Time-critical Rendering of Discrete and Continuous Levels of Detail

Christopher Zach
VRVis Research Center
Inffeldgasse 16
A–8010 Graz
zach@vrvis.at

Stephan Mantler
VRVis Research Center
Donau–City–Straße 1
A–1220 Wien
step@vrvis.at

Konrad Karner
VRVis Research Center
Inffeldgasse 16
A–8010 Graz
karner@vrvis.at

ABSTRACT

We present a novel level of detail selection method for real-time rendering, that works on hierarchies of discrete and continuous representations. We integrate point rendered objects with polygonal geometry and demonstrate our approach in a terrain flyover application, where the digital elevation model is augmented with forests. The vegetation is rendered as continuous sequence of splats, which are organized in a hierarchy. Further we discuss enhancements to our basic method to improve its scalability.

Categories and Subject Descriptors

I.3.7 [**Three-Dimensional Graphics and Realism**]: Virtual reality

Keywords

Real-time rendering, level of detail management, point rendering, rendering of vegetation

1. INTRODUCTION

Time-critical rendering ensures guaranteed frame rates even for scenes with very high complexity. Therefore it provides a convenient framework for real-time rendering applications. Typically a time-critical framework mainly consists of a level of detail (LOD) selection method, that chooses the most valuable representation for visible objects not exceeding the available rendering budget.

We present a time-critical rendering approach that combines discrete and continuous LOD selection and we demonstrate its benefits in a terrain flyover application. Our LOD selection method is based on an iterative optimization method over mixed (discrete and continuous) variables, that essentially combines previously used approaches for discrete and smooth LOD selection. Further we describe an extension to this basic LOD selection method to handle a large number of objects with smooth representations.

Our testing application consists of a terrain rendering system augmented with the ability to display dense vegetation. We employ point based representations of procedurally generated trees for this task.

In Section 2 we give an overview of previous work related to time-critical rendering and visualization of vegetation. Our own LOD selection method is described in Section 3. Our testbed is briefly sketched in Section 4 and details on the implementation are given in Section 5. A section on obtained results and on conclusions and future work close this paper.

2. PREVIOUS WORK

2.1 Time-critical Rendering

Most work on rendering acceleration techniques focuses on improving the frame rate while maintaining a controlled image quality. We aim on real-time rendering that guarantees a user given frame rate regardless of the complexity of the scene. The rendering process of every frame has to meet strict timing constraints and the goal of the rendering framework is to attain the best visual quality with the available rendering time. In this section we summarize related work on time-critical rendering.

2.1.1 Discrete Level of Detail Management

Most scene graph libraries use distance based or screen size based LOD switching to accelerate rendering of complex scenes. The rendering time for each frame depends on the scene content, the viewing parameters and the LOD switching threshold, but it can be arbitrarily large. In terms of optimization these methods maximize the frame rate subject to controlled image quality. Whenever a guaranteed interactive frame rate is needed, the role of frame rate and image quality must be exchanged: maximize the image quality that is achievable subject to controlled render time.

Funkhouser and Séquin [3] formulated this optimization task as a multiple choice knapsack problem (MCKP), which is known to be NP-hard, and used an approximation method to select the appropriate LOD for each object to be rendered. The importance of every object and the accuracy of every LOD depend on the viewing parameters, thus the MCKP must be solved for every frame.

Solving the MCKP needs a certain amount of time, which may affect the rendering performance on a uniprocessor computer. Funkhouser and Séquin used a dual processor solution for rendering: one processor selects the representations for the next frame and the other processor feeds the graphics pipeline.

This approach has one major disadvantage: visible objects may be completely missing in the rendered image, if the allowed rendering time is not large enough to draw at least the coarsest representation of every visible object. Both Mason and Blake [13] and Maciel and Shirley [11] utilized a hierarchical level of detail approach and used variations of the MCKP to select appropriate representations for every frame. Mason and Blakes approximation method of their extended MCKP can be seen as a top down greedy traversal of the LOD hierarchy and guarantees half optimality, whereas Maciel and Shirleys heuristics don't provide such lower bounds. An extensive discussion of LOD management approaches can be found in [12].

Since in our implementation the selection of discrete levels of detail is based on Mason and Blakes work, we describe their idea briefly. Basically their LOD selection performs a sequence of so called increment steps as long as the rendering budget constraint is not violated. An increment step replaces a currently selected representation by the most valuable successor nodes in the LOD hierarchy. A converse decrement step is also introduced, which replaces least rated successor nodes with their parent. The root node (representing the entire scene at the coarsest level) is the initially selected representation. Additionally Mason and Blake exploited frame–to–frame coherence using the LOD assignment for the previous frame as starting point for a sequence of increment and decrement steps. Since our approach does currently not exploit frame–to–frame coherence and therefore starts from scratch at every frame, our method performs only increment steps.

2.1.2 Multiresolution Level of Detail Management

If the object representation allows smooth levels of detail, a similar optimization problem can be formulated, where the discrete variables are replaced by continuous ones. Gobetti and Bouvier [4], [5] applied optimization techniques for solving non-linear constrained systems to time-critical rendering of multiresolution meshes. They partitioned the available frame time into the time available to the LOD selection procedure and the actual rendering time. Their LOD selection method proceeds incrementally by successively improving the current solution. Thus, this method can be interrupted at any time returning a suboptimal, but feasible solution.

Their LOD selection procedure is not applicable to scenes with a large number of multiresolution objects, since they assign one variable to every object.

Wimmer and Schmalstieg [20] describe a direct (non iterative) solution for smooth LOD selection using Lagrange multipliers, but they take only the rendering budget constraint into account and ignore potential constraints on the maximal resolution of every object. Therefore this approach works well only for objects with virtually unbounded resolution.

Schmalstieg and Fuhrmann [17] implemented a LOD selection policy for hierarchical and deformable multiresolution models and applied their method to character animation and terrain rendering.

2.2 Point based Rendering

Point based rendering was first introduced by Levoy and Whitted [9]. They observed that with the growing complexity of computer generated scenes, classical modeling primitives such as triangles become less appealing. Using points as a rendering primitive bears several advantages, such as being able to render arbitrarily complex geometry with a standardized rendering algorithm.

Point based rendering is also applicable to the direct rendering of volumetric data, first introduced by Westover [19] and improved by numerous authors.

Pfister et al. [14] and Rusinkiewicz et al. [16] almost simultaneously published new point based systems that used additional information, such as surface normals and texture data, for each point sample. However, the method introduced by Pfister was aimed at high fidelity, whereas Rusinkiewicz was more interested in handling huge amounts of data. The goal was to interactively visualize laser scans of up to 2 billion samples at interactive frame rates. Therefore, they designed a hierarchical data structure that allowed lower resolutions to be displayed even while additional data was still being read in.

Recently Cohen et al. [2] and Chen and Nguyen [1] combined polygonal and point based rendering by exploiting a hybrid representation of the entire scene. This is different to our approach, since we represent one object either point based or polygonal.

2.3 Terrain Visualization

Recent work on large scale terrain visualization with regular heightfields was done for instance by Hoppe [7] and Lindstrom and Pascucci [10]. Both approaches employ a smooth level of detail framework and the required resolution is calculated from the allowed screen space error. This threshold can be adaptively refined during the animation to match the desired frame rate.

If the scene database consists of several LOD hierarchies (e.g. terrain augmented with vegetation), the adjustment of resolutions of different parts is no longer unique. Therefore we utilize a predictive model of the visual quality and rendering cost to calculate the appropriate resolution for each scene component.

Our terrain visualization is based on the Styrian flyover project (Kofler et. al. [8]), which uses a quad-tree representation of the digital elevation model and the ground texture. During rendering the determination of the required resolution is based on the distance to the virtual camera.

The quadtree approach can be seen as a coarse variant of view–dependent multiresolution meshes (e.g. view–dependent progressive meshes, VDPM [6]). Even with smoother view–dependent progressive meshes our method described in Section 3 remains the same, since the current level of detail of a VDPM consists of a (discrete) set of nodes in the VDPM representation.

3. MIXED LEVEL OF DETAIL SELECTION

In this section we present a LOD selection method for scenes, that consist of objects represented by discrete and smooth levels of detail.

3.1 Overview

Essentially our algorithm performs in every iteration either a greedy refinement step of discrete levels of detail or a gradient ascent step for smooth representations, whatever is favourable in terms of (estimated) visual benefit to rendering cost ratio. The iteration stops when the available rendering budget is exhausted, the highest level of detail of visible objects is attained or the time available to the optimization procedure itself is completely spent. In any case the obtained assignment of resolution for every object is feasible.

Since in our application there is a huge number of smoothly represented objects and therefore a huge set of continuous resolutions to determine, we utilize a hierarchical arrangement to accelerate the LOD selection procedure.

3.2 Problem Formulation

We consider the LOD selection problem for discrete and continuous variables. A set of representations $\{r_i\}$ of discrete objects and a set of objects $\{o_i\}$ with smooth levels of detail are given. We will identify r_j with a boolean variable of the same name; we set $r_j = 1$, if the LOD selection chooses r_j for rendering in the next frame and $r_j = 0$ otherwise. Additionaly we associate a continuous variable x_i with o_i, that denotes the chosen resolution for rendering o_i.

For every representation a pair of functions is given: the first function (the *benefit*) estimates the effect on the visual quality, if this representation is rendered; the other one estimates the corresponding cost of drawing (i.e. the rendering time). For discrete objects we denote these two functions by $benefit_d(r_i)$ and $cost_d(r_i)$ respectively. For objects with continuous resolutions these functions have the resolution x_i as the argument and are written as $benefit_c(x_i)$ and $cost_c(x_i)$. Obviously these functions depend on the current viewing parameters, but for convenience we will not expose this dependency explicitly.

The task of the LOD selection method is to choose r_j and x_i:

$$\begin{aligned} \max \sum_j benefit_d(r_j) &+ \sum_i benefit_c(x_i) \text{ s.t.} \\ \sum_j cost_d(r_j) &+ \sum_i cost_c(x_i) \leq T \\ r_j &\in \{0,1\} \\ x_i &\in [0,1] \end{aligned}$$

We restrict the resolution x_i to the range $[0,1]$, where $x_i = 1$ represents the maximal resolution assigned to a multiresolution object. Further there are usually more constraints related to r_j, that encode the LOD hierarchy (details can be found in Mason and Blake [13]).

We aggregate the x_i into one vector $\vec{x}$ and denote the total cost caused by rendering smooth representations by

$$cost_c(\vec{x}) = \sum_i cost_c(x_i).$$

Similarly we define the total benefit of rendering the smooth levels of detail at resolution $\vec{x}$ as

$$benefit_c(\vec{x}) = \sum_i benefit_c(x_i).$$

3.3 Terminology

We require some additional notation to express the algorithm and its derivation in a compact manner. For any current assignment of smooth variables $\vec{x}$ and an arbitrary search direction $\vec{a}$ (with $\|\vec{a}\| = 1$) we define the directional cost function

$$\widehat{cost}_c(t) := cost_c(\vec{x} + t\,\vec{a})$$

and (by the chain rule) we have

$$\frac{d\widehat{cost}_c}{dt} = (\vec{a},\, \nabla_{\vec{x}} cost_c(\vec{x} + t\,\vec{a}))$$

(We denote the inner product of $\vec{a}$ and $\vec{b}$ by $(\vec{a}, \vec{b})$.) Similarly, we introduce the directional benefit function

$$\hat{b}_c(t) = benefit_c(\vec{x} + t\,\vec{a}).$$

Again, we will need the derivative in the following sections:

$$\frac{d\hat{b}_c}{dt} = (\vec{a},\, \nabla_{\vec{x}} benefit_c(\vec{x} + t\,\vec{a})).$$

These functions depend on $\vec{x}$ and $\vec{a}$ as well, but for clarity we omit these arguments, since they are clear from the context.

3.4 Optimization Method

Essentially our method is an interleaved combination of Masons and Blakes [13] increment steps for discrete LODs and gradient ascent steps for smooth representations. The optimization method works iteratively: Given a current assignment of discrete and continuous resolutions for each object, one of three cases may occur:

1. The rendering budget or the time available to LOD selection is exhausted and the procedure terminates with the current solution.
2. The discrete variables and therefore the total benefit and costs are increased.
3. The smooth resolutions are raised, thus increasing both the total benefit and cost.

In every iteration, the algorithm decides either to increase the discrete or continuous variables. If the discrete variables are already at the highest resolution or increasing every possible discrete representation causes violation of the constraint, the algorithm tries to enhance the smooth representations.

On the other hand, if the smooth levels are already at the highest resolution, only improving the discrete resolutions is considered by the algorithm.

If both improvement steps are feasible, then the algorithm compares the added benefit to added cost ratio of the best discrete improvement with the slope of the benefit function along the search direction. More formally we denote the added benefit obtained by the increment step as $\Delta benefit_d$ and the corresponding cost as $\Delta cost_d$. The quantities to compare are

$$slope_d = \frac{\Delta benefit_d}{\Delta cost_d}$$

and

$$slope_c = \frac{d\tilde{b}_c}{dcost}(cost_c(\vec{x}_i)),$$

where

$$\tilde{b}_c(cost) = \hat{b}_c(\widehat{cost}_c^{-1}(cost)).$$

Applying the chain rule we obtain

$$\frac{d\tilde{b}_c}{dcost}(cost_c(\vec{x}_i)) = \frac{d\hat{b}_c}{dt}(0)\,\frac{1}{(\vec{a},\, \nabla_{\vec{x}} cost_c(\vec{x}_i))}.$$

(Recall that $\hat{b}_c$ implicitely depends on the current assignment $\vec{x}$ and the current search direction $\vec{a}$.)

If $slope_d > slope_c$, then incrementing the discrete LODs is certainly favourable. Otherwise the smooth variables are increased and the new solution $\vec{x}_{new}$ is set according to

$$\vec{x}_{new} = \vec{x} + t_{opt}\,\vec{a}.$$

3.4.1 *Selection of the Search Direction*

Essentially the search direction $\vec{a}$ is the gradient of $b_c(\vec{x})$. If x_i is already saturated ($x_i = 1$), then $a_i = 0$. To avoid too small steps because of repeated saturation of smooth variables, approaching the boundary $x_i \leq 1$ can be penalized by a barrier term, e.g.:

$$a_i = \max\left\{0, \frac{\partial b}{\partial x_i}(\vec{x}) - \frac{\varepsilon}{1 - x_i}\right\},$$

where ε is a small constant.

3.4.2 *Line Search*

Given the search direction $\vec{a}$, $\vec{x} + t\,\vec{a}$ leaves the feasible region for some $t_{max} > 0$. If $d\tilde{b}_c/dcost > slope_d$ at t_{max}, then $t_{opt} = t_{max}$, since raising the continuous resolutions along the search direction $\vec{a}$ gives always better values.

Otherwise we determine $t_{opt} \in (0, t_{max})$ such that

$$\frac{d\tilde{b}_c}{dcost}(\widehat{cost}_c(t_{opt})) = slope_d.$$

Thus, we increase the values of the smooth variables as long as this is preferable over the discrete incrementation step. The graphical explanation is given in Figure 1.

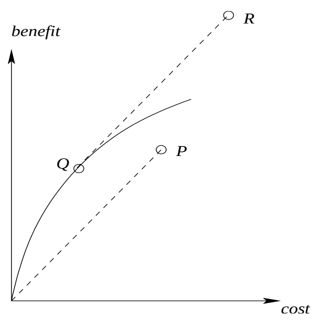


Figure 1: Determination of t_{opt}. The dashed line corresponds to a discrete increment step, whereas the solid curve illustrates $\tilde{b}_c$. P represents the location in the benefit/cost space after the discrete increment step, Q corresponds to the optimal increase of the continuous resolutions and R depicts the added result.

3.5 Continuous Variable Splitting

The optimization procedure described in Section 3.4 turned out to be not sufficiently scalable, because the number of continuous variables is unnecessarily large for certain viewpoints. Although the time used for LOD selection can be amortized over several frames, there is an even better option available. We observed, that (almost) identical resolutions are often assigned to spatially close smooth objects. Therefore we augmented view frustum culling of smooth objects with hierarchical splitting of variables.

The idea is the following: Consider the optimization problem described in Section 3.2 augmented with additional constraints:

$$x_1 = x_2 = \ldots = x_n \tag{1}$$

These constraints ensure that the continuous variables always have the same value. The optimal value of this tightened problem yields a non-optimal assignment of smooth LODs, but this loss in optimality can be predicted. If the estimated loss is smaller than some threshold θ, we replace the n smooth variables by only one, namely x, and set $x_i = x/n$. Additionally we replace the upper bounds on x_i with $x \leq n$. Note, that the benefit and cost is still expressed wrt. x_i, e.g.

$$\sum benefit_c(x_i) = n\, benefit_c(x/n).$$

Otherwise we relax constraint 1 and replace it by one of the following four constraints:

$$\begin{aligned} x_1 &= \ldots = x_{n/4} \\ x_{n/4+1} &= \ldots = x_{n/2} \\ x_{n/2+1} &= \ldots = x_{3n/4} \\ x_{3n/4+1} &= \ldots = x_n \end{aligned}$$

These groups of variables correspond to child nodes in a quad-tree structure, in which the smooth objects are organized. For each of these constraints we estimate the loss in the corresponding optimization problem and compare it with an appropriate threshold $\theta/4$. If the loss is too large, the corresponding group of variables is recursively divided (i.e. the quad-tree node is replaced by its children) until the loss is small enough or the original variables are reached.

The detailed description of this splitting method is postponed to Appendix A.

4. TEST APPLICATION

Our test application consists of a terrain flyover scenario showing a large part of the Styrian province. The data set and our approach to terrain visualization is taken from the Styrian flyover project (Kofler et al. [8]). The complete height field consists of 2049×2049 samples covering a region of about $120 \times 120 km^2$. The corresponding texture has a resolution of 4096×4096. Both the height field and the associated texture fit entirely into main memory.

About 1.4 million trees are placed on a jittered grid into the terrain according to the forest regions found by image classification of the satellite image. The heights of these trees are exaggerated to have a closer match between the resolution of the ground texture and the vegetation. The type of tree was selected randomly, but spruces are more likely in higher terrains. In valleys and other lower parts the forest is a mixture of various conifers and broad-leafed trees.

The digital elevation model and the ground texture are organized in a quad-tree data structure as it is commonly used in terrain visualization systems (e.g. the Styrian Flyover project [8], TerraVision II [15]). This quad-tree forms the discrete LOD hierarchy.

The trees are grouped into sets of at most 64 spatially close, individual trees and the appropriate resolution is determined for these groups. A quad-tree hierarchy is employed again to speed up view frustum culling and to allow variable splitting as described in Section 3.5.

5. IMPLEMENTATION

In this section we provide some details on our implementation.

5.1 Tree Generation

In the current system, the tree generation itself is based on the algorithm by Weber and Penn [18]. Tree models are generated offline and stored on disk in an intermediate format, storing geometric information for each stem or branch as well as leaf locations. Upon startup of the test application, the desired trees are read in and the branch information is added as a list. The leaf position data is converted to a K-d-tree by recursively splitting the volume at the median of the longest axis. Each model is only read in once, and instances can be made by setting up the OpenGL transformations as required.

5.2 Point based Tree Rendering

The point based renderer is called seperately for each tree, giving the rendering system fine control over the fidelity of each tree instance. Visual quality is controlled through a budget value that is passed to the rendering function. This value basically represents the number of point samples that may be rendered for this particular instance. In addition, the current view direction is also available to the renderer. Since many leaves on the back side of the tree are possibly obscured, the view direction can be used to advise the renderer where to spend more of the total budget.

The rendering itself is a recursive traversal of the K-d-tree; at each level, the budget is divided between each of the branches. If the budget reaches zero, the current level is rendered as one point splat.

5.3 Benefit and Cost Heuristics

Predicting the cost of rendering a representation is based on a model of the graphics pipeline. The cost of rendering a splatted representation is directly proportional to the resolution, i.e.

$$cost_c(x_i) = c_{splat}\, N_{splats}\, x_i + c_{tree},$$

where c_{splat} denotes the rendering time for one splat and c_{tree} denotes the cost for individual tree setup, e.g. for transformation of the tree prototype to the final tree position. Both values are determined by a benchmarking procedure. On our hardware c_{splat} is $0.35\mu s$ and c_{tree} is $2.45\mu s$.

Since our polygonal objects (patches of a regular digital elevation model) consist of long triangle strips, we estimate the rendering cost as

$$cost_d(n) = c_{vertex}\, n,$$

where n is the number of sent vertices in the triangle strip and c_{vertex} is again an empirically measured combined cost for vertex and triangle processing (determined as $0.06\mu s$ on our hardware).

The benefit functions are chosen as follows: For every terrain patch at level d the benefit is approximated by

$$\sqrt{2^{d_{max}-d}}\, S,$$

where S is the estimated screen size in pixels and d_{max} is the height of the terrain quad–tree.

The benefit associated with a group of N_{trees} trees is chosen as

$$k\, S^2\, \frac{N_{trees}}{64}\, \sqrt{\frac{x_i}{N_{trees}}\, N_{splats}},$$

where S is again the estimated screen size and x_i is the assigned resolution. k is a weighting constant currently set to $1/16$. Recall that $\frac{N_{trees}}{64}$ denotes the relative density of trees in one group containing at most 64 trees. We assign a higher benefit to trees with larger extents on the screen to avoid sparse splatting of vegetation in the foreground.

5.4 Small Detail Culling

In order to avoid expensive transformations to position the tree templates we perform small detail culling for groups of trees, which have a very small estimated screen size. This culling process is performed in combination with view frustum culling. The remaining nodes (collections of trees according to the quad-tree hierarchy) are passed to the LOD selection procedure.

If the point budget assigned to a group of trees is not sufficient to display each tree with at least one splat, only the corresponding fraction of the group is rendered. Thus, we obtain a smoother transition between culled vegetation and still displayed groups.

6. RESULTS

We performed several measurements on a notebook computer with 1GHz processor and GeForce2 Go graphics card. Figure 4 shows frames from our demonstration application at various frame rates. It can be clearly observed, that trees closer to the virtual viewpoint have higher resolution than distant vegetation. Figure 2 compares the predicted rendering time with the actual rendering time. Since trees with an assigned resolution of less than one splat are not displayed at all, the available rendering budget is usually not completely exploited.

Further we measured the time spent in the optimization procedure. LOD selection times without the hierarchical variable splitting approach for a fixed path are shown in Figure 3 as solid line, whereas the time spent for LOD selection with the variable splitting approach is drawn as dashed line. These timings include culling and the actual LOD selection. The time used for view frustum culling is usually less than 2ms. Note that the vertical axis has logarithmic scale and the basic method took longer than 1 second in the worst case. Obviously the later, enhanced method is far more favourable.

7. CONCLUSIONS

We presented a time-critical LOD management method that incorporates discrete and continuous levels of detail. Our approach combines approximation methods for discrete LOD selection with gradient ascent methods to choose an appropriate multiresolution representation. A terrain flyover application comprises the current testbed for our LOD management procedure.

In this work our main focus is the LOD selection method and we are aware that both terrain rendering and drawing of vegetation can be substantially enhanced. Especially our prototype of the point based vegetation renderer can be highly accelerated. Nevertheless our work demonstrates the prospects of mixed resolution objects within a time-critical framework.

We intent to utilize this approach in an urban visualization application, which aims on the real-time photorealistic presentation of an existing city. Further we will refine the terrain rendering engine and work on improvements on the drawing of vegetation. Additionally occlusion culling in terrain environments would significantly increase the accuracy of our LOD selection.

8. ACKNOWLEDGEMENTS

We would like to thank Martin Urschler for the terrain rendering implementation and Matthias Rüther for his forest segmentation. Further we would like to thank Horst Bischof for valuable suggestions.

This work has been done in the VRVis research center, Graz and Vienna/Austria (http://www.vrvis.at), which is partly funded by the Austrian government research program Kplus.

9. REFERENCES

[1] B. Chen and M. X. Nguyen. POP: a hybrid point and polygon rendering system for large data. In *IEEE Visualization 2001*, pages 45–52, 2001.

[2] J. D. Cohen, D. G. Aliaga, and W. Zhang. Hybrid simplification: Combining multi–resolution polygon and point rendering. In *IEEE Visualization 2001*, pages 37–44, 2001.

[3] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of SIGGRAPH '93*, pages 247–254, 1993.

[4] E. Gobbetti and E. Bouvier. Time-critical multiresolution scene rendering. In *IEEE Visualization '99*, pages 123–130, 1999.

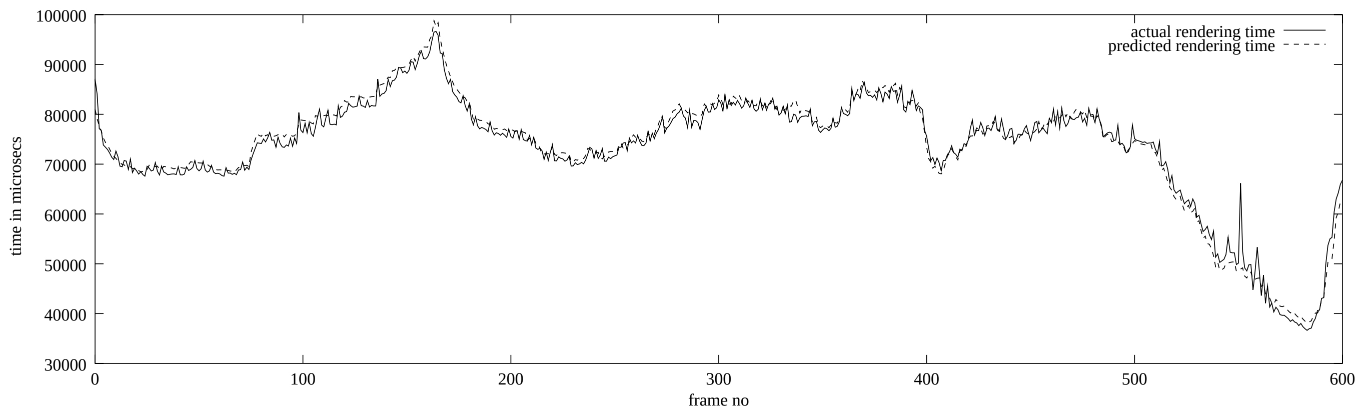
[5] E. Gobbetti and E. Bouvier. Time-critical multiresolution rendering of large complex models. *Journal of Computer-Aided Design*, 32(13):785–803, 2000.

[6] H. Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH '97*, pages 189–198, 1997.

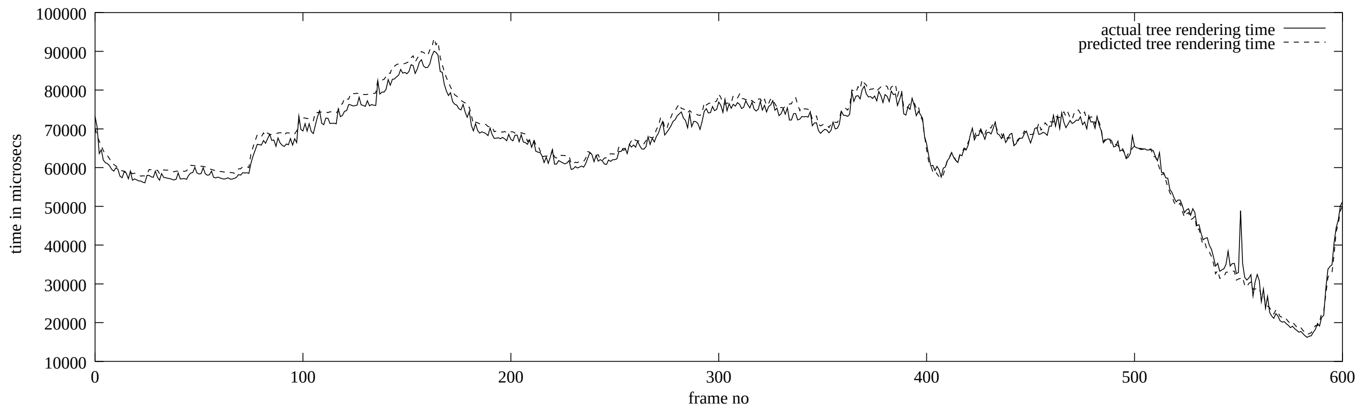
[7] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Visualization '98*, pages 35–42, 1998.

[8] M. Kofler, M. Gervautz, and M. Gruber. The Styrian Flyover – LOD management for huge textured DTM models. In *Proceedings of Computer Graphics International*, pages 444–454, 1998.

[9] M. Levoy and T. Whitted. The use of points as a display primitive. Technical report, Univ. of North Carolina at Chapel Hill, Dept. of Computer Science, 1985.



(a) Total rendering time per frame



(b) Rendering time for point based vegetation only

Figure 2: Predicted and actual rendering times for a fly–over path. The rendering budget was 100ms per frame. The solid lines depict actual rendering times, whereas the dashed lines represent predicted frame times.

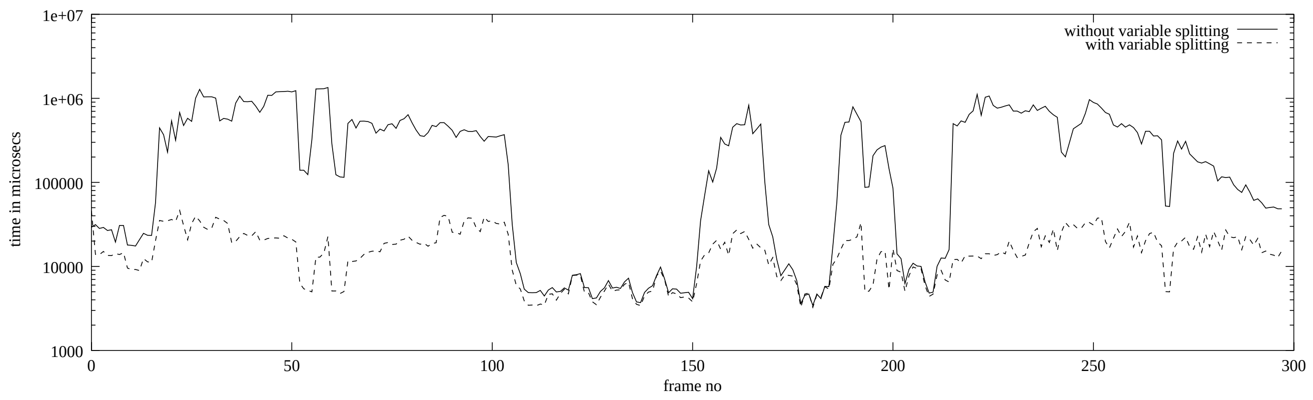


Figure 3: LOD selection times for the basic LOD selection method (solid line) and the enhanced method with variable splitting (dashed line). The available rendering budget was 100ms in both cases.

[10] P. Lindstrom and V. Pascucci. Visualization of large terrains made easy. In *Proceedings of IEEE Visualization 2001*, pages 363–370, 2001.

[11] Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In *Symposium on Interactive 3D Graphics*, pages 95–102, 211, 1995.

[12] A. E. W. Mason. *Predictive Hierarchical Level of Detail Optimization*. PhD thesis, University of Cape Town, 1999.

[13] A. E. W. Mason and E. H. Blake. Automatic hierarchical level of detail optimization in computer animation. *Computer Graphics Forum*, 16(3):191–200, 1997.

[14] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of SIGGRAPH 2000*, pages 335–342, 2000.

[15] M. Reddy, Y. Leclere, L. Iverson, and N. Bletter. TerraVision II: visualizing massive terrain databases in VRML. *IEEE Computer Graphics and Applications (Special Issue on VRML)*, 19(2):30–38, 1999.

[16] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of SIGGRAPH 2000*, pages 343–352, 2000.

[17] D. Schmalstieg and A. Fuhrmann. Coarse view-dependent levels of detail for hierarchical and deformable models. Technical report, Vienna University of Technology, 1999.

[18] J. Weber and J. Penn. Creation and rendering of realistic trees. In *Proceedings of SIGGRAPH '95*, pages 119–128, 1995.

[19] L. Westover. Interactive volume rendering. In *Volume Visualization Workshop*, pages 9–16, 1989.

[20] M. Wimmer and D. Schmalstieg. Load balancing for smooth LODs. Technical report, Vienna University of Technology, 1998.

APPENDIX

A. THE CONTINUOUS VARIABLE SPLITTING METHOD

Let $o_i, i \in \{1, \ldots, n\}$ be a set of objects with smooth LODs. If x_i is the resolution assigned to o_i, then the total benefit is $\sum c_i \sqrt{x_i}$ for suitable c_i (essentially proportional to the square of the estimated screen size, see Section 5.3). Our goal is the estimation of the maximal benefit loss, if we replace the assigned resolutions x_i by the mean $\sum x_i/n$. Further we assume, that $\sum x_i$ is bounded by the available budget X. Overall we search for the solution of

$$\begin{aligned} \max \sum c_i \sqrt{x_i} \quad &- \quad \sum c_i \sqrt{\frac{X}{n}} \quad \text{s.t.} \\ \sum x_i &= X \qquad \text{the budget is limited} \\ \sum c_i &= C \qquad \text{the screen extent is bounded} \\ x_i &\geq 0 \\ c_i &\geq 0 \end{aligned}$$

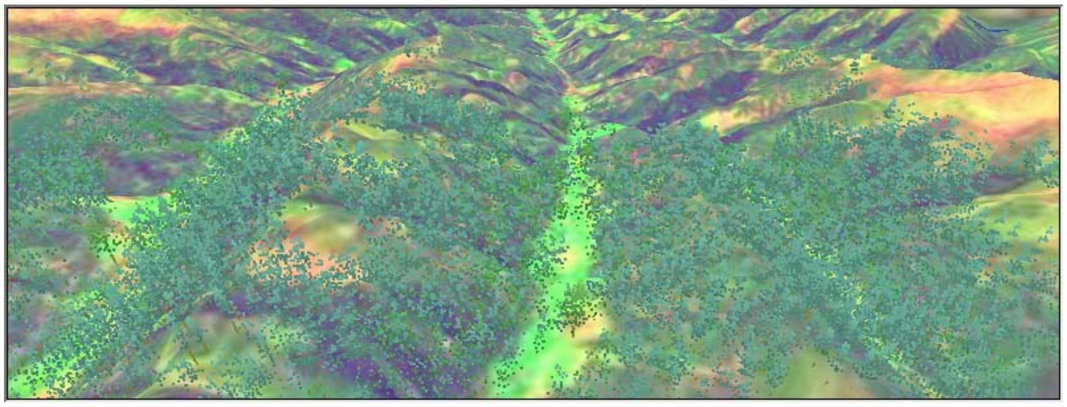
The maximum is attained if $c_1 = C$ and $c_i = 0$ for $i > 1$. This implies, that $x_1 = X$ and $x_i = 0$ for $i > 1$. In this case the error is

$$C\sqrt{X}(1 - \frac{1}{\sqrt{n}}).$$

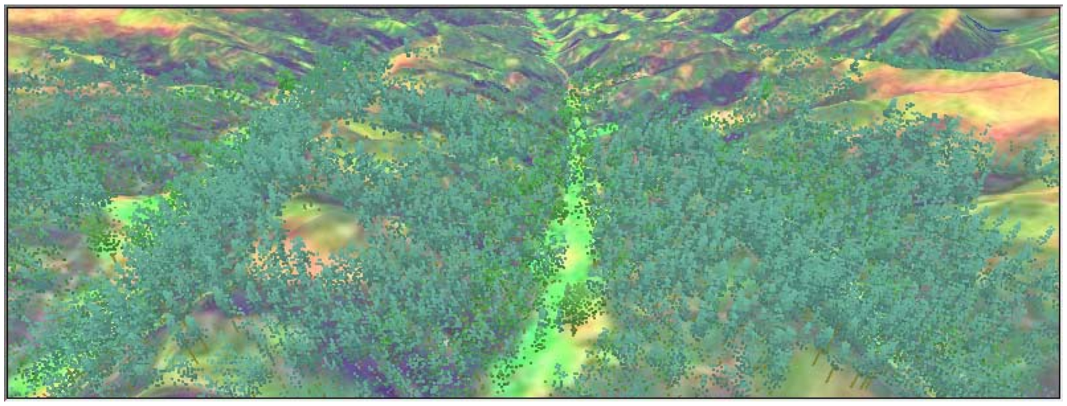
This observation can be used to guide the process of splitting continuous variables. The recursive procedure to associate continuous variables with groups of smooth LOD representations is given in Algorithm 1. Instead of using the same total resolution X (which can be computed from the available rendering budget), we estimate the allowed maximal resolution for every child node.

Algorithm 1 Determine the required granularity of smooth variables

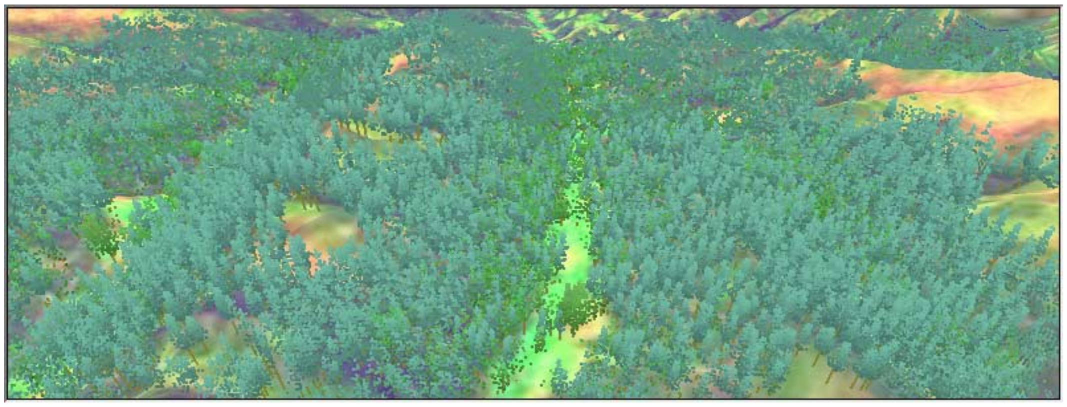
```
Function SplitVar
Input: $Node$, $X$, depth $d$, tree height $d_{max}$, $\varepsilon$
  $S \leftarrow$ screen coverage of $Node$
  if $k\, S^2 \sqrt{X} \left(1 - \frac{1}{\sqrt{2^{d_{max}-d}}}\right) < \frac{\varepsilon}{2^{d_{max}-d}}$ then
    Associate a continuous variable with $Node$
  else
    {Solve the continuous LOD selection problem for the children $Child_i$ of $Node$ and obtain $X_1, \ldots, X_4$}
    {The solution can be directly calculated using Lagrange multipliers}
    $S_i \leftarrow$ screen coverage of $Child_i$
    $X_i \leftarrow X / \sum S_i^4$
    for $i = 1$ to 4 do
      Call SplitVar($Child_i$, $X_i$, $d + 1$, $\varepsilon$) recursively
    end for
  end if
```



(a) 80ms



(b) 100ms



(c) 150ms

Figure 4: A view on the terrain with different rendering budgets. The size of the point splats is 2 pixels. (This figure is reproduced in color on page 209.)