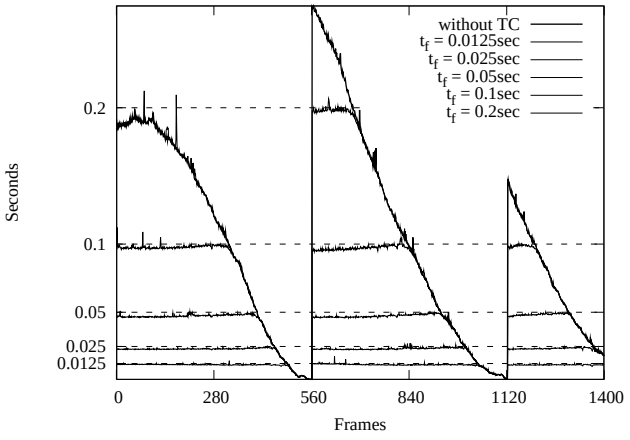

Figure 4: Time consumed by the frame during the zigzag camera path in the time critical test without LOD.

Table 2 shows the numeric results for these tests. Notice that the mean time closely matches the target time t_f and the average error is less than 10% for all tests. Figure 5 shows the appearance of the scene during the tests at the frame 500 of the linear camera path. We also tested different scenes populated by 1000 and 2000 trees,

Path	t_f	Mean	StdDev	AvgErr
	0.0125	0.011791	0.000606	0.065083
	0.025	0.023228	0.000830	0.073824
Linear	0.05	0.047253	0.001121	0.055051
	0.1	0.097081	0.001478	0.029591
	0.2	0.198541	0.001742	0.009370
	0.0125	0.011606	0.000391	0.073498
	0.025	0.023329	0.000545	0.067375
Zigzag	0.05	0.047577	0.001012	0.048548
	0.1	0.097725	0.001561	0.024280
	0.2	0.198201	0.001725	0.010892

Table 2: Numeric results of the time critical tests without LOD. An average error equal to 1 means that the real time is 100% different from the target time.

results were almost identical and in some cases better. Tests on different machines produced similar results.

7.2 Time-Critical Test with LOD

Ranges of available line widths and point sizes for the LOD were defined to be $[1,20]$. Each instance (trunk or leaves) verifies if its allocated time t_i is big enough to support the representation l_0, if the condition fails the instance replaces triangles by lines or points starting with the smaller widths or sizes until it finds a representation that fulfills the time restriction. While increasing the line width or point size produces a faster representation it also reduces the visual quality. If no line width or point size produces a valid representation, the fastest representation found is rendered in time-critical mode, i.e., with some details missing.

Figure 6 shows the number of primitives rendered to meet a target time constrain of $t_f = 0.04seg$ during the linear camera path. The time consumed by the frame without restriction and using TCLOD is presented in figure 7. Notice that the use of points and lines is incremented when the complexity of scene increases around the frame 500. Figure 8 compares the appearance of the scene at frame 500 with TCLOD (top image) and without restriction (bottom image). Although the use of LOD is evident, top image shows complete trunks and leafy leaves projecting an area similar to the one in the bottom image. For this test we used the value function 11 which substitutes depth by distance to increment the value of objects appearing near the middle of the screen (focus). This function produces a visual result similar to the one of function 10 but the square root reduces the remaining time.

$$v_i = \sqrt{\frac{a_i}{d_i^2}} I \tag{11}$$

This test produces a mean time of 0.036754 sec, standard deviation of 0.002260 and average error of 0.088692. Event though results are pretty acceptable they are not as accurate as the ones presented in section 7.1, some reasons are: a) the simultaneous use of different primitives; b) the approximation of the trunk projected length; and c) the assumption that the time required to search a proper representation is constant. These reasons contribute to hinder the time estimation but the main reason of the error increment is that function 11 allocates big amount of time to instances closer to the camera which sometimes produces a big remaining of time, if these instances belong to a complex type of object Robin Hood's algorithm is unable to completely consume the remaining time with the following instances if any. These cases can be easily detected by comparing the time required by l_0 with t_i, since t_i is proportional

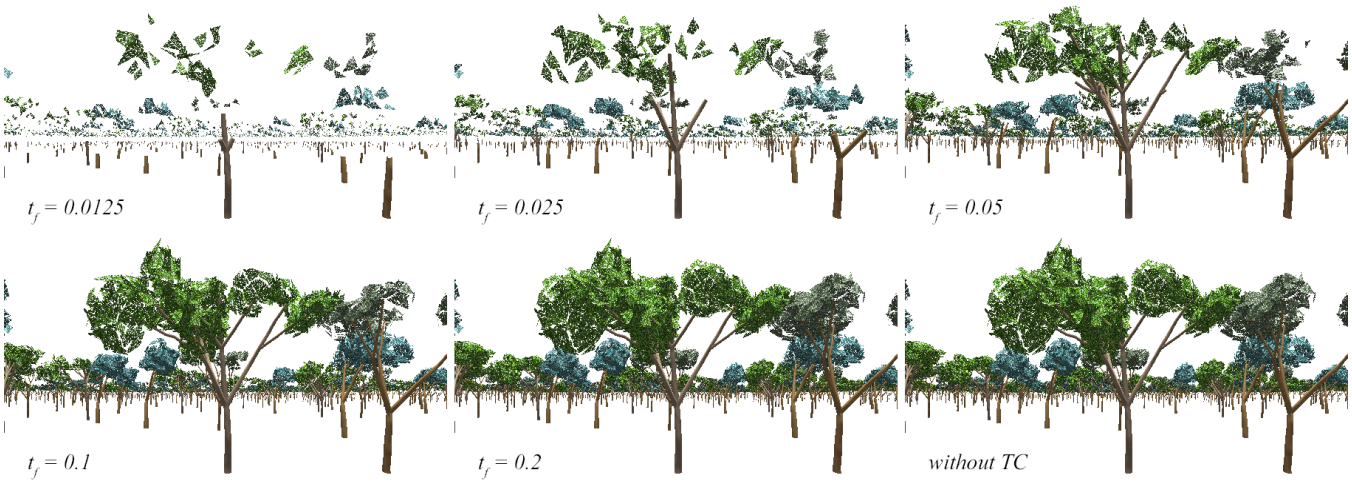

Figure 5: Snapshots of the frame 500 at different target times. The linear camera path and the time critical test without LOD were used.

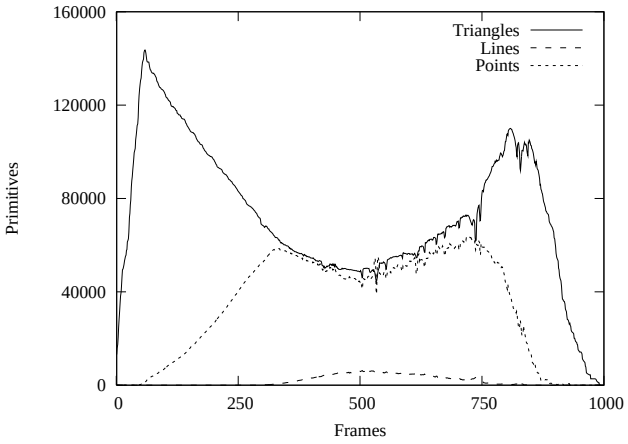

Figure 6: Rendered primitives during the linear camera path using TCLOD implementation with a target time $t_f = 0.04sec$.

to the visual importance of the instance a proper solution is to reduce the remaining time up to a value by improving the quality of the instance with techniques such as tessellation [Turk 1992], anti-aliasing, shadows, etc. This solution not only provides a more stable frame-rate but also improves the image quality. In conclusion the algorithm can offer better results when combined with techniques that diminish objects quality and techniques that improve it.

8 Conclusion and Future Work

We have described a method for time-critical rendering of complex scenes populated by numerous distinct objects having a complex geometry. We measured the time consumed by logic and rendering processes and for each frame we estimated the rendering time and distribute it between each potentially visible object choosing its best representation within the time restrictions. Finally we took advantage of the unused time to improve the rendering process of the remaining objects. Our test proved the ability of the method

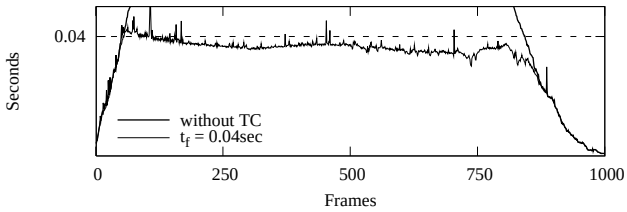

Figure 7: Time consumed by the frame during the linear camera path using TCLOD implementation with a target time $t_f = 0.04sec$.

to adjusts image quality adaptively and maintain a fairly constant bounded frame rate. In contrast to the current solutions our algorithm is not restricted to one type of LOD technique or to specific scene conditions. We also described an extension of a current LOD technique for trees.

Future research will include better techniques to estimate times and projected areas. Also mechanisms to improve and diminish the visual quality of objects in complex environments including physics simulations, character movement, AI, etc.

Acknowledgments

I would like to thank to my sister Andrea Hernández for proof reading the paper.

References

ARMSTRONG, R. D., KUNG, D. S., SINHA, P., AND ZOLTNERS, A. A. 1983. A computational study of a multiple-choice knapsack algorithm. *ACM Trans. Math. Softw. 9*, 2, 184–198.

DEUSSEN, O., HANRAHAN, P., LINTERMANN, B., MĚCH, R., PHARR, M., AND PRUSINKIEWICZ, P. 1998. Realistic modeling and rendering of plant ecosystems. In *Proceedings of the*

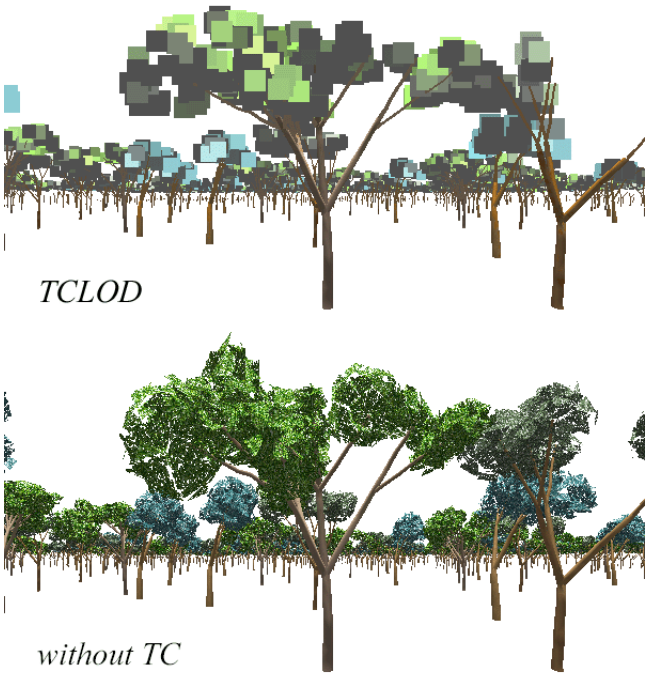

Figure 8: Snapshots of the scene using TCLOD and without TC.

25th annual conference on Computer graphics and interactive techniques, ACM Press, 275–286.

DEUSSEN, O., COLDITZ, C., STAMMINGER, M., AND DRETTAKIS, G. 2002. Interactive visualization of complex plant ecosystems. In *Proceedings of the conference on Visualization '02*, IEEE Computer Society, 219–226.

FUNKHOUSER, T. A., AND SÉQUIN, C. H. 1993. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM Press, 247–254.

GOBBETTI, E., AND BOUVIER, E. 1999. Time-critical multiresolution scene rendering. In *Proceedings of the conference on Visualization '99*, IEEE Computer Society Press, 123–130.

LLUCH, J., CAMAHORT, E., AND VIVÓ, R. 2003. Procedural multiresolution for plant and tree rendering. In *Proceedings of the 2nd international conference on Computer graphics, virtual Reality, visualisation and interaction in Africa*, ACM Press, 31–38.

LUEBKE, D., WATSON, B., COHEN, J. D., REDDY, M., AND VARSHNEY, A. 2002. *Level of Detail for 3D Graphics*. Elsevier Science Inc.

MACIEL, P. W. C., AND SHIRLEY, P. 1995. Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, ACM Press, 95–ff.

MARSHALL, D., FUSSELL, D., AND CAMPBELL III, A. T. 1997. Multiresolution rendering of complex botanical scenes. In *Graphics Interface '97*, Canadian Human-Computer Communications Society, W. A. Davis, M. Mantei, and R. V. Klassen, Eds., 97–104.

MASON, A. E. W., AND BLAKE, E. H. 1997. Automatic hierarchical level of detail optimization in computer animation. *Computer Graphics Forum 16*, 3 (September), 191–199.

SCHACHTER, B. J. 1983. *Computer Image Generation*. Krieger Publishing Co., Inc.

STAMMINGER, M., AND DRETTAKIS, G. 2001. Interactive sampling and rendering for complex and procedural geometry. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, Springer-Verlag, 151–162.

TELLER, S. J., AND SÉQUIN, C. H. 1991. Visibility preprocessing for interactive walkthroughs. *Computer Graphics 25*, 4, 61–68.

TURK, G. 1992. Re-tiling polygonal surfaces. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, ACM Press, 55–64.

WIMMER, M., AND SCHMALSTIEG, D., 1998. Load balancing for smooth lods. Technical report, Vienna University of Technology.

ZACH, C., MANTLER, S., AND KARNER, K. 2002. Time-critical rendering of discrete and continuous levels of detail. In *Proceedings of the ACM symposium on Virtual reality software and technology*, ACM Press, 1–8.

About the authors

Eduardo Hernández has just graduated from the Masters in Computer Science. He has worked as a game developer in Japan. His interests include: virtual and mixed reality, real-time rendering and game design. He is currently looking for a position in the game industry. You can contact him at A00716539@itesm.mx

Bedřich Beneš has his Ph.D. in Computer Graphics from the Czech Technical University. He works in the area of artificial life, procedural modeling, and real-time rendering. He is author of three books about Computer Graphics and more than twenty papers. You can contact him at bedrich.benes@itesm.mx