DISSERTATION

HLOD Refinement Driven by Hardware Occlusion Queries

Jean Pierre Charalambos[1,2,3]

Advisor: Dr. Eduardo Romero[1]
Co-advisor: Dr. Michael Wimmer[2]

December 2007

[1]Universidad Nacional de Colombia, [2]Vienna University of technology
[3]Universidad Politécnica de Cataluña

A quien se vea solo, para que se recoja, insista en lo que cree y aguarde la maravillosa ocasión de dar con quien lo complemente.

Resumen

Con objeto de realizar una eficiente visualización interactiva de modelos geométricos complejos, que pueden llegar a comprender varios millones de polígonos, es clave reducir sustancialmente la cantidad de datos procesados. Los métodos LOD (*"level-of-detail"*) permiten efectuar una agresiva reducción de los datos enviados a la GPU, a expensas de sacrificar algo de calidad visual. Particularmente, los métodos HLOD (*"hierarchical level-of-detail"*), en los que se precomputan LODs en los distintos niveles de una jerarquía espacial dada, han demostrado ser el más eficiente enfoque para la visualización interactiva de este tipo de modelos. Además de soportar en forma directa algoritmos *"out-of-core"* (en los que durante tiempo de ejecución los datos deben ser constantemente leídos desde la memoria secundaria del sistema), los métodos HLODs permiten efectuar una carga óptima de los datos entre la CPU y la GPU.

Obedeciendo al mismo objeto, un enfoque ortogonal al anterior es *"occlusion culling"* (descarte por oclusión). Respecto a un punto de vista dado se busca descartar de modo eficiente las partes invisibles de la escena y en visualizar solo sus partes visibles. Los métodos más recientes pertenecientes a esta categoría emplean HOQs (*"hardware occlusion queries"*).

Los efectos relativos a HLODs y occlussion culling pueden combinarse de modo efectivo. Primero, es posible descartar aquellos nodos de la jerarquía que resulten invisibles. Segundo, para los nodos visibles, es posible emplear los resultados de los HOQs como parte integral de la condición de refinamiento de la jerarquía: de acuerdo con el grado de visibilidad de un nodo dado y teniendo en cuenta un fenómeno de la percepción denominado *"visual masking"*, es factible determinar cuando no habría una ganancia apreciable en la apariencia final de la imagen obtenida si el nodo fuera refinado ulteriormente. En este caso, HOQs permiten reducir aún más agresivamente el total de primitivas visualizadas. Sin embargo, debido a la latencia presente entre el momento de iniciar el HOQ y la disponibilidad de su resultado, el uso directo de HOQs en las condiciones de refinamiento resultaría siendo una fuente de estancamiento de la CPU, lo que a su vez redundaría en un apreciable desaprovechamiento de la GPU.

En esta tesis presentamos una novedosa métrica que emplea información de visibilidad (determinada a partir de HOQs) como parte integral de la condición de refinamiento de un modelo HLOD (de nuestro conocimiento, el primer enfoque en este contexto con esta meta en mente). También contribuimos con un novedoso algoritmo para atravesar la jerarquía del HLOD que permite sacar el máximo provecho de esta métrica. A partir de una rutina básica de predicción de la condición de refinamiento del HLOD, el algoritmo minimiza el estancamiento de la CPU y permite obtener así un mejor aprovechamiento de la GPU.

Las principales propiedades expuestas en nuestro enfoque combinado son: 1. Mejor

rendimiento con la misma calidad visual: mediante nuestro sistema es posible visualizar un menor número de primitivas (no por ello nuestra técnica de occlusion culling deja de ser *conservativa*) con una pérdida mínima en la calidad visual del modelo; 2. Generalidad: nuestra métrica soporta cualquier tipo de HLOD; 3. Uso integral de los resultados obtenidos en HOQs: nuestra métrica aprovecha de modo completo la información obtenida mediante HOQs; 4. Aprovechamiento integral de la coherencia espacio-temporal inherente a las representaciones jerárquicas; y, 5. Implementación directa.

Abstract

In order to achieve interactive rendering of complex models comprising several millions of polygons, the amount of processed data has to be substantially reduced. Level-of-detail (LOD) methods allow the amount of data sent to the GPU to be aggressively reduced at the expense of sacrificing image quality. Hierarchical level-of-detail (HLOD) methods have proved particularly capable of interactive visualisation of huge data sets by precomputing levels-of-detail at different levels of a spatial hierarchy. HLODs support out-of-core algorithms in a straightforward way and allow an optimal balance between CPU and GPU load during rendering.

Occlusion culling represents an orthogonal approach for reducing the amount of rendered primitives. Occlusion culling methods aim to quickly cull the invisible part of the model and render only its visible part. Most recent methods use hardware occlusion queries (HOQs) to achieve this task.

The effects of HLODs and occlusion culling can be successfully combined. Firstly, nodes which are completely invisible can be culled. Secondly, HOQ results can be used for visible nodes when refining an HLOD model; according to the degree of visibility of a node and the visual masking perceptual phenomenon, then it could be determined that there would be no gain in the final appearance of the image obtained if the node were further refined. In the latter case, HOQs allow more aggressive culling of the HLOD hierarchy, further reducing the amount of rendered primitives. However, due to the latency between issuing an HOQ and the availability of its result, the direct use of HOQs for refinement criteria cause CPU stalls and GPU starvation.

This thesis introduces a novel error metric, taking visibility information (gathered from HOQs) as an integral part of refining an HLOD model, this being the first approach within this context to the best of our knowledge. A novel traversal algorithm for HLOD refinement is also presented for taking full advantage of the introduced HOQ-based error metric. The algorithm minimises CPU stalls and GPU starvation by predicting HLOD refinement conditions using spatio-temporal coherence of visibility.

Some properties of the combined approach presented here involve improved performance having the same visual quality (whilst our occlusion culling technique still remained conservative). Our error metric supports both polygon-based and point-based HLODs, ensuring full use of HOQ results (our error metrics take full advantage of the information gathered in HOQs). Our traversal algorithm makes full use of the spatial and temporal coherency inherent in hierarchical representations. Our approach can be straightforwardly implemented.

Acknowledgements

I would like to thank Professors Romero and Wimmer for their comments and ideas regarding this thesis; it has been always a pleasure to work with them.

My special thanks go to Dr. Jiří Bittner whose invaluable collaboration made this work possible.

I would like to thank Professor Arinyo for his constructive criticism regarding the thesis proposal and the doctorate scholarship holders at LSI-UPC for the good working climate. I am also grateful to Professor Purgathofer for inviting me to the Institute of Computer Graphics and Algorithms at Vienna University of Technology where I found a very comfortable and collaborative environment necessary for finishing this thesis.

I am grateful to Mariana Renthel for her ideas regarding the historical notes in the introductory chapter. I also want to thank Katalin Koncz and Sarah McCarthy for their criticism regarding the style of my first two papers in the field of this dissertation.

I also want to thank the following people, although I never had the pleasure to meet them in person. Dr. Jiří Žára for sharing his thoughts with me since the brainstorming times of my dissertation activities and Gilles Debunne and Mario Botsch for their assistance with the libQGLViewer software library and the OpenMesh software library, respectively.

Special thanks also go to my mother and to all friends who have always believed that one day I would make this work see the light of day.

Contents

1 Introduction

1.1 Historical notes

Convincing images

The concept of illusion has always been bound up with the history of art and the development of thought regarding image handling. "The best artists are those who lie the best", said Bembenuto Celini, as sometimes reality and fiction wish to be confused.

Plato referred to art as representation in terms of *mimesis* (imitation), imitating to trick the eye to make the brain believe some things exist. A very well-known painting effect used in baroque times to open up spaces (known as the *trompe-lóeil*) confused the common people whilst enriching the great houses just to make excess the main course.

What really matters in image representation then? Representing the concept through sensations is not just the most realistic situation but the most credible one, going far beyond realism. Credibility is obtained by regulating the degree of attention, focusing the viewer's eye and creating atmosphere. Several resources helped artists with the task of representation, ranging from perspective to a bird's eye view; nonetheless, acknowledging atmosphere is quite interesting regarding the subject of this thesis. The contrasting works from Leonardo da Vinci and Salvador Dalí reveal how art history has developed regarding representation.

An opened image ensures the construction of a picture's atmosphere. *Sfumato* was the term Leonardo used for describing how to gride the mass into particles, a figure within a background. It mainly consisted of using dissolution, thus defining the background from a figure in the case of landscapes and thereby creating three-dimensionality. The paint brush was not just used to apply colour, keeping quantities equal on the surface, but as an airbrush or device for creating a blurred background, thereby creating transparent layers. The result not only involved introducing atmosphere into the piece but also focused on whatever element or subject the artist desired to be highlighted. Such strategy was based on the economy of elements and synthesis of objects, thereby ensuring clarity and credibility; *The Virgin and Child with St. Anne* provides a good example of this approach (Figure 1.1). Disposed within an invisibly-shaped triangle we can see three figures forming the entrance to a mystical world spreading out beneath them. The figures are the most important element here, even though the real grace lies in the involving environment as determined by unfinished pieces of landscape. However, if we enter this tripartite form and look closely at a detail such as a hand or an eye we could also appreciate the same treatment.

As art evolved, abstraction began to provide other aspects within reality, according to the concept which an artist wished to portray not the image itself; in surrealism the

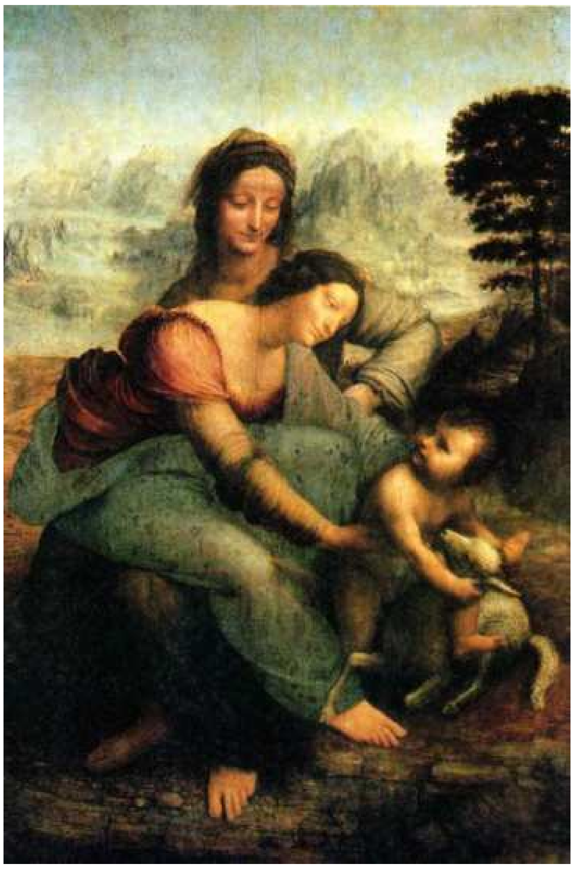

Figure 1.1: Leonardo Da Vinci: The Virgin and Child with St. Anne. Near 1508, oil on wood, 168 × 112 cm., Louvre, Paris.

oneiric played a main character on representation. In the particular and complex case of Dalí's work, we can mark the treatment not only involving the degree of specificity in background and landscapes but also dual images. Of course, as stated, this was indeed the result of a complex concept to be represented which had to respond to dreams and symbolism and also leave space for free interpretation. In the case of *Swans Reflecting Elephants* (see Figure 1.2) we can appreciate very realistic production in the central figures and the image of the man on the left, even though it does not leave aside the duality and the level of specificity used in the background. We can thus have at least two paintings in one. The shaded tones on the sky, continuing or extending the horizon and returning in the form of the lake, involving rocky formations shaped as a crusty but melting substance, seem to be the renewed legacy of Leonardo's *sfumato*.

Computer graphics rendering

Rendering is the process of generating a *convincing* image from a model by means of computer programmes within the context of computer science. The model is a description

Figure 1.2: Salvador Dalí: Swans Reflecting Elephants. 1937, oil on canvas, 51 x 77 cm., Cavalieri Holding Co. Inc., Geneva.

of three-dimensional objects in a strictly-defined language which may contain geometry, texture, lighting and shading information. The term may be an analogy of an "artist's rendering" of a scene. With the increasing sophistication of computer graphics since the 1970's, rendering has now become a more distinct subject.

Rendering has uses in architecture, medicine, simulators, video games, special effects in movies and design visualisation, each employing a different balance of features and techniques. A wide variety of *rendering engines* is available nowadays; some are integrated into larger modelling and animation packages, some are stand-alone whilst others are free open-source projects [Wik07].

A scene may be slowly rendered, as in prerendering, or in real time. Whilst prerendering is a computationally intensive process typically used for movie creation, real-time rendering is more suitable for applications requiring interactivity, such a 3D video game. Real-time rendering requires 3D hardware acceleration which is provided by special purpose graphics cards (nowadays ubiquitously available).

Rendering techniques

Since tracing every ray of light in a scene to produce an image is impractical, due to the gigantic amounts of time this would require, more efficient techniques for modelling the transport of light have emerged over the last decades. These techniques fall into the following categories:

- *Rasterisation*: this method loops every frame through each of the geometric primitives comprising the 3D model to determine which pixels in the image it affects, modifying those pixels accordingly. This rendering method is the one used by all current graphics cards. Rasterisation is thus usually the fastest approach and hence it is the main choice when interactive rendering is required.

- *Ray casting*: this method casts rays from the viewpoint, one per pixel, and find the closest object blocking the path of that ray. When an object is intersected, the colour value at that point may be evaluated by using several methods. In the simplest, the colour value of the object at the point of intersection becomes the value of that pixel. The colour may also be determined from a texture-map or be modified by an illumination factor.

- *Radiosity*: also known as *global illumination*, is a method which uses finite element mathematics to simulate the way in which directly illuminated surfaces act as indirect light sources illuminating other surfaces [Wik07]. This produces more realistic shading and seems to better capture the *ambience* of an indoor scene.

- *Ray tracing*: this category is similar to ray casting, but it employs more advanced optical simulation leading to more realistic results at a speed which is often orders of magnitude slower. A ray tracing renderer usually uses Monte Carlo techniques for averaging a number of randomly-generated samples (imaginary rays of light intersecting the viewpoint from the objects in the scene) from a model.

Most advanced software could combine two or even more of these techniques to obtain good-enough results at reasonable cost.

1.2 Problem domain of this thesis

One of the fundamental problems of computer graphics lies in rendering very large polygonal datasets at interactive rates. Time critical rendering applications arise in several arenas such as architectural walkthroughs, medicine, video games, range scanners, terrain rendering, CAD applications and numerical simulation. However, despite the rapid improvement in hardware performance, the size of the models to be rendered largely overload the performance and memory capacity of state-of-the-art graphics and computational platforms.

The amount of processed data must therefore be substantially reduced to achieve interactive rendering of such models. Level-of-detail (LOD) methods allow aggressive reduction of the amount of data sent to the GPU at the expense of sacrificing image quality. Hierarchical level-of-detail (HLOD) methods have particularly proved capable of the interactive visualisation of huge data sets by precomputing levels-of-detail at different levels of the spatial hierarchy. HLODs support out-of-core algorithms in a straightforward way and allow an optimal balance between CPU and GPU load during rendering.

Occlusion culling represents an orthogonal approach for reducing the amount of rendered primitives. Occlusion culling methods aim to quickly cull the invisible part of the model and render only its visible part. Most recent methods employ hardware occlusion queries (HOQs) for achieving this task.

Combining HLODs and HOQs

The effects of HLODs and occlusion culling can be effectively combined in two different ways:

1. The HLOD hierarchy nodes which are completely invisible can be culled. The problem of using HOQs to cull a general scene hierarchy has been recently studied to some extent.

2. HOQ results can be used in the refinement condition for visible nodes of the HLOD hierarchy. In this case, the occlusion queries allow more aggressive culling of the HLOD hierarchy, further reducing the amount of rendered primitives. However, due to the latency between issuing an HOQ and the availability of its result, the direct use of HOQs for refinement criteria causes CPU stalls and GPU starvation.

1.3 Purpose and outline of this thesis

The main goal was to improve HLOD-based system performance by taking visibility information (gathered from HOQ results) into account within refinement criteria while preserving image quality results. An HOQ-based model was sought which took the degree of visibility of a given node into account and also attempted to exploit the visual masking perceptual phenomenon. To our knowledge, no HLOD approaches have so far had this goal in mind. Previous approaches have treated refinement criteria and occlusion culling as independent subjects. The visibility information potentially gathered in most recent occlusion culling approaches has thus been restrictively used, simply to cull the invisible nodes of the HLOD hierarchy (see item 1 above).

Two challenges had to be identified to achieve our goal and two contributions were posed for each. One of the challenges related to defining the visibility-based error metric needed to refine a given HLOD and the other related to designing the HLOD hierarchy traversal algorithm. Since one of the contributions regarding the visibility-based error metric became nicely reduced into the other, it was possible to introduce them as a single entity (see Section 6.4). Since they represent two different approaches, the two contributions regarding the traversal algorithm were a different story. However, instead of only presenting the best approach found (see Section 7.2.2), it was decided to introduce both of them here to show the evolution of our ideas while trying to keep the discussion as consistent as possible (see Sections 7.2.1 and 7.2.2).

A general overview of the state of the art concerning the real-time visualisation of complex models (see Chapter 2) is first given before presenting our own contribution, focusing more closely on HLODs and HOQs (see Chapters 3 and 4, respectively). The two groups of contributions are then outlined (see Chapters 6 and 7, respectively) and their viability demonstrated by testing them in various representative settings (see Chapter 8). The thesis concludes by outlining future directions for related research (see Chapter 9).

2 Real-time visualisation of complex models

The amount of processed data has to be substantially reduced to achieve interactive rendering of complex models comprising several million polygons. Two of the most common approaches (orthogonal between each other) to accomplishing this task are described in this chapter: LOD (see Section 2.1) and occlusion culling (see Section 2.2). HLODs and HOQs are discussed separately in following chapters since they are of central importance to this work, being particular cases of LOD and occlusion culling respectively (Chapter 3 and 4, respectively).

2.1 Level-of-detail (LOD)

The original level-of-detail (LOD) scheme proposed by Clark [Cla76] (nowadays referred to as *discrete LOD* [LWC*02]) creates multiple versions of *interesting* objects within a scene during a preprocess, each one having a different level of detail. The appropriate LOD is chosen to represent the object at runtime (see Section 2.1.2). Since distant objects use coarser LODs the total number of polygons is reduced and rendering speed increased (see Figure 2.1).

The simplification is unaware of viewing directions since LODs are computed offline during preprocessing; it therefore uniformly reduces object detail. Discrete LOD are thus also referred to as *isotropic* or *view-independent LOD* [LWC*02]. The two main advantages of discrete LODs are: 1. *Easy to use*: decoupling simplification and rendering makes this the simplest model to programme. Independently of the simplification, the runtime rendering algorithm simply needs to choose which LOD to render for each object (see section 2.1.2); and 2. *Offline optimisation*: depending on the particular hardware targeted, LODs can be easily converted to an optimal rendering format (which will render much faster than simply rendering the LODs as a triangle soup) such as triangle strips, display lists, vertex arrays and vertex buffer objects.

2.1.1 View-dependent LOD

View-dependent LOD departs from the traditional view-independent LOD approach. Rather than creating individual LODs during the preprocessing stage (see above), the simplification of a view-dependent LOD creates a data structure encoding a continuous spectrum of detail [Hop96]. The structure stores the evolution of a mesh throughout a sequence of local modifications performed to simplify the model [PS97].

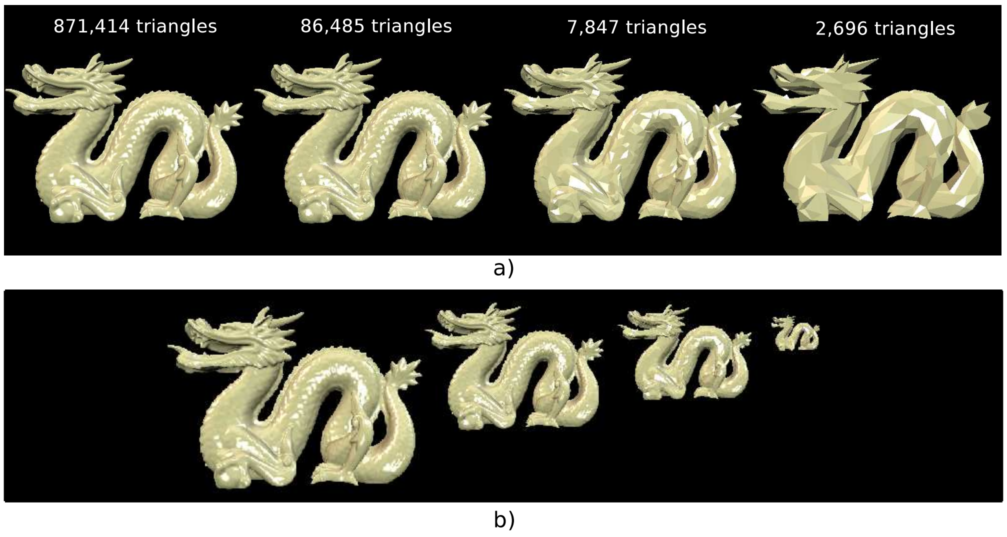

Figure 2.1: The fundamental concept of LOD. (a) A complex object is simplified, (b) Creating LODs to reduce the rendering cost of small, distant, or unimportant geometry [LWC*02]. Model courtesy of Stanford Graphics Group [SU].

The desired LOD is then extracted from this structure at runtime. The rendering algorithm uses view-dependent simplification criteria to dynamically select the most appropriate LOD for the current view [Hop97, LE97, XESV97, KL01] (see Section 2.1.2.3). View-dependent LOD is therefore *anisotropic*: a single object can span multiple levels of simplification [LWC*02]; for instance, nearby portions of the object may be shown at higher resolution than distant portions (see Figure 2.2). This leads to the following advantages over view-independent LOD: 1. *Better granularity*, since polygons are allocated where they are most needed rather than selected from a few previously created options; no more polygons are used than necessary; and 2. *Better fidelity for a given polygon count* thereby optimising distribution of this scarce resource.

View-dependent LOD can enable interactive rendering of complex models (interactive rendering of very complex models is discussed in greater detail in Chapter 3) which cannot otherwise be suitably simplified using view-independent techniques. In this case, the compromise between fidelity and frame rate becomes unaffordable.

2.1.2 LOD selection factors

The most important issue in LOD management is to decide when to switch to a lower or higher resolution model. A brief overview is given of the main selection mechanisms which have been used to modulate LOD, such as object space factors, screen space factors and visibility information. HLOD refinement criteria is discussed in detail in Section 3.3 and our proposed visibility-based error metric for HLOD refinement is discussed in Chapter 6.

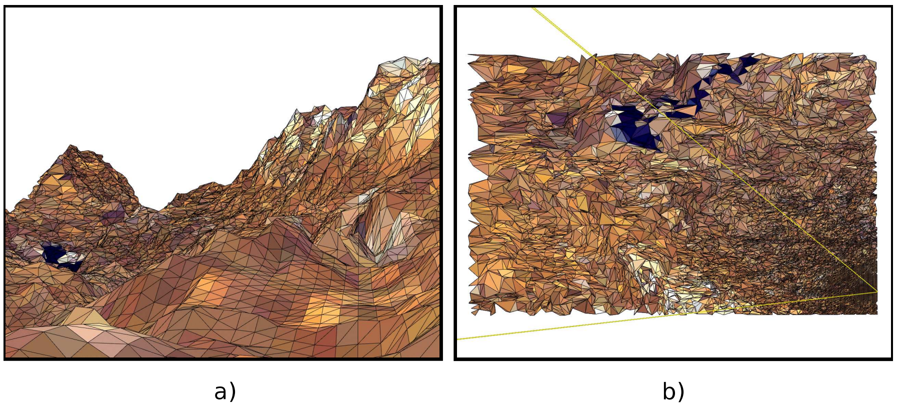

Figure 2.2: View-dependent LODs show nearby portions of the object at higher resolution than distant portions. (a) View from eyepoint, (b) Bird's eye view. From [Lue01]. Copyright © 2001 IEEE.

2.1.2.1 Distance

Distance has been the main LOD selection criterion used in *object space*. The idea is simple; since fewer high-detail features of a distant object are visible, a lower LOD could then be selected without greatly affecting the fidelity of the image [LWC*02]. The distance from the viewer to an object is compared to a given distance threshold when properly selecting an LOD. Whilst distance thresholds are separate and previously assigned for each LOD comprising the object, the viewer's object space distance to the object is computed from the viewpoint to an arbitrary point within the object, such as its centroid.

Distance-based LOD is both simple and efficient and is more suitable for view-independent LOD; a few conditional statements suffice to check whether a distance exceeds the predefined thresholds. However, since the actual distance to the viewpoint can change depending on orientation, the arbitrary choice of the point within the object for all distance calculations gives poor accuracy leading to obvious popping artifacts [LWC*02].

2.1.2.2 Size

Size-based LOD techniques use the projected screen coverage of an object since objects become smaller as they move further away and switch between LOD based on a series of size thresholds [LWC*02]. Since projecting the object itself would be too expensive, current methods project instead a *bounding volume* of the object (such as its bounding box or its bounding sphere). Size-based techniques avoid some of the problems with distance-based LOD since projected size is insensitive to display resolution, object scaling or field of view. Moreover, size-based LOD selection uses the entire object thereby providing more generic and accurate means for modulating LOD than distance-based techniques [LWC*02].

2.1.2.3 Screen space error

The *screen space error* (SSE) metric uses the projection onto screen space of the *model space error* (MSE, see below) and switch between LOD if the magnitude of the projection falls below an user-defined tolerance, given in pixels (sometimes known as *pixels of error* [YSGM04]). Some possible MSE computations are presented, followed by a method for computing an SSE upper bound.

MSE **computation.** The MSE (sometimes also referred to as *geometric error*) is a measurement of the difference between two models, one being a simplified version of the other. There are several possible ways to compute the MSE. Suppose model A is simplified to obtain model B for the following definitions:

- *Huasdorff distance:* The Hausdorff distance ($H(A, B)$) is the max of min distances between points in the two models [LWC*02]. The Hausdorff distance may be expressed algebraically as: $H(A, B) = max(h(A, B), h(B, A))$, where $h(A, B) = \underset{a \in A}{max} \underset{b \in B}{min} \parallel a - b \parallel$. The Hausdorff distance is (by construction) the tightest possible bound on the maximum distance between two models.
- *Mapping distance:* Given a continuous one-to-one mapping between two models, the mapping distance is the distance regarding this mapping [LWC*02], i.e., Let $F : A \to B$ be such a continuous mapping, the mapping distance is defined as: $D(F) = \underset{a \in A}{max} \parallel a - F(a) \parallel$.
- *Quadric error metrics:* Given a vertex, $v = (x, y, z, 1)$, and a plane, $p = (n_x, n_y, n_z, D)$, the quadric error metric (E_v) is the sum of squared vertex–plane distances [GH97], i.e., $E_v = \sum_{p \in planes(v)} (p \cdot v)^2$. The quadric error metric measures the cumulative surface distortion introduced by a series of vertex merge operations, which are accomplished during the simplification of the model. The MSE could be approximated from E_v by the following expression: $s \sqrt[3]{\underset{v \in B}{max} E_v}$, where s is an empirical scale factor for converting to world units [Lin03]. The scale factor is determined prior to rendering time by finding the smallest value of s leading to no image difference in a fixed number of random views, when setting the screen space tolerance below 1 pixel [CGG*04].

SSE **upper bound.** An upper SSE bound could be obtained by projecting the MSE from the point on the minimum bounding sphere of the LOD closest to the viewpoint (v). This could easily be achieved by projecting a sphere centered at v and with a diameter equals to the magnitude of the MSE and then counting the pixels of the screen x-axis aligned diameter (see Figure 2.3). The LOD may actually be the entire object (in the case of a discrete LOD system) or just a small surface portion of the object (in the case of a view-dependent LOD system) [LWC*02]. The error metric should be monotonic down the LOD structure to preserve the consistency in the latter case; any criterion which would refine a node should also refine its parent (see Section 3.3).

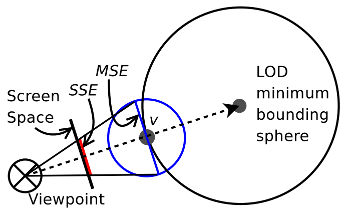

Figure 2.3: Computation of an SSE upper bound.

2.1.2.4 Visibility

The exclusive use of view-parameters to perform LOD selection causes even occluded regions to be rendered with a high level of detail. Visibility-based LOD selection techniques aim to overcome this serious drawback by integrating occlusion culling information into the LOD selection mechanism [FS93] (occlusion culling is discussed separately in Section 2.2). However, there have been few approaches which integrated LODs with occlusion culling because computing exact visibility has been too expensive to be performed in real time until recently (see Section 2.2 and Chapter 4). Nonetheless, two notable exceptions based on visibility estimation techniques would be:

1. *Hardly visible sets*: in the context of discrete LOD, Andújar *et al.,* divide the scene into *cells* and classify the objects into sets according to their visibility degree for each cell [ASVNB00]. A representative error which measures the possible contribution to the final image in pixels is then assigned to each set. The sets related to each cell are obtained during a preprocess from the selection of some objects which act as occluders. During runtime, the error is used for discarding the objects of the sets not meeting an user-defined threshold or, for objects having a discrete LOD representation, to select the proper LOD needed to display them.

2. *Cell solidity values:* in the context of view-dependent LOD, El-Sana *et al.*, impose a 3D grid over the scene and precompute *solidity values* for each cell [ESSS01]. These values are view-dependent indicators for how a cell occludes other cells from the imposed 3D grid [KS00, KS01]. The solidity values are used during runtime to estimate the occlusion probability of a given node in the view-dependent LOD structure. The LOD at a node is then determined by computing its view-parameter contribution and its occlusion probability estimation.

2.1.2.5 Visual masking

Visual masking describes the perceptual phenomenon that the presence of one visual pattern can affect the visibility of another pattern [FSPG97, KB05]. For example, a large adjacent stimulus (in time or space) can raise the threshold of a smaller stimulus, meaning that the smaller stimulus must be more intense to be visible. In terms of computer graphics, this means that a visual texture can mask small details or aliasing artifacts in an image [BM95] (see Figure 2.4) which could then be exploited for LOD selection [LWC*02].

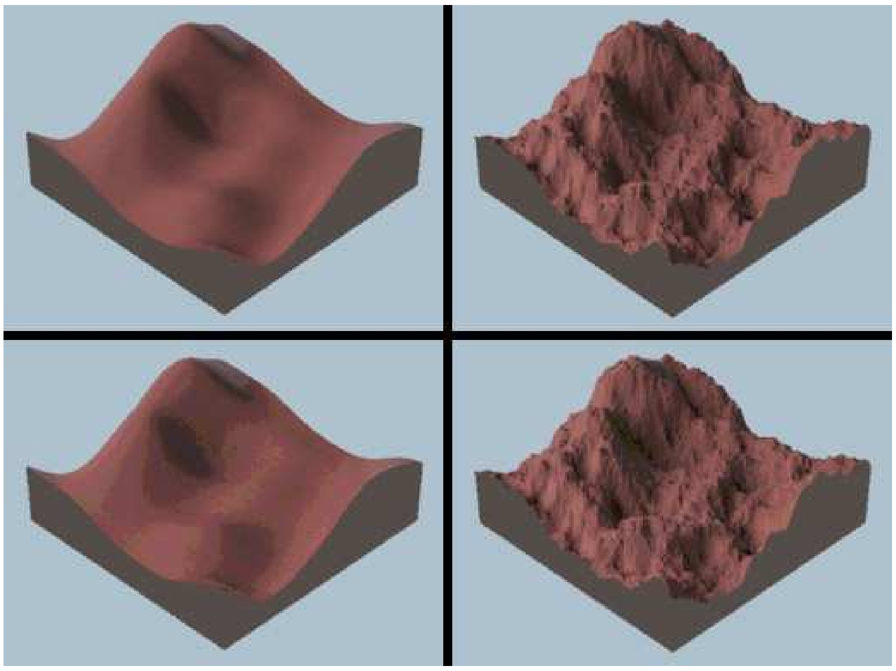

Figure 2.4: Masking in computer graphics. The upper pair images are quantised to 8 bits; the lower pair to 4 bits. Banding is visible in the smooth surface on the lower left but not in the rough surface on the lower right due to masking effects from the visual texture created by the rough surface. From [BM95]. Copyright © 1995 Association for Computing Machinery, Inc.

2.1.2.6 Other factors

The following are other LOD selection criteria which researchers have proposed over the last few decades (a complete overview of these factors lies beyond the scope of this thesis. For a more thorough discussion of this subject readers may refer to the work of Luebke *et al.,* [LWC*02]):

1. *Priority:* giving a priority ranking to objects according to their role in a scene could allow objects considered as most important to be degraded the least.

2. *Focus and velocity:* as their names suggest, whilst focus-based LODs consist of simplifying objects in an user's peripheral vision more aggressively than objects under direct attention, velocity-based LODs consist of simplifying objects moving quickly more aggressively than slow-moving objects.

3. *Environmental conditions:* atmospheric effects such as haze and fog could help to reduce the appearance of popping artifacts due to the switching between LODs, or to reduce the range over which models need to be displayed [LWC*02].

2.2 Occlusion culling

Visibility culling is an orthogonal approach to LOD for reducing the amount of rendered primitives; it aims at quickly rejecting invisible geometry before performing the actual hidden-surface removal (HSR) which is typically handled with the z-buffer algorithm. Visibility culling strategies consist of *back-face culling*, *view-frustum culling* and *occlusion culling*. Whilst back-face culling algorithms avoid rendering geometry which faces away from the viewer, viewing-frustum culling algorithms avoid rendering geometry which is outside the viewing frustum. On the other hand, occlusion culling techniques aim at avoiding rendering primitives which are occluded by some other part of the scene. This technique involves interrelationships amongst polygons and is thus far more complex than back-face and view-frustum culling [COCSD02]. The three culling strategies are depicted in Figure 2.5.

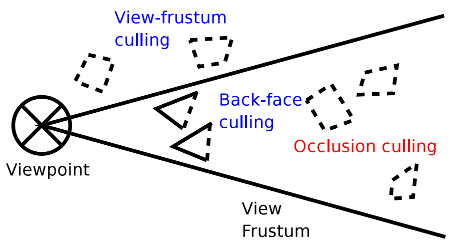

Figure 2.5: Visibility culling strategies.

Typically, the higher the depth complexity of the scene, the larger the amount of geometry occluded and hence the more effective the occlusion culling could be. Starting with the pioneering work of Airey *et al.,* [ARFPB90] and Teller and Séquin [TS91] on precomputing lists of potentially visible geometry from cells, occlusion culling grew into an important area of research in computer graphics. There are several classes of occlusion culling algorithms (a complete overview of occlusion culling lies beyond the scope of this thesis. For a more complete taxonomy the reader may refer to recent surveys, such as the one by Cohen-Or *et al.*, [COCSD02] or that by Durand [Dur00]):

1. *Image-precision* algorithms perform visibility computations with image precision, i.e., using depth values [GKM93, BMH98] or occlusion maps [ZMHH97]. Most recent methods – particular interesting regarding the subject of this thesis – employ hardware occlusion queries (HOQs) [BWPP04, GBK06] (see Chapter 4).

2. *Object-precision* algorithms perform visibility computations with object precision, usually by finding a set of large occluders which can be used to determine if other objects (or a hierarchy of objects) in a scene are visible [CT97, HMC*97].

3. *From-region* algorithms perform visibility computations not just for a single viewpoint but also compute the set of potentially visible cells for a whole region [DDTP00, SDDS00].

Occlusion culling techniques are usually *conservative*, producing images which are identical to those which would result from rendering all of the geometry. However, they can also be approximation techniques producing images which are mostly correct in exchange for even greater levels of interactivity [ZMHH97, KS99, KS00].

3 Hierarchical levels of detail (HLOD)

Hierarchical levels of detail (HLOD) extend the traditional view-dependent LOD approach (previously described in Section 2.1.1) by moving the granularity implicit in hierarchical traversal operations from single primitives to small surface portions. HLOD thus amortises traversal costs over many graphics primitives, allowing interactive visualisation of huge models which otherwise could not be refined in real time using view-dependent LOD techniques. The notion behind HLOD and its main properties are first described, followed by the current HLOD refinement criteria which are central to our study.

3.1 HLOD notion

A hierarchical level of detail (HLOD) system corresponds to a nested subdivision of the space occupied by a given model, where:

1. The subdivision is encoded as a hierarchical data structure.
2. Each node in the hierarchy contains a small surface portion of the region encompassed by the node [EMWVB01]. Geometric data is related to each node in the following manner:
    a) The original geometry of the model is spread amongst leaf nodes.
    b) Each inner node contains one, or few simplified versions, i.e., levels of detail (see Section 2.1), of the geometry contained in their children.

An HLOD system typically consists of two steps: preprocess and runtime. The HLOD hierarchy and the geometric patches related to each node (see Figure 3.1) are built offline during the preprocess. The goal during runtime is to determine the *front of termination nodes* (see Figure 3.2), i.e., nodes of the hierarchy where the traversal is terminated for each frame (see Section 4.2.2). The refinement stops at a given node either because the node is determined to be invisible, i.e., the node is either view frustum culled, back face culled, or occlusion culled, (see Chapter 4), or because it is determined that there would be no gain in the final appearance of the image obtained if the node were further refined (see Section 3.3). The latter mechanism (being the only one in which geometric data is actually rendered) allows local view-dependent refinement of the model.

3.2 Properties

Some of the main properties shared among HLOD-based systems are:

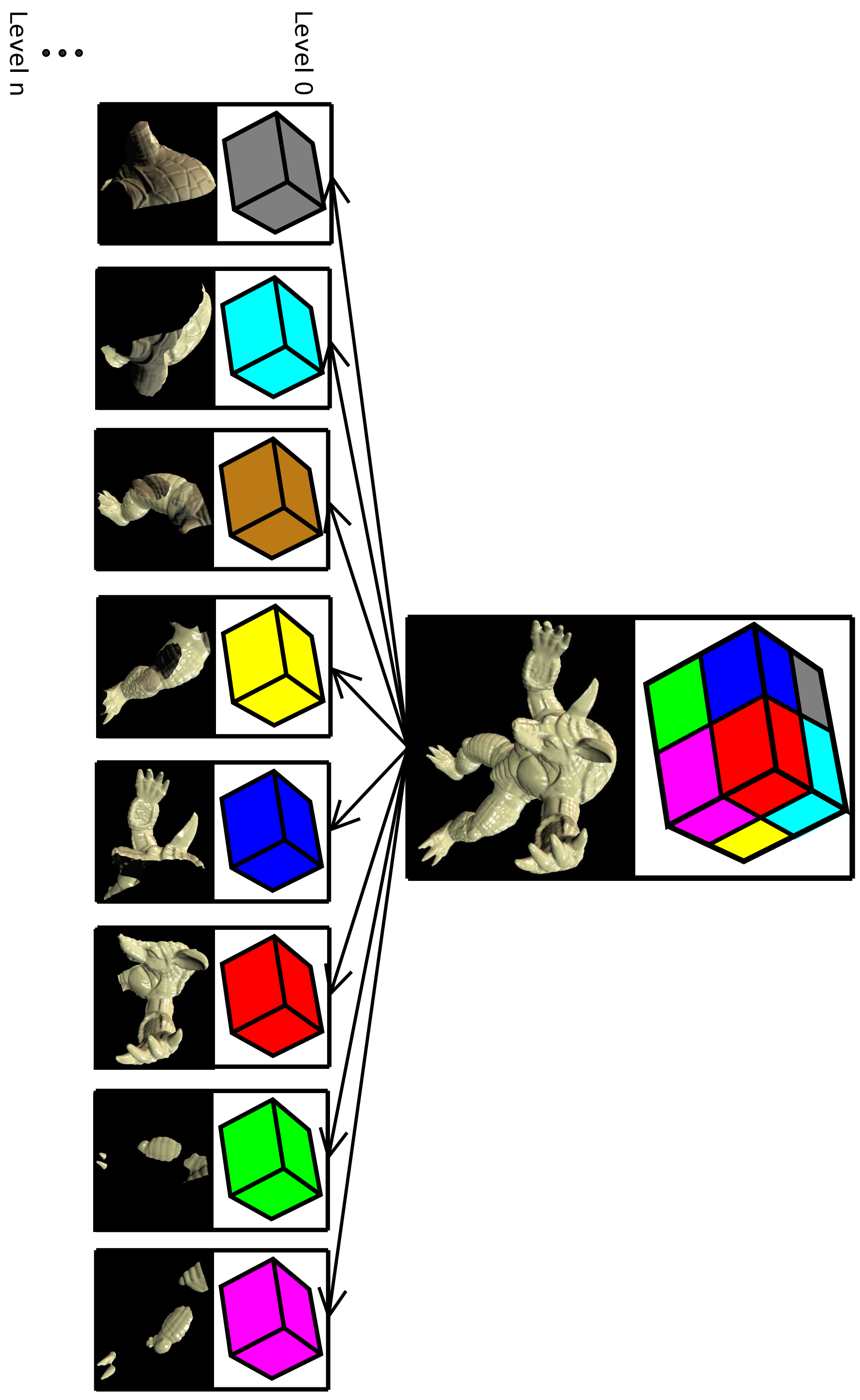

Figure 3.1: Example of a multiresolution structure. Octree with associated LOD hierarchy. Model courtesy of Stanford Graphics Group [SU].

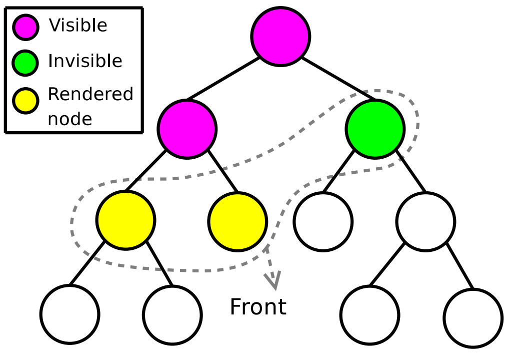

Figure 3.2: Front of termination nodes.

1. *Generality of the scene*: HLOD approaches are not limited to meshes from a particular topological genus. They commonly take a model represented as a triangle soup as input, i.e., a flat list of triangles with direct vertex information.

2. *Cost amortisation*: since the granularity of the HLOD structure is coarse, data management, traversal and occlusion culling cost is amortised over many graphics primitives [CGG*03, CGG*04]. Note that the refinement operations were carried out in view-dependent LOD on a per primitive basis (see Section 2.1.1). Actually, HLOD can enable interactive rendering of gigantic models on current commodity graphics platforms which otherwise could not be refined in real time using view-dependent LOD techniques.

3. *HLOD types*: HLOD systems either use polygonal representation of LODs [CGG*04], point-based representations [RL00, GM05], or a combination of both [GBK03].

4. *HLOD phases:*

   a) *Preprocess*: the goal of the preprocess is to prepare data and perform the optimisations needed to interactively visualise the system at runtime. The preprocess typically consists of two offline steps:

      i. *Constructing the hierarchy*: the scene hierarchy is built in a top-down manner using spatial subdivision driven solely by an user-defined target number of primitives per node.

      ii. *Simplification*: a single, or a few, levels of detail are generated for each node in the hierarchy using offline simplification in a bottom-up manner. The model space error is computed once the node has been simplified and is also kept at the node.

   b) *Runtime*: the hierarchy structure is loaded only once into the main memory during runtime inspection of the model and nodes are fetched from disk and loaded to high-speed GPU memory space using explicit memory management routines (see below). The main selection mechanisms for the nodes comprising the front for termination nodes (see Section 4.2.2) are: view frustum culling, back face culling, occlusion culling (see Chapter 4) and the hierarchical refinement criterion (see Section 3.3).

5. *Memory management*: an explicit memory manager should be implemented for HLODs exceeding memory storage capacity. This could easily be done by implementing a memory cache based on an LRU strategy exploiting the spatio-temporal coherence inherent in hierarchical representations. The idea is that most of the nodes which were processed recently could then be reused in subsequent frames. Depending on the memory needs, there could be two different levels of memory caches:

    a) GPU cache: for storing the most recently rendered nodes. This could easily be implemented in OpenGL using the *vertex buffer objects* extension, thereby allowing optimising data throughput which relies heavily on extensive on-board data caching in the GPU.

    b) RAM cache: for storing the most recently visited nodes. This is necessary for out-of-core rendering of massive models which do not fit into RAM such as range scanners [LPC*00], terrain rendering [LP01], CAD applications [VM02] and numerical simulation [MCC*99]. It is also possible to employ a speculative prefetching technique to prepare data which is likely to be used in the future [CKS03]. This technique is commonly implemented as a separate thread [CGG*04, YSGM04].

6. *Static scenes*: The main limitation inherently presented in HLODs is that they can only deal with static scenes.

3.3 Refinement criteria: model and screen space errors

Local view-dependent refinement of the HLOD model is accomplished by evaluating a refinement condition which commonly uses the *screen space error* (SSE) metric [CGG*03, CGG*04, YSGM04, GM05] (see Section 2.1.2.3). When visiting a node in a top down traversal of the hierarchy (see Section 4.2), the refinement condition may be written as follows: if ($SSE(node) \leq \tau$) then *stop hierarchical refinement* (e.g., [CGG*04]), where: the returning value of the $SSE(node)$ function corresponds to the SSE related to a given *node* and τ corresponds to the user specified threshold given in pixels (sometimes known as *pixels of error* [YSGM04]). It is worth noting that the MSE has typically been approximated from the quadric error metric (see Section 2.1.2.3) [CGG*03, CGG*04, YSGM04, GM05].

An upper bound of the SSE could be obtained as described in Section 2.1.2.3, using the node minimum bounding sphere (built around the node geometry) to project the MSE from (see Figure 3.3). Since the node bounding spheres form a self-contained hierarchy of bounding volumes, this procedure ensures that the error metric is always monotonic down the HLOD hierarchy (see Section 2.1.2.3).

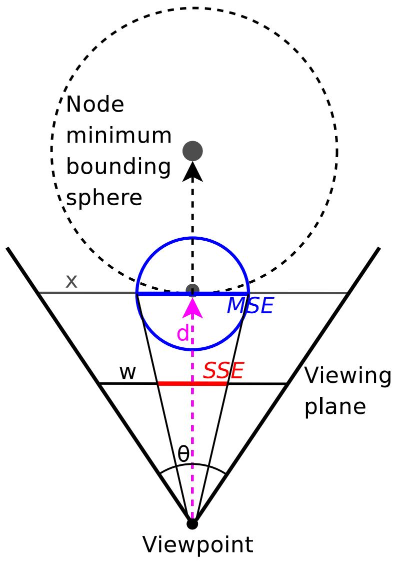

Figure 3.3: Computation of an SSE upper bound in polygon-based HLODs. Under the assumption that the SSE is aligned with the screen x-axis it follows that $SSE = MSE\left(\frac{w}{2d\tan(\theta/2)}\right)$, where w corresponds to the screen width in pixels.

Instead of $MSEs$, point-based HLODs typically use *sizes* for SSE computation (see Section 2.1.2.2), e.g., the projection of the node bounding sphere [RL00], or the projection of the node bounding box [GM05].

Refinement within the nodes

Popping artifacts may appear when refining (or coarsening) a previous visible node in polygon-based HLODs in a given frame during runtime inspection of the model. These visual artifacts are prominent in hierarchies having fast growing subdivision [Pas02]. A few levels of detail are related to each node instead of a single level of detail to avoid their appearance. An additional finer-grained refinement is thus needed within the nodes of the hierarchy [YSGM04]. A range of model space errors $[MSE_{min}, MSE_{max}]$ is first related to each node to perform such refinement (during the simplification step). When visiting a node during runtime, τ is *mapped back* from screen to model space to obtain MSE_τ, i.e., MSE_τ simply corresponds to the value of the model space error leading to $SSE = \tau$ when projected onto the screen. The value of MSE_τ is then used to: 1. Refine the hierarchy by means of the following refinement condition: if $(MSE_{min} \leq MSE_\tau \leq MSE_{max})$ then *stop hierarchical refinement*. 2. Refine the geometry within the node. Hoppe has described how to refine a geometric model when MSE_τ is given [Hop97].

4 Hardware Occlusion Queries (HOQs)

This chapter describes using hardware occlusion queries (HOQs) for occlusion culling which is central to our study (for a general overview of occlusion culling techniques see Section 2.2). The HOQ mechanism is described, followed by a traversal algorithm for a scene arranged as a hierarchy which has been commonly used to deal with the latency presented in HOQs.

4.1 HOQ mechanism

Hardware occlusion queries follow a simple pattern: to test visibility of an occludee, its bounding volume is sent to the GPU. The volume is rasterised (but it is not rendered to screen) and its fragments are compared to the current contents of the z-buffer. The GPU then returns the number of visible fragments. If there is no visible fragment, the occludee is invisible and it need not be rendered [HGJ03, BWPP04].

4.1.1 Properties of HOQs

Currently, the two main supported HOQ properties in OpenGL (see [SA04]) are:

Property 1. Multiple occlusion queries may be sent at once (see Section 4.2).

Property 2. The returning value corresponds to the number of visible pixels of the queried object, but without saying anything about their position (see Chapter 5).

4.1.2 Advantages and disadvantages of HOQs

HOQs have several advantages:

1. *Generality of occluders.* The original scene geometry can be used as occluders, since the queries use the current contents of the z-buffer.
2. *Occluder fusion.* The occluders are merged in the z-buffer so the queries automatically account for occluder fusion. Such fusion comes free since the intermediate result of the rendering itself can be used.
3. *Generality of occludees.* Complex occluders can be used; anything which can be rasterised quickly is suitable.
4. *Better use of GPU power.* The queries take full advantage of the high fill rates and internal parallelism provided by modern GPUs.

5. *Easy use.* Hardware occlusion queries can be easily integrated into a rendering algorithm. They provide a powerful tool for minimising the implementation effort, especially when compared to CPU-based occlusion culling.

Due to their advantages HOQs have become the preferred occlusion culling method for those scenes spatially arranged with a hierarchical data structure, e.g., [YSGM04, BWPP04, GM05, GBK06].

The main disadvantage presented by HOQs is that there is a *latency* between issuing a query and the availability of the result [BWPP04]. The latency occurs due to the delayed processing of the query in a long graphics pipeline, the cost of processing the query itself and the cost of transferring the result back to the CPU. The latency causes two major problems: CPU stalls and GPU starvation. After issuing the query, the CPU waits for its result and does not feed the GPU with new data. When the result finally becomes available, the GPU pipeline may already be empty; the GPU thus needs to wait for the CPU to process the query result and feed the GPU with new data. A major challenge when using HOQs is to avoid CPU stalls by filling the latency time with other tasks, such as rendering visible scene objects or issuing other, independent occlusion queries.

4.2 Coherent hierarchical culling (CHC)

Some authors have focused their attention on property 1 stated above when dealing with the latency introduced in HOQs (see Section 4.1.1), i.e., the possibility of issuing several occlusion queries in parallel and using their results later [BWPP04, GBK06]. Bittner *et al.,* [BWPP04] have proposed a *coherent hierarchical culling* algorithm (CHC) which exploits the spatial and temporal coherence of visibility inherent in hierarchical representations. The two main ideas introduced in CHC are the following:

1. Assuming coherence of the front of termination nodes:
   a) HOQs are initiated along the *front* of termination nodes (see Section 4.2.2) as determined in the previous frame. This allows saving HOQs on all previously visible interior nodes [BH01].
   b) Previously visible nodes where traversal is determined to be stopped are assumed to remain visible and thus their associated geometry is rendered without waiting for the corresponding HOQ result.
2. Interleaving the queries with the rendering of visible geometry: issued HOQs are stored in a query queue until it is known that they have been carried out by the GPU.

In the particular context of HLOD-based systems, Gobbetti *et al.,* [GM05] adapted the CHC algorithm to perform the traversal of the HLOD hierarchy. The generalities common to both approaches are first described (see Section 4.2.1), followed by the main differences relying on the type of nodes comprising the front of termination nodes (see Section 4.2.2).

4.2.1 CHC traversal algorithm

The CHC algorithm traverses the hierarchy in a top-down (and front-to-back) manner starting at its root node (see Algorithm 1). For each visited node the algorithm proceeds as follows:

- For a previously invisible node, an HOQ is issued and stored in a query queue. The node is delayed for traversal until the HOQ result is available, i.e., the expectation is that the node still remains invisible in the current frame.
- There are two cases for a previously visible node:
  1. If traversal is not terminated at the node (see Section 4.2.2), then its children are scheduled for traversal in a front-to-back order using a priority queue (see the function $traverse()$ in the pseudocode used in Algorithm 1).
  2. If traversal is determined to be stopped at the node (see Section 4.2.2), an HOQ is issued and stored in a query queue. The associated geometry is then rendered immediately without waiting for the HOQ result.

When the query queue is not empty, CHC checks if the result of the oldest query in the queue is already available; if the query result is not available, the traversal continues as described above. If it is available, the result is fetched and the node is removed from the query queue. If it is determined that the node is visible, it is recursively traversed. Otherwise, the whole sub-tree of the node is invisible and thus it is culled. It is worth noting that since the visibility state of the invisible nodes is verified when the HOQ result is available, CHC performs the occlusion culling in a conservative fashion (see Section 2.2).

Visibility classification is pulled up according to the following rule to propagate changes in visibility upwards in the hierarchy: An interior node is invisible only if all its children have been classified invisible, otherwise it remains visible (see the $pullUpVisibility()$ function in the pseudocode).

4.2.2 Front of termination nodes

The set of nodes where traversal algorithm refinement stops because the node has been determined to be invisible or because it becomes rendered forms the *front of termination nodes*. Depending on the type of scene hierarchy, *renderable* nodes correspond to the following:

General scene hierarchies: since only leaf nodes in a non-HLOD scene contain geometry, renderable nodes are only leaf nodes. Therefore, the $evalStopCondition(node)$ function in the pseudocode of Algorithm 1 simply corresponds to querying whether the node is a leaf.

HLOD hierarchies: since every node in the HLOD hierarchy contains geometry, renderable nodes are those where the refinement condition indicates stop. Therefore, the

$evalStopCondition(node)$ function in Algorithm 1 pseudocode simply corresponds to verifying the HLOD stop refinement condition: $SSE(node) \leq \tau$, which is a quick operation [CGG*04] (see Section 3.3).

```
PriorityQueue.Enqueue(hierarchy.Root);
while ¬PriorityQueue.Empty() ∨ ¬QueryQueue.Empty() do
    while ¬QueryQueue.Empty() ∧ (ResultAvailable(QueryQueue.Front()) ∨
    PriorityQueue.Empty()) do
        node←QueryQueue.Dequeue();
        visiblePixels←GetOcclusionQueryResult(node);
        if visiblePixels> 0 then
            PullUpVisibility(node);
            stopTraversal ← evalStopCondition(node);
            Traverse(node, stopTraversal);
    if ¬PriorityQueue.Empty() then
        node←PriorityQueue.Dequeue();
        if InsideViewFrustum(node) then
            wasVisible ← node.visible=true ∧ (node.lastVisited=frameID-1);
            stopTraversal ← evalStopCondition(node);
            node.visible ← false;
            node.lastVisited ← frameID;
            if ¬wasVisible ∨ stopTraversal then
                IssueOclussionQuery(node);
                QueryQueue.Enqueue(node);
            if wasVisible then
                Traverse(node, stopTraversal);

Traverse(node, stop);
if stop then
    Render(node);
else
    PriorityQueue.EnqueueChildren(node);

PullUpVisibility(node);
while ¬node.visible do
    node.visible ← true;
    node ← node.parent;
```
Algorithm 1: Coherent HLOD Culling

5 HLOD refinement driven by HOQs

5.1 Motivation

Although HOQs have recently opened up the possibility for very simple and quite precise visibility computations (see Chapter 4), no HLOD approaches have taken visibility information as an integral part of the refinement condition, to the best of our knowledge. Current HLOD refinement criteria mechanisms (see Section 3.3) cause even occluded regions to be rendered in a high level of detail.

Our aim has been to provide the means for a visibility-based refinement error metric for HLODs to overcome this serious drawback, based on the information gathered from HOQs. We can thus expect increased average rendering frame-rate; once it is determined that there is no gain in further refining a node, due to partial occlusion, the refinement could be stopped in advance, sending less primitives to the graphics card while achieving the same approximated visual quality. Concerning previous work, our aim in improving HLOD performance could be regarded as what *hardly visible sets* are to discrete LODs [ASVNB00] and what *cell solidity values* are to view-dependent LODs [ESSS01] (see Section 2.1.2.4).

Observations

1. Previous HLOD approaches have not taken visibility information into account as an integral part of the HLOD refinement condition: if there is at least one visible pixel and if $(SSE(node) > \tau)$, then the node is further refined [YSGM04, GM05], i.e., HOQs have been used restrictively as if their result were Boolean. The main issue involved is that this approach results in being too conservative, particularly when the number of visible pixels is low [Cha06b, Cha06a].

2. A naive approach to using HOQ results would be to redefine the refinement condition as follows: if $((SSE(node) \leq \tau)$ or $(HOQ(node) \leq \psi))$ then *stop refinement*, where ψ would be an additional user-defined threshold given in pixels. The reason is that the new proposition $(HOQ(node) \leq \psi)$, is view independent, i.e., it does not take into account the viewpoint, nor other viewing parameters and is thus incompatible with the nature of the refinement criterion.

5.2 Challenges

5.2.1 A novel visibility-based refinement error metric

Our error metric should take into account visibility information for more aggressively deciding the stop refinement condition. Due to the underlying occlusion culling method, the visibility information is thus gathered from HOQs. To avoid refinement being stopped at a level where geometric error is still apparent, our new visibility-based error metric should ideally take into account:

1. The *degree* of visibility of the node.
2. The *distribution* of the pixels comprising the regions through which the geometry of the node is reachable. It should be stressed that this feature is not currently supported by HOQs (see Section 4.1.1).

This thesis contributes two approaches for addressing this challenge. Whilst the first approach only takes the degree of visibility of the node into account, the second attempts to exploit the visual masking perceptual phenomenon (see Section 2.1.2.5) by taking into account the possible clustering of the visible pixels comprising the regions through which the geometry of the node is reachable (see Section 6.4). The user-parameters provided in the second approach not only allows it to be reduced into the former, but more importantly, enables the user to fine-tune the tradeoff between speedup and image quality.

5.2.2 A novel HLOD traversal algorithm

Dealing with the latency presented in HOQs and, at the same time, using them as an integral part of the refinement criteria (i.e., within our new visibility-based error metric) results in them being mutually exclusive goals. Because of the latency introduced in HOQs, speedup due to occlusion culling mainly relies on the possibility of issuing several occlusion queries in parallel and using their results later on (see Section 4.1.1). Thus, when HLOD refinement criteria are evaluated, the HOQ result is not readily available (see Algorithm 1).

A new traversal algorithm was thus needed for addressing this shortcoming; it had to be able to predict the visibility-based refinement criterion. This thesis contributes two approaches which exploit the tempo-spatial coherency inherent in hierarchical representations. Whilst the first approach predicts the degree of visibility of the node which is to be used in place of the unknown value when evaluating the HLOD refinement condition (see Section 7.2.1), the second improved approach directly predicts the HLOD refinement condition for the current frame based on the results from the last frame (see Section 7.2.2).

6 Virtual multiresolution screen space errors

By contrast with previous approaches we use HOQ results as an integral part of the HLOD refinement criterion (see Chapter 5). An HOQ issued on an HLOD node bounding volume may be thought as a mask on the screen space that forms *visibility windows* through which the geometry of the node is reachable. An ideal refinement strategy would stop the traversal at the invisible nodes (culling them away) and *selectively* refine the regions within the visible nodes according to the location and size of each visibility window. Due to the visual masking perceptual phenomenon, finer detailed representations should be used only at larger visibility windows while coarser representations may be used at smaller ones (see Section 2.1.2.5). However this method should be aware of the exact position of the visible pixels returned by a given HOQ. Since this functionality is not currently supported by HOQs (see Section 4.1.1), our approach is to use the number of visible pixels returned by the HOQ to modulate the node SSE without performing any selective refinement within visited nodes. However, since the location and size of the visibility windows are disregarded, a loss in image quality is inevitably introduced. It is therefore imperative to keep the loss of image quality within some *reasonable* limits while still managing to more aggressively decide the stop refinement condition.

To begin with, the necessary notions are introduced and our refinement strategy is described (see Sections 6.1 and 6.2, respectively). Our visibility-based HLOD refinement condition is then derived (see 6.3), followed by our approach for minimising the loss of image quality which uses a simple model for reflecting the possible clustering of visible pixels (see Section 6.4).

6.1 Notions and notation

Let M^γ be the subset of screen space that corresponds to the projection of a given node bounding volume B, at the (real or virtual) screen resolution γ, i.e., γ corresponds to the number of pixels in M^γ. Observe that M^γ bounds the projection onto screen space of the MSE related to B, and also the (eventual) projection of its related geometry. Let q be the number of visible pixels on screen space obtained when an HOQ on B is issued, i.e., the HOQ result. Let us say that the *real node resolution* (herein simply referred to as *node resolution*) denoted by α, corresponds to the number of pixels of the projection of B onto screen space, at its real resolution[1] (see Figure 6.1). Finally, we refer to each cluster

[1]Note that calculating the exact node resolution at runtime would be too expensive, e.g., it would require an occlusion query by itself on a reset z-buffer (i.e., if the query is issued when there are no

of visible contiguous pixels found in M^γ as a *visibility window* (or simply *window*), and the number of pixels within it as the *window resolution*.

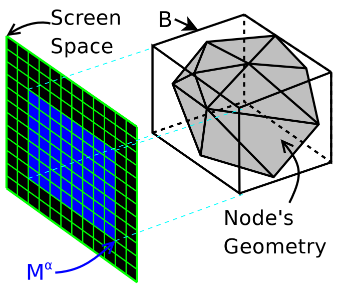

Figure 6.1: Notions. Screen space M^α and node bounding volume B. The total number of pixels comprising the projection of B onto screen space corresponds to α. The returning value of an HOQ issued on B corresponds to q. Since in this example there are no objects between screen space and B, $\alpha = q = 56$.

Table 6.1 gives a summary of the notation used throughout this thesis.

Acronym	*Meaning*	*Reference*
MSE	Model space error	Sections 2.1.2.3 and 3.3
SSE	Screen space error	Sections 2.1.2.3 and 3.3
τ	Pixels of error	Section 3.3
q	HOQ result	This Section
γ	Given real or virtual resolution	This Section
α	Node resolution	This Section
δ	Coarse virtual resolution	Section 6.2
B	Node bounding volume	This Section
M^γ	Projection of B onto screen space at the screen resolution γ	This Section
$VMSSE$	Virtual multiresolution screen space error	Section 6.3
$VMSSE$ *ratio*	$\frac{\delta}{\alpha}$	Section 6.3
μ	Degree of visibility of a node	Section 6.4.1
$\rho_{s,t}(q)$	Sigmoid function used to approximate the $VMSSE$ *ratio*	Section 6.4.1

Table 6.1: Notation used throughout this thesis.

objects between S and B, then $\alpha = q$) or projecting the vertexes of B and then computing the area of the convex hull. However it could efficiently be approximated by simply interpolating the known (e.g., precomputed) node resolutions for the visible faces of B from a given viewpoint and viewing direction at runtime.

6.2 Visibility-based refinement strategy

Our refinement strategy consists of modulating the SSE by taking the HOQ result to *carefully* coarsen the resolution of M^{γ}. That is to say, we interpret a given HOQ result as the virtual resolution (δ) of the screen space where we are going to project a given node data. In all previous approaches the MSE has been directly projected onto screen space at its real resolution (see Section 3.3). The value of the new error metric, dubbed *virtual multiresolution screen space error* ($VMSSE$), is used as the quantity guiding HLOD refinement (see Section 6.3).

As we will see, our visibility-based error metrics produce fairly high quality results regarding image quality. They also possess a useful geometric meaning. Observe that an aggressive coarsening in the node resolution does not necessarily imply a drastic change in the window resolutions. Indeed, it depends on the clustering of visible pixels. The more clustered together the pixels of the windows are, the more likely that a coarsening in the node virtual resolution would produce an important change in the window resolutions. Thus if the node coarser virtual resolution δ ($\delta = f(q) \leq \alpha$) is chosen so that the window resolutions are preserved (see Figure 6.2), then the visual appearance of the geometry should not be substantially affected. However if δ is used instead of α for evaluating the refinement condition, the refinement could be terminated higher in the hierarchy and thus less primitives would be sent to the GPU.

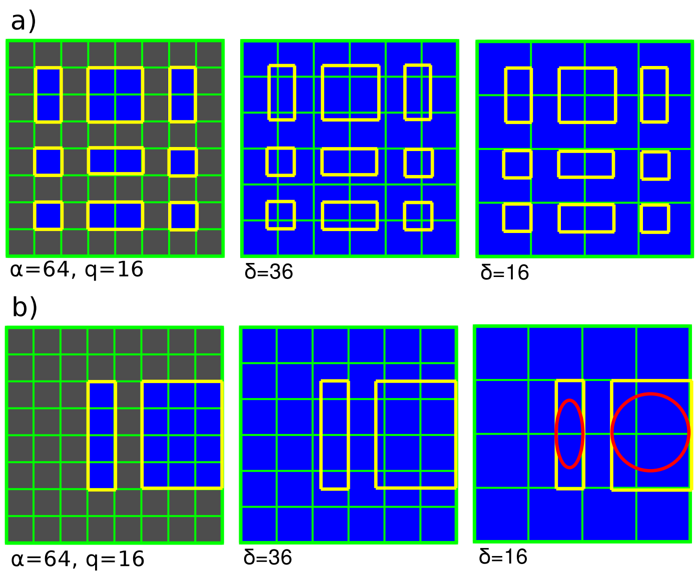

Figure 6.2: Coarsening of a virtual resolution (δ). The HOQ result (q) is the same but the distribution of the visible pixels are different: a) Small size windows (shown in yellow). The window resolutions are kept, even for $\delta = 16$. b) Medium and large size windows (shown in yellow). The window resolutions are kept for $\delta = 36$, but not for $\delta = 16$.

6.3 Visibility-based HLOD refinement condition

Since we can not allow an actual change in the resolution to take place, we now seek to compute the SSE at a virtual resolution δ, from its computation at α. The value of the former can then be taken for effectively refining the hierarchy, without actually modifying the original resolution. For doing so, we show how the SSE could be *virtually* switched among multiple resolutions.

Observe first that if the virtual resolution δ is chosen so that the visual appearance of the geometry is not affected (see Section 6.2), then the appearance of the respective MSE should not be affected either. However, due to the coarsening of the resolution, simply projecting the MSE onto screen space (as described in Sections 2.1.2.3 and 3.3) at the coarse resolution δ leads to a different appearance than when projecting it at the real resolution α (see Figure 6.3). We may thus think of the SSE as the total *intensity* (brightness) of the pixels comprising the MSE projection onto screen space; and keep the absolute intensity constant when switching the resolutions to preserve the appearance (see Figure 6.3).

We now generalise the definition of the screen space error according to a given (real or virtual) screen resolution γ. For doing so, let us say that the *virtual multiresolution screen space error* SSE^γ at a given screen resolution γ, is given by the following expression:

$$SSE^\gamma = \sum_i \sum_j I^\gamma_{i,j},\ \forall i,j \in M^\gamma,$$

where $I^\gamma_{i,j}$ corresponds to the intensity value of the pixel at position (i,j) in M^γ; $I^\gamma_{i,j} \in [0,1]$ if the pixel lays in the projection of the MSE (as described in Sections 2.1.2.3 and 3.3) and $I^\gamma_{i,j} = 0$ if it does not (see Figure 6.3). It should be noted that SSE^γ expresses the total intensity due to SSE, relative to a given screen resolution γ. The convention that if $\gamma = \alpha$ then $I^\alpha_{i,j} = 1$ (for pixels laying on the projection of the MSE) is used for *calibrating* the metric. In this case SSE^γ simply becomes reduced to the original definition of the SSE.

Now let a^γ be the area occupied by a single pixel at a given screen resolution γ. As observed above, the absolute intensity due to SSE^γ should be maintained constant to keep the estimation of the error consistent when *virtually* switching the resolutions. Therefore, since the absolute intensity due to SSE^γ is given by $a^\gamma \sum_i \sum_j I^\gamma_{i,j}$, it follows that (see Figure 6.3):

$$a^\alpha \sum_i \sum_j I^\alpha_{i,j} = a^\delta \sum_{i'} \sum_{j'} I^\delta_{i',j'},\ \forall i,j \in M^\alpha \wedge \forall i',j' \in M^\delta$$

However, since $a^\delta \delta = a^\alpha \alpha$ and $SSE = SSE^\alpha$, we finally get:

$$SSE^\delta = \left(\frac{\delta}{\alpha}\right) \cdot SSE, \tag{6.1}$$

where the expression SSE^δ is herein referred to as $VMSSE$. It should be observed that the ratio $\left(\frac{\delta}{\alpha}\right) \in [0,1]$ modulates the SSE according to the HOQ result, and thus it is herein referred to as the *$VMSSE$ modulation ratio* (or simply *$VMSSE$ ratio*). Note that our new visibility-based HLOD refinement condition may be simply written as: if $(VMSSE(node) \leq \tau)$ then *stop hierarchical refinement*, where the returning value of the function $VMSSE(node)$ corresponds to the $VMSSE$ related to a given *node*.

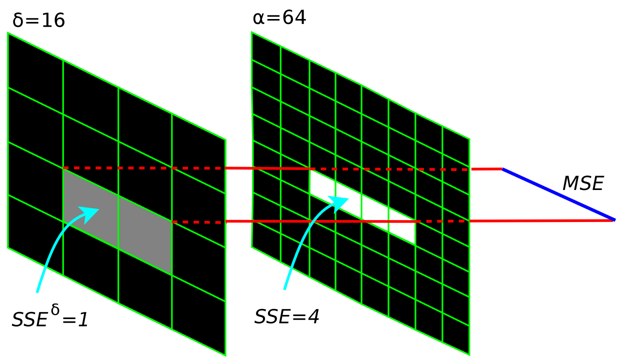

Figure 6.3: Virtual multiresolution screen space errors. The areas covered by the projection of the MSE (which by construction leads to a line segment parallel to the screen x-axis) are different for the two resolutions, but the absolute intensity is the same. In this example $SSE = 4$ and $SSE^{\delta} = 1$ (e.g., $I^{\delta}_{i',j'} = 1/2$, for $(3,2)$ and $(3,3)$).

6.4 Image quality loss minimisation

Observe that the coarser the virtual resolution (δ) is, the more aggressive the stop refinement condition is decided (see Equation 6.1). To take full advantage of the $VMSSE$, δ should therefore be assigned the lowest value leading to no substantial change in window resolutions, otherwise a noticeable loss in image quality may arise (see Section 6.2). However, it would be necessary to determine the exact position of the visible pixels returned by a given HOQ at runtime to precisely compute δ. Since this functionality is not currently supported by HOQs (see Section 4.1.1), an eventual implementation would not only lose simplicity, but would most likely result in being too expensive. Our approach is therefore to approximate the value of δ. It should be noted that a poor approximation may lead to the following drawbacks:

1. When the approximation is too conservative, the benefits of using the $VMSSE$ are missed.
2. When the approximation is too aggressive, a noticeable loss in image quality may arise.

The following simple model was designed to avoid these drawbacks, aiming to reflect the possible clustering of visible pixels.

Note that for extreme visibility conditions, the simple approximation $\delta = q$, effectively minimises the loss in image quality: if $q = 0$ then the value of the $VMSSE$ would be 0 (see **Rule 1** below) and if $q = \alpha$ then the value of the $VMSSE$ would be the same as if no visibility information would have been gathered, i.e., $VMSSE = SSE$ (see **Rule 2** below). The above image quality loss minimisation problem could thus be stated in the following much simpler terms:

> *We seek the curve $\delta = \rho(q)$ that interpolates the points $(0, 0)$ and (α, α), and minimises the loss in image quality while still allowing an aggressive modulation of the SSE.*

It should finally be observed that the less scatter the visible pixels of the HOQ are, the higher the chance that the pixels of the windows would be more clustered together, and thus it would be more likely that a coarsening in the virtual resolution (δ) would produce an important change in the window resolutions (see Section 6.2, and **Rules 3** and **4** below). The shape of $\rho(q)$ can thus be modelled according to the following set of rules:

Rule 1: If $q = 0$ then $\delta = q$, i.e., the value of the $VMSSE$ should be 0 (the object is invisible and thus refinement should be stopped). See Figure 6.4.

Rule 2: If $q = \alpha$ then $\delta = q$, i.e., the value of the $VMSSE$ should be the same as if no visibility information had been gathered, $VMSSE = SSE$. See Figure 6.4.

Rule 3: If the value of q is close to 0 then $\delta \to 0$, i.e., the visible pixels of the HOQ are scatter and thus we can aggressively modulate the SSE. This rule defines the cyan region in Figure 6.4.

Rule 4: If the value of q is close to α then $\delta \to \alpha$, i.e., the visible pixels of the HOQ are more clustered together and thus in order to minimise the image quality loss we should conservatively modulate the SSE. This rule defines the blue region in Figure 6.4.

It is worth noting that **Rules 3** and **4** express an attempt to exploit the visual masking perceptual phenomenon to deciding the stop refinement condition (see Section 2.1.2.5).

6.4.1 *VMSSE ratio* sigmoid approximation

We simply use a sigmoid function to conveniently capture the set of rules stated above: $\delta = \rho_{s,t}(q)$. Two user defined parameters are used to provide more control over the shape of the sigmoid curve: 1. A *movable turn over* point t, $t \in [0, 1]$ of the sigmoid function, i.e, the closer t is to 0, the more conservative the approximation is (see Figure 6.4); and, 2. A *smoothness* parameter s, $s \in [0, 1]$ that linearly interpolates between $\delta = q$ and the sigmoid, i.e., the closer s is to 0, the smoother the shape of δ is (see Figure 6.4) [CBWR07]. The two parameters, s and t (which typically takes small values), are determined experimentally by the user according to the underlying scene (see Chapter 8).

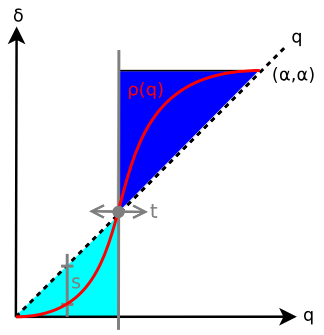

Figure 6.4: Sigmoid function $\delta = \rho_{s,t}(q)$ used to minimise the loss in image quality due to *VMSSE*. In the cyan region ($q \leq t$) the function provides aggressive modulation of the *SSE* compared to linear scaling, while in the blue region ($q > t$) the modulation is rather conservative. The s value is used to control the smoothness of the curve. Note that taking $s = 0$ leads to linear scaling.

For convenience, the sigmoid function may be normalised using the node resolution (α), i.e., $P_{s,t}(\mu) = \rho_{s,t}(q)/\alpha$, where $\mu = (\frac{q}{\alpha}) \in [0,1]$ corresponds to the degree of visibility of the node (simply computed as the degree of visibility of the node bounding volume). Since $\delta = \rho_{s,t}(q)$, then the normalised sigmoid function $P_{s,t}(\mu)$ simply corresponds to the *VMSSE ratio* (see Equation 6.1).

6.4.2 *VMSSE ratio* linear approximation

It is also possible to directly take the number of visible pixels returned by an HOQ as the number that defines the node virtual resolution used to evaluate the refinement condition, i.e., $\delta = q$ [Cha06b]. Observe that this model only takes **Rules 1** and **2** stated above into account and that it ignores the visual masking phenomenon (**Rules 3** and **4**). Also note that this model could be reduced from the sigmoid approximation stated above by simply taking user-defined parameter $s = 0$, i.e., $P_{s=0,t}(\mu) = \mu$ (see Figure 6.4).

It should also be stressed that the user parameters s and t provide the means for fine-tuning the tradeoff between speedup and image quality according to user needs. Indeed, it has been found experimentally that the highest speedup has typically been obtained when using the linear *VMSSE ratio* function $P_{s=0,t}(\mu) = \mu$, i.e., $\delta = q$ (see Chapter 8).

7 Visibility-based HLOD culling algorithm

Dealing with the latency presented in HOQs and, at the same time, using HOQs to assist the refinement mechanism proves cumbersome; the reason is that there is a mutual dependence between them. On the one hand, the culling algorithm should issue several HOQs and proceed with HLOD refinement without waiting for their results (see Section 4.2). On the other hand, the query result becomes necessary for determining the stop refinement condition (see Chapter 6). The HLOD refinement condition must therefore be predicted to break such mutual dependence.

Before showing our two proposed approaches for predicting the HLOD refinement condition [Cha06a, CBWR07], the traversal algorithm which is common to both approaches is first described (see Section 7.1). A detailed discussion is then given of our two proposed methods for predicting the HLOD refinement condition by using temporal coherence of visibility (see Section 7.2).

7.1 Visibility-based HLOD traversal

The aim of our new algorithm is to perform efficient traversal of the HLOD hierarchy while using visibility information to drive HLOD refinement (see Chapter 5). A naive algorithm would issue an occlusion query for every traversed node, wait for its result, compute the refinement condition and decide whether to descend the hierarchy. Waiting for the query result is expensive as it stalls the CPU and in turn causes GPU starvation (see Section 4.1.2). Our algorithm solves this problem by predicting the refinement criterion using temporal coherence of visibility. When proceeding from one frame to the next, it is most likely that refinement will stop at the same set of nodes where it has stopped in the previous frame (a fact fully exploited by the technique described in Section 7.2.2). Exceptions occur when refinement is shifted up or down from the current level of the node due to a change in its $VMSSE$.

Our coherent HLOD culling algorithm proceeds as follows; the traversal of the HLOD hierarchy is started at the root node. At each traversed node, the refinement condition for the node is predicted using some form of temporal coherence (the prediction will be described in detail in the next Section). The prediction indicates one of the following actions: (1) *refine*, (2) *stop* refinement or (3) *delay* the decision to the moment when the visibility of the node for the current frame becomes known.

In case (1), the children of the node are traversed by putting them on the priority queue. In case (2) and (3), an HOQ is issued for the node and put in the query queue. In case (2), the geometry associated with the node is immediately rendered and the refinement is stopped. In case (3), the processing of the node is delayed until the HOQ result becomes available in the query queue. The decision is then made using the updated information about the visibility of the node: if the node is invisible, it is culled. If the $VMSSE$ is lower than the threshold, refinement stops and the geometry of the node is rendered. Otherwise, the refinement continues by inserting the children of the node into the priority queue. Before showing our two approaches for predicting the HLOD refinement condition, let us first analyse the consequences of these actions for the traversal algorithm:

1. *Refine.* The children of the node are traversed immediately. No HOQ or rendering is performed for the current node. If it turns out that the prediction was *too conservative* (it actually might have stopped for the current node) then more geometry than necessary is rendered.
2. *Stop.* An HOQ is issued and the node is rendered immediately. When the query result is available, the $VMSSE$ of the current node is computed. The children of the node must continue to be traversed if the prediction was *too aggressive.* Note, in this case, that the geometry of (some) child nodes is rendered over the geometry of the parent node, thereby increasing the rendering cost and very possibly leading to visual artifacts.
3. *Delay.* In this case, the query result is awaited for deciding on the refinement condition. Thus, for a node which was delayed and for which refinement should have stopped, a latency is introduced in passing its geometry to the GPU.

The new traversal algorithm is outlined in Algorithm 2. Note that differences in relation to the traversal algorithm of Gobbetti *et al.*, [GM05] this being the closest algorithm published to date regarding our approach, have been coloured (see Section 4.2.1). The non-coloured parts of the algorithm are also virtually identical to the basic CHC algorithm produced by Bittner *et al.*, [BWPP04] and are therefore not explained in detail here (see Section 4.2.1). It is worth noting that since the visibility state of nodes predicted as being invisible is verified when the HOQ result is available, then this strategy leads to conservative occlusion culling (see Section 2.2).

7.2 Predicting the HLOD refinement condition

We designed two different techniques which aim at minimising the number of incorrect predictions (see Sections 7.2.1 and 7.2.2). Both techniques estimate the $VMSSE$ ($\widehat{VMSSE}$) by combining the SSE of the current frame with the estimated value of the $VMSSE$ *ratio* ($\widehat{ratio}$):

$$\widehat{VMSSE} = \widehat{ratio} \cdot SSE \tag{7.1}$$

The $\widehat{VMSSE}$ is used when evaluating the visibility-based stop refinement condition during the traversal. The main difference between the two predictors lays in how they compute $\widehat{ratio}$.

```
PriorityQueue.Enqueue(hierarchy.Root);
while ¬PriorityQueue.Empty() ∨ ¬QueryQueue.Empty() do
    while ¬QueryQueue.Empty() ∧ (ResultAvailable(QueryQueue.Front()) ∨
    PriorityQueue.Empty()) do
        node←QueryQueue.Dequeue();
        visiblePixels←GetOcclusionQueryResult(node);
        if visiblePixels> 0 then
            PullUpVisibility(node);
            $\mu \leftarrow visiblePixels/BBoxPixels(node)$;
            VMSSE ← $P_{s,t}(\mu)*SSE(node)$;
            stopTraversal ← VMSSE ≤ $\tau$;
            node.ratio ← $P_{s,t}(\mu)$ ; // only for the approach in Section 7.2.2
            node.stopRefinement ← stopTraversal ; // only for Section 7.2.2
            Traverse(node, stopTraversal);
    if ¬PriorityQueue.Empty() then
        node←PriorityQueue.Dequeue();
        if InsideViewFrustum(node) then
            stopMode←PredictRefinement(node);
            stopTraversal ← ¬(stopMode=Refine);
            node.stopRefinement ← stopTraversal ; // only for Section 7.2.2
            node.visible ← false;
            node.lastVisited ← frameID;
            if stopTraversal then
                IssueOclussionQuery(node);
                QueryQueue.Enqueue(node);
            if ¬(stopMode=Delay) then
                Traverse(node, stopTraversal);
```

Algorithm 2: Coherent Visibility-based HLOD Culling

7.2.1 Approximation of the degree of visibility of the node

A simple approach to compute $\widehat{ratio}$ is to approximate the degree of visibility of the node ($\widehat{\mu}$) [Cha06a].

$$\widehat{ratio} = P_{s,t}(\widehat{\mu}) \tag{7.2}$$

The main observation for predicting $\widehat{\mu}$ is to use the position of the node in the priority queue as determined in the previous frame. The reason is that the nodes in the priority queue are scheduled in an approximated front-to-back traversal order [BWPP04] (see Section 4.2.1). Thus, the assumption in computing $\widehat{\mu}$ is that it linearly decreases from 1 to 0 with the position of the node in the priority queue (see Algorithm 3).

The main limitation of this technique is that it is not precise enough (see Chapter 8),

```
if ¬(node.lastVisited=frameID-1) ∨ node.visible=false then
    return Delay;
$\widehat{\mu} = 1 - (node.insertionPosition/totalScheduledNodes(frameID - 1))$;
$\widehat{VMSSE} \leftarrow P_{s,t}(\widehat{\mu}) * SSE(node)$;
if $\widehat{VMSSE} \leq \tau$ then
    return Stop;
return Refine;
```

Algorithm 3: PredictRefinement(node)

leading to two main drawbacks: (1) when the prediction is too conservative, more nodes become refined and in turn more primitives are rendered; and (2) when the prediction is too aggressive, the node is rendered, but the same node is then also refined when the query result is available. This means that the refined children are rendered together with their parent, which can cause visual artifacts (see Section 7.1 and 8.3). This is because this technique does not take the underlying scene depth complexity into account; this clearly plays a role when estimating the degree of visibility of a node respecting its scheduled traversal position.

7.2.2 Coherency of the front of termination nodes

From the above analysis of the actions needed to be carried out by the predictor (see Section 7.1) and to overcome the lack of precision of the previous method (see Section 7.2.1), we designed a prediction technique aiming at minimising the number of incorrect predictions by assuming coherence of the front of termination nodes. It was primarily aimed at predicting either *refine* or *stop* conditions with a high degree of accuracy. If a stop condition is expected, but with lower confidence, then the predictor returns *delay.* It also returns *delay* for nodes which have been previously invisible and thus it is expected that the refinement will terminate without rendering the geometry of these nodes. The main idea of the prediction is to estimate the $VMSSE$ *ratio* ($\widehat{ratio}$) using the cached *ratio* as determined in the previous frame:

$$\widehat{ratio}_i = ratio_{i-1} \tag{7.3}$$

However, it should be noted that the actual *ratio* is only computed for nodes where refinement stopped in the previous frame. Therefore, to check the availability of the cached *ratio* at the current frame, the result of the stop refinement condition is also cached at the node. The prediction works as follows (see also the pseudocode depicted in Algorithm 4), if:

- *the node was invisible in the previous frame*, then the prediction returns *delay.*
- *the refinement was stopped for the node in the previous frame*, then the $\widehat{VMSSE}$ is calculated and if it is still below the threshold, the predictor returns *stop.* Otherwise, a significant change in the node visibility has occurred and the predictor returns *refine.*

- *the node was refined in the previous frame, but refinement stopped for all its child nodes*, then the node is a good candidate for pulling up the termination front (see the red node in Figure 7.1). This is verified by first checking whether refinement for all children would still stop in the current frame based on their $\widehat{VMSSE}$ estimations. If any of these indicates continue refinement, then the predictor returns *refine*. Otherwise, since the node itself does not have a cached *ratio* value, the node $\widehat{ratio}_i$ is approximated by taking the average cached *ratios* from all children. If the resulting $\widehat{VMSSE}$ is above the threshold, the predictor returns *refine*. Otherwise the predictor returns *delay*.

```
if ¬(node.lastVisited=frameID-1) ∨ node.visible=false then
    return Delay;
if ¬node.stopRefinement then
    candidateToShiftUp ← true;
    forall child ∈ node.children do
        if ¬child.stopRefinement ∨ ¬(child.ratio*SSE(child) ≤ τ) then
            candidateToShiftUp ← false;
    if candidateToShiftUp then
        if AvgRatio(node.children)*SSE(node) ≤ τ then
            return Delay;
else if node.ratio*SSE(node) ≤ τ then
    return Stop;
return Refine;
```

Algorithm 4: PredictRefinement(node)

The candidates for updating the front of termination used by the predictor described above are depicted in Figure 7.1.

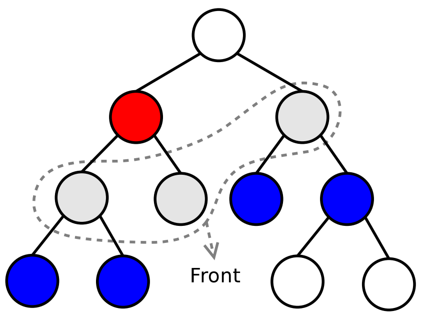

Figure 7.1: Set of nodes which are candidates for being in the front in the next frame: (1) the front of termination nodes in the current frame, (2) nodes one level above this set (only if refinement has stopped for all its children), coloured as red and, (3) nodes one level below this set, coloured blue.

Compared to the first approach [Cha06a] (see Section 7.2.1), it has been found that this technique better predicts the HLOD stop refinement condition [CBWR07]. Independent of scene depth complexity, using this method has produced nearly the optimal number of nodes to be drawn (see Chapter 8). Moreover, it also preserves the visual quality obtained in the final rendered image, i.e., the only source which could hinder the visual quality is when there is an important number of nodes needing to be refined once they have been rendered in the same frame (see Section 7.1). However, the accuracy of the prediction and the verification of the stop refinement condition described above practically eliminate this problem (see Section 8.3).

8 Results and discussion

We implemented an experimental software supporting our coherent HLOD culling algorithm using C++ and OpenGL in Linux. The HLODs use an octree with a single discrete LOD per node consisting of about 2000 triangles. The quadric error metric was used for constructing the HLODs and for deriving model space errors [CGG*04, YSGM04] (see Sections 2.1.2.3 and 3.3). Axis-aligned minimum bounding boxes around the geometry of the nodes were precomputed offline and were then used for issuing HOQs and performing $VMSSE$ computations at runtime (see Chapters 4 and 6, respectively). Vertex buffer objects were used for efficient caching of the geometry on the GPU (see Section 3.2). The measurements were performed on three scenes having different depth complexity. A $640 * 480$ pixel resolution window was used for all tests and error threshold $\tau = 1$. The tests were evaluated on a PC with Intel-Core 2 Duo (2.4GHz) and nVidia GeForce 8800 GTX.

8.1 Tests

Three scenes having low, middle and high depth complexity were used, respectively called *scene 1*, *scene 2* and *scene 3* (see Figures 8.1, 8.2 and 8.3). A session representing typical inspection tasks was designed for each scene. Our inspection sequences include rotations and changes from overall views to extreme close-ups, heavily stressing the system. The sequence was played for each scene to collect the frame rates and number of drawn nodes. The following scenarios were evaluated depending on the traversal algorithm and the error metrics used to refine the hierarchy:

1. *Bool*: the hierarchy was traversed using the coherent culling algorithm version of Gobbetti *et al.*, [GM05] (see Section 4.2.1). The error metric used for refining the hierarchy was SSE, i.e., HOQs were used as if their result were Boolean. Note that this configuration yielded the *ideal* image quality given as reference in our tests below (see Section 8.2).

2. The following two scenarios were set up to evaluate the tradeoff between image quality and speedup due to the combination of both, the error metric and the traversal algorithm introduced in this thesis:

   a) $Coh(P)$: the hierarchy was traversed with our coherent HLOD culling algorithm (see Section 7.2.2) using $VMSSE$ to refine the hierarchy and $P_{s,t}(\mu)$ to compute the $VMSSE$ *ratio* (see Section 6.4.1). Note that this configuration sought to obtain high quality image results. User parameters s and t were determined experimentally with this purpose in mind (see Figures 8.1, 8.2 and 8.3).

b) $Coh(\mu)$: the hierarchy was traversed with our coherent HLOD culling algorithm (see Section 7.2.2) using μ to compute the *VMSSE ratio*, i.e., $P_{s=0,t}(\mu) = \mu$ (see Section 6.4.2). Note that this configuration sought to obtain the highest possible speedup.

8.2 Image quality

The differences were measured regarding the final ideal image obtained when using *Bool* and that obtained when using $Coh(P)$ and $Coh(\mu)$. 20 frames from the inspection sequence were thus randomly selected and the peak signal-to-noise ratio (*psnr*) difference calculated. This measurement has been traditionally used as an estimator of the distortion introduced by compression algorithms [Net89, TM02] and corresponds to the ratio of the power of a particular signal and the corrupting noise. The average *psnr* values for the 20 frames are presented in Table 8.1. Note that:

1. Using the two computations proposed for the *VMSSE ratio* (see Section 6.4) did not substantially alter final image quality as objectively measured in these frames from the inspection sequences. The fact that a *psnr* > 30 was returned for all colour components indicated that the two proposed methods practically did not alter final image quality [GW01].
2. Using $Coh(P)$ gave better image quality than $Coh(\mu)$ [GW01]. The reason was that the latter approximation only took the degree of visibility of the node into account whilst the former also considered the visual masking perceptual phenomenon.

Eval.	*Bool vs Coh(P)*			*Bool vs Coh(μ)*		
dB	*Luminance*	C_b	C_r	*Luminance*	C_b	C_r
S1	48.5 ± 2.9	70.8 ± 3.7	60.3 ± 3.6	46.4 ± 2.6	68.1 ± 2.6	57.89 ± 3
S2	41.1 ± 2.6	64.2 ± 2.7	53.9 ± 2.9	37.4 ± 1.6	60.7 ± 2.2	49.9 ± 2.5
S3	34.4 ± 1.5	59.9 ± 2	50.4 ± 2.6	30.1 ± 1.5	55.6 ± 1.7	45.6 ± 1.6

Table 8.1: Average and standard deviation *psnr* values, i.e., luminance and chrominance (C_b and C_r) for 20 image pairs from the tests scenes randomly selected from the inspection sequence. Each pair comprises the ideal image obtained when using *Bool* and that obtained when using: 1. $Coh(P)$; and, 2. $Coh(\mu)$.

In addition to *psnr* measurements, the geometry has been coloured from blue to magenta to red according to the severity level of the modulation introduced by the *VMSSE* to emphasise the influence on image quality caused by $Coh(\mu)$ and $Coh(P)$ respecting *Bool* for each node in the front: blue represents regions of the model where the modulation is weak, magenta represents regions where the modulation is moderate and red represents regions where the modulation is strong (see the last two rows in Figures 8.1, 8.2 and 8.3). Note that:

1. Using $Coh(\mu)$ and $Coh(P)$ attenuated the loss in image quality, i.e., the stronger the modulation was, the less likely it was that the node was actually visible.

2. As noticed above, using $Coh(P)$ better attenuated the loss of image quality than $Coh(\mu)$, i.e., SSE modulation due to the $VMSSE$ *ratio* in $Coh(P)$ was determined according to the visual masking perceptual phenomenon (see Section 6.4.1). The reader may compare the magenta and red areas present in $Coh(P)$ and in $Coh(\mu)$ from the user's viewpoint (second row in Figures 8.1, 8.2 and 8.3).

8.3 Speedup

The following two scenarios were also set up for evaluating speedup due to $Coh(P)$ and $Coh(\mu)$, not only respecting *Bool* but also respecting our alternative (former) approach to predicting the HLOD refinement condition as depicted in Section 7.2.1:

1. $Simp(P)$: the hierarchy was traversed with our former coherent HLOD culling algorithm (see Section 7.2.1) using $P_{s,t}(\mu)$ to compute the $VMSSE$ *ratio* (see Section 6.4.1).

2. $Simp(\mu)$: the hierarchy was traversed with our former coherent HLOD culling algorithm (see Section 7.2.1) using μ to compute the $VMSSE$ *ratio*, i.e., $P_{s=0,t}(\mu) = \mu$ (see Section 6.4.2).

The following two scenarios were also set up to determine the ideal number of nodes to be drawn in $Coh(P)$ as well as in $Coh(\mu)$:

1. $SW(P)$: the hierarchy was traversed with the hierarchical *stop-and-wait* method referred to in Bittner *et al.*, [BWPP04] using $P_{s,t}(\mu)$ to compute the $VMSSE$ *ratio* (see Section 6.4.1).

2. $SW(\mu)$: the hierarchy was traversed with the hierarchical *stop-and-wait* method referred to in Bittner *et al.*, [BWPP04] using μ to compute the $VMSSE$ *ratio* (see Section 6.4.2).

Figures 8.4 and 8.5 show the whole sequence of drawn nodes together with the frame rates for the three scenes. All scene statistics have been summarised in Table 8.2. Table 8.3 shows the relative speedup and node savings for $Coh(P)$ respecting *Bool* and $Simp(P)$. Table 8.4 shows the relative speedup and node savings regarding $Coh(\mu)$ respecting to *Bool* and $Simp(\mu)$.

It can be seen that $Coh(P)$ significantly reduced the number of drawn nodes compared to both *Bool* and $Simp(P)$, also directly translating into higher frame rates. The main

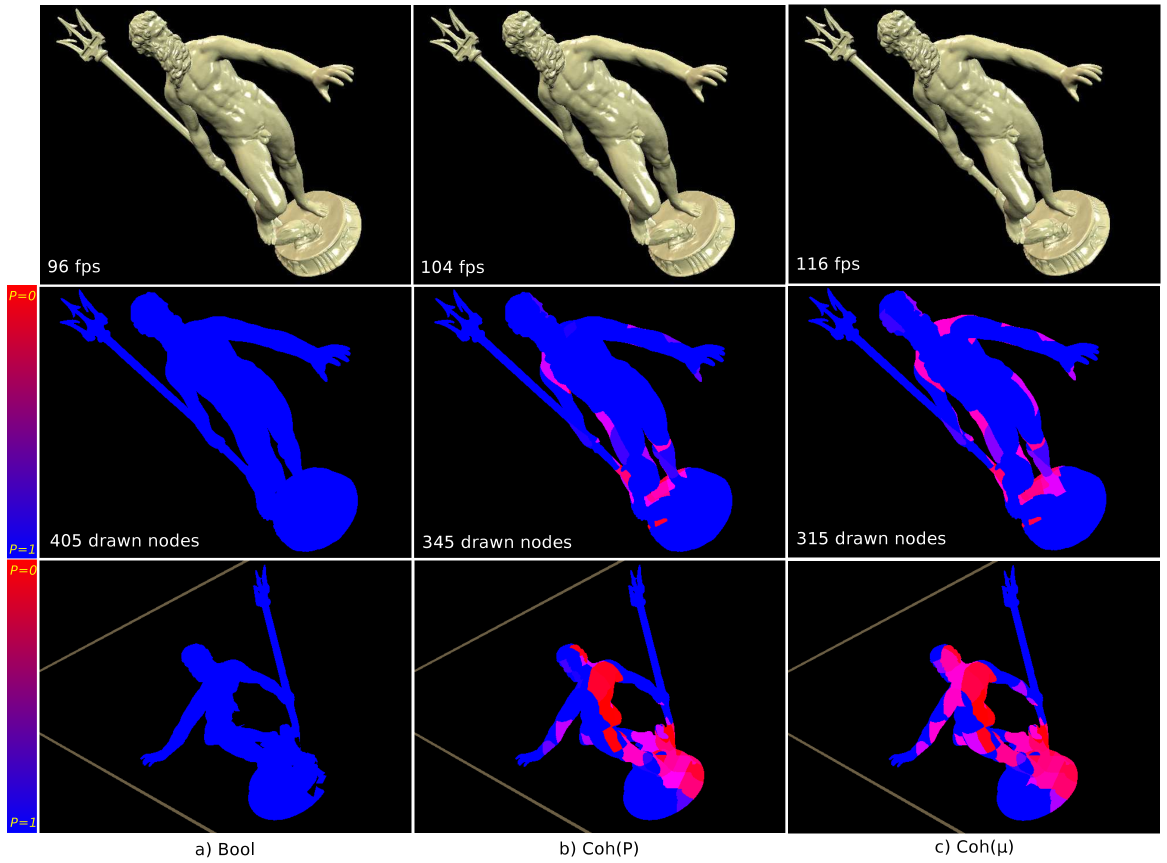

Figure 8.1: Scene 1: selected frame of the visualisation sequence when using *Bool*, *Coh*(P) and *Coh*(μ). The last two rows correspond to a visualisation (from the user's viewpoint and a bird's eye view) of the introduced modulation of the nodes selected to be drawn due to the *VMSSE ratio* $(P_{s=0.6,t=0.2}(\mu))$: blue represents weak modulation (small reduction of LOD level, i.e., $P_{s,t}(\mu) \simeq 0$), magenta represents moderate modulation and red represents high modulation (strong reduction of LOD level, i.e., $P_{s,t}(\mu) \simeq 1$). Model courtesy of the AIM@SHAPE Repository [AS].

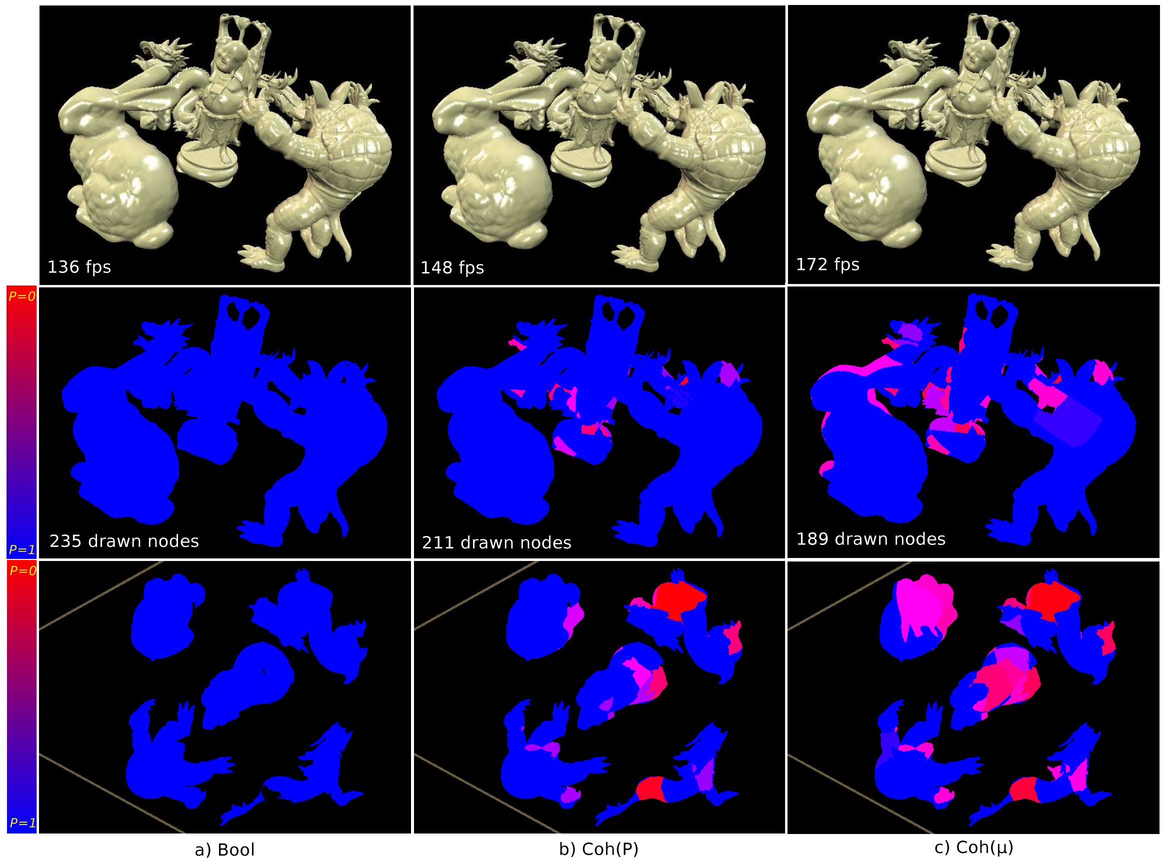

Figure 8.2: Scene 2: selected frame of the visualisation sequence when using *Bool*, $Coh(P)$ and $Coh(\mu)$. The last two rows correspond to a visualisation (from the user's viewpoint and a bird's eye view) of the introduced modulation of the nodes selected to be drawn due to the *VMSSE ratio* ($P_{s=0.7,t=0.15}(\mu)$): blue represents weak modulation (small reduction of LOD level, i.e., $P_{s,t}(\mu) \simeq 0$), magenta represents moderate modulation and red represents high modulation (strong reduction of LOD level, i.e., $P_{s,t}(\mu) \simeq 1$). Model courtesy of Stanford Graphics Group [SU].

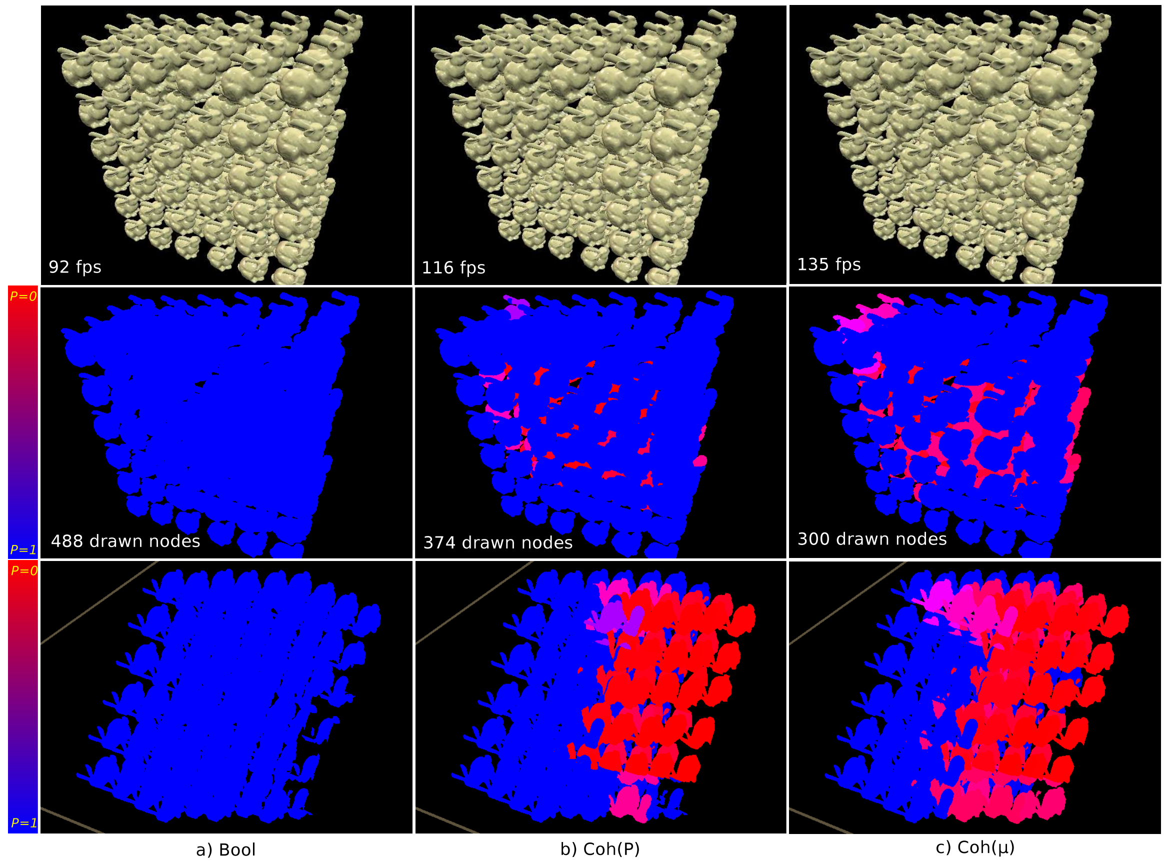

Figure 8.3: Scene 3: selected frame of the visualisation sequence when using *Bool*, *Coh*(P) and *Coh*(μ). The last two rows correspond to a visualisation (from the user's viewpoint and a bird's eye view) of the introduced modulation of the nodes selected to be drawn due to the *VMSSE ratio* $(P_{s=0.9,t=0.05}(\mu))$: blue represents weak modulation (small reduction of LOD level, i.e., $P_{s,t}(\mu) \simeq 0$), magenta represents moderate modulation and red represents high modulation (strong reduction of LOD level, i.e., $P_{s,t}(\mu) \simeq 1$). Model courtesy of Stanford Graphics Group [SU].

Statistics	*scene 1* *full resolution model* $\approx$ *4M* $\triangle'$*s* *number of HLOD nodes* $\approx$ *10k*				*scene 2* *full resolution model* $\approx$ *5M* $\triangle'$*s* *number of HLOD nodes* $\approx$ *12k*				*scene 3* *full resolution model* $\approx$ *12M* $\triangle'$*s* *number of HLOD nodes* $\approx$ *21k*			
Scenario	FPS	DN	RARN	D	FPS	DN	RARN	D	FPS	DN	RARN	D
Bool	153.4	291.2	–	–	173.9	217	–	–	101	457.4	–	–
$Coh(P)$	179.8	223.2	0.1	6	194.5	183.7	0.1	5	135.9	312.2	0.3	4.5
$Simp(P)$	164.8	251.5	9.7	–	179.3	199.9	5	–	109.3	381.7	3.9	–
$SW(P)$	67.7	220.1	–	–	47.8	181.4	–	–	66.6	310.7	–	–
$Coh(\mu)$	201.1	194.8	0.2	7.3	211.2	166.8	0.2	5.3	174.3	235.2	0.2	6.9
$Simp(\mu)$	181.2	220.6	9.5	-	195.6	183.8	5.3	–	138.6	300.9	7.3	–
$SW(\mu)$	76.4	192.7	–	–	50.8	165	–	–	90.8	233.8	–	–

Table 8.2: Statistics for the test scenes. FPS is the number of frames per second, DN is the number of drawn nodes, RARN is the number of nodes that once rendered in a given frame need further refinement within the same frame (only for $Coh(P)$, $Simp(P)$, $Coh(\mu)$ and $Simp(\mu)$) and D is the number of nodes delayed for rendering (only for $Coh(P)$ and $Coh(\mu)$). All values are averages over all frames.

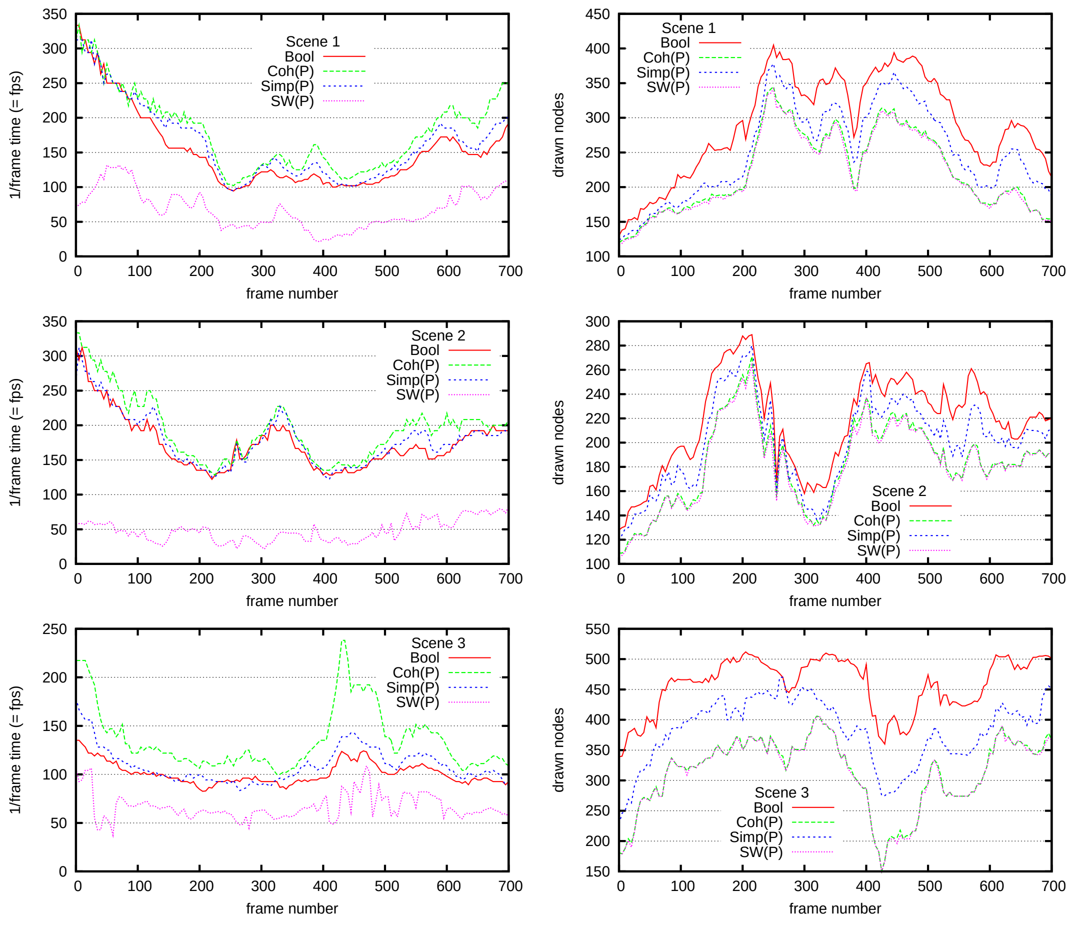

Figure 8.4: Drawn nodes and frame rates for the test scenes using *bool*, $Coh(P)$, $Simp(P)$ and $SW(P)$.

reason for the node savings obtained in $Coh(P)$ respecting *Bool* was the use of $VMSSE$ instead of SSE within refinement conditions (image quality results have been previously discussed in Section 8.2). The benefits obtained from using $VMSSE$ are more clearly perceived in scenes having higher depth complexities where the number of node savings is more important. The reason for the node savings obtained in $Coh(P)$ respecting $Simp(P)$ was the tighter approximation it gave for the ideal number of nodes to be drawn, relying on the method for approximating $VMSSE$ *ratio*. Comparison with $SW(P)$, which does not use prediction, showed that $Coh(P)$ gave more than 98% precision for this number (see Table 8.3). It is also worth noting that $Simp(P)$ precision depends on scene depth complexity; it behaves poorly in scenes having higher depth complexities. A similar analysis revealed comparable results for $Coh(\mu)$ respecting $Simp(\mu)$. However, $Coh(\mu)$ gave even higher speedup over *Bool* than $Coh(P)$, as expected.

The only source for visual artifacts inherent in the traversal algorithm (as opposed to the $VMSSE$ calculation) is the case when there is an important number of nodes

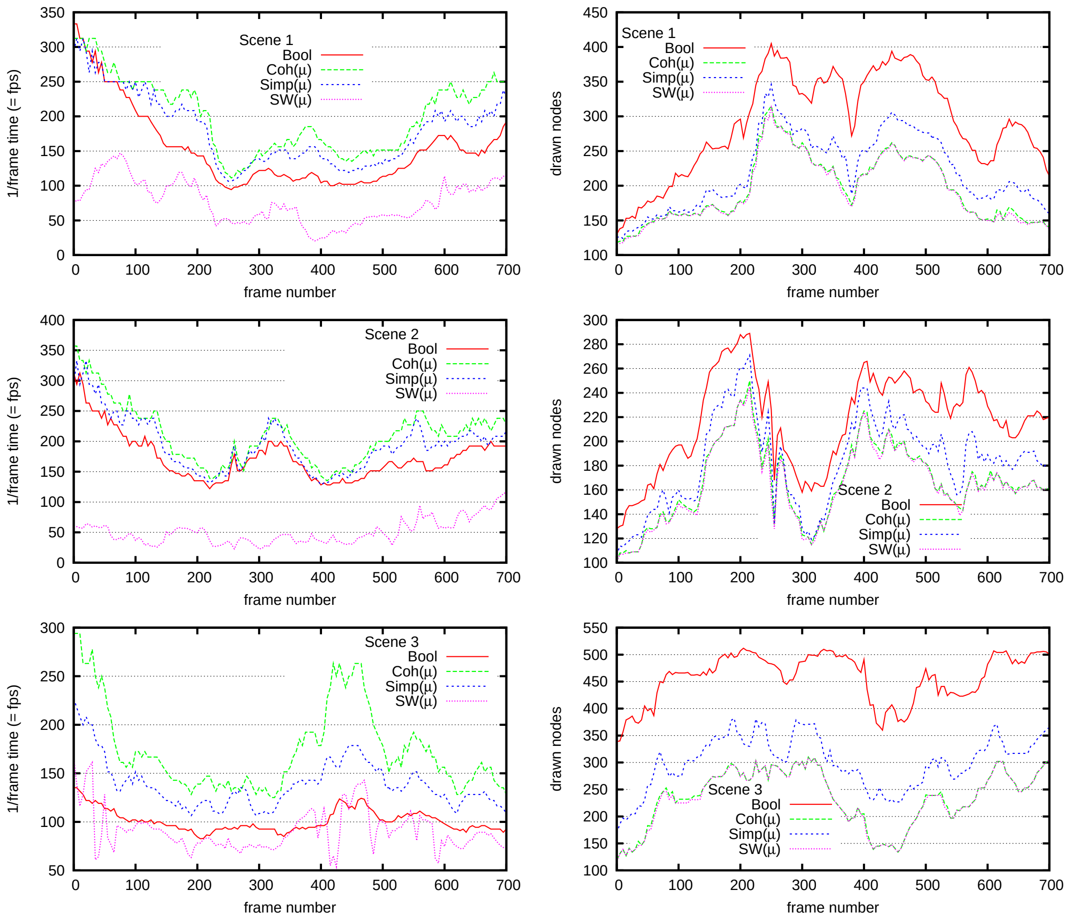

Figure 8.5: Drawn nodes and frame rates for the test scenes using *bool*, $Coh(\mu)$, $Simp(\mu)$ and $SW(\mu)$.

(RARN) which need to be refined, even though they have already been rendered in the same frame (see Section 7.1). Figures 8.6 and 8.7 have followed the same colouring scheme described in the previous section to reveal this source for the appearance of visual artifacts. It was always found that the value of RARN was negligible, unlike $Simp(P)$ and $Simp(\mu)$, in $Coh(P)$ and $Coh(\mu)$ (see Table 8.2). The reason was that our delay strategy for the nodes where the stop refinement condition was predicted to be shifted up (see Section 7.2.2) effectively minimised RARN without hindering performance; the average number of nodes which were delayed for rendering in $Coh(P)$ out of the total number of drawn nodes for the three scenes were only 2.7%, 2.7% and 1.4%, respectively, and the correspondent values in $Coh(\mu)$ for the three scenes were 3.8%, 3.2% and 3.1%, respectively.

Statistics	*Speedup*			*Node Savings*		
Scenario	*scene1*	*scene2*	*scene3*	*scene1*	*scene2*	*scene3*
Bool	17.2	11.8	34.6	23.4	15.3	31.7
$Simp(P)$	9.1	8.5	24.4	11.2	8.1	18.2
$SW(P)$	165.3	306.5	104.2	−1.4	−1.3	−0.5

Table 8.3: Relative average speedup and node savings statistics for the test scenes found in $Coh(P)$ respect to *Bool*, $Simp(P)$ and $SW(P)$. All values are percentages.

Statistics	*Speedup*			*Node Savings*		
Scenario	*scene1*	*scene2*	*scene3*	*scene1*	*scene2*	*scene3*
Bool	31.1	21.5	72.6	33.1	23.1	48.6
$Simp(\mu)$	11	8	25.8	11.7	9.2	21.8
$SW(\mu)$	163.2	316.1	91.9	−1.1	−1.1	−0.6

Table 8.4: Relative average speedup and node savings statistics for the test scenes found in $Coh(\mu)$ respect to *Bool*, $Simp(\mu)$ and $SW(\mu)$. All values are percentages.

8.4 Summary of results

The results showed that our approach ($Coh(P)$ and $Coh(\mu)$) was superior respect to previous state-of-the-art methods regarding frame-rate while keeping good visual quality. Compared to the method of Gobbetti *et al.*, [GM05] (*Bool*), the reference solution for image quality measurements, $Coh(P)$ led to obtaining 17,2%, 11.8% and 34.6% of speedup for the three scenes, respectively, while $Coh(\mu)$ led to obtaining 31.1%, 21.5% and 72.6% of speedup for the three scenes, respectively. Both speedups were significant, while the visual quality of our method did not incur a perceivable penalty, especially for $Coh(P)$ which gave better image quality results than $Coh(\mu)$ (average *psnr* values for the 20 frames are presented in Table 8.1).

The following results were obtained by comparison with our former traversal algorithm ($Simp(P)$ and $Simp(\mu)$). In comparison to $Simp(P)$, the speedup for the three scenes in $Coh(P)$ was 9.1%, 8.5% and 24.4%, respectively. Compared to $Simp(\mu)$, the speedup for the three scenes in $Coh(\mu)$ was 11%, 8% and 25.8%, respectively. However, this technique ($Simp(P)$ and $Simp(\mu)$) showed frequent visual artifacts which might not have been acceptable in walkthrough or inspection applications and which our latter approach ($Coh(P)$ and $Coh(\mu)$) avoids. Our later approach was therefore qualitatively superior while still managing to be faster.

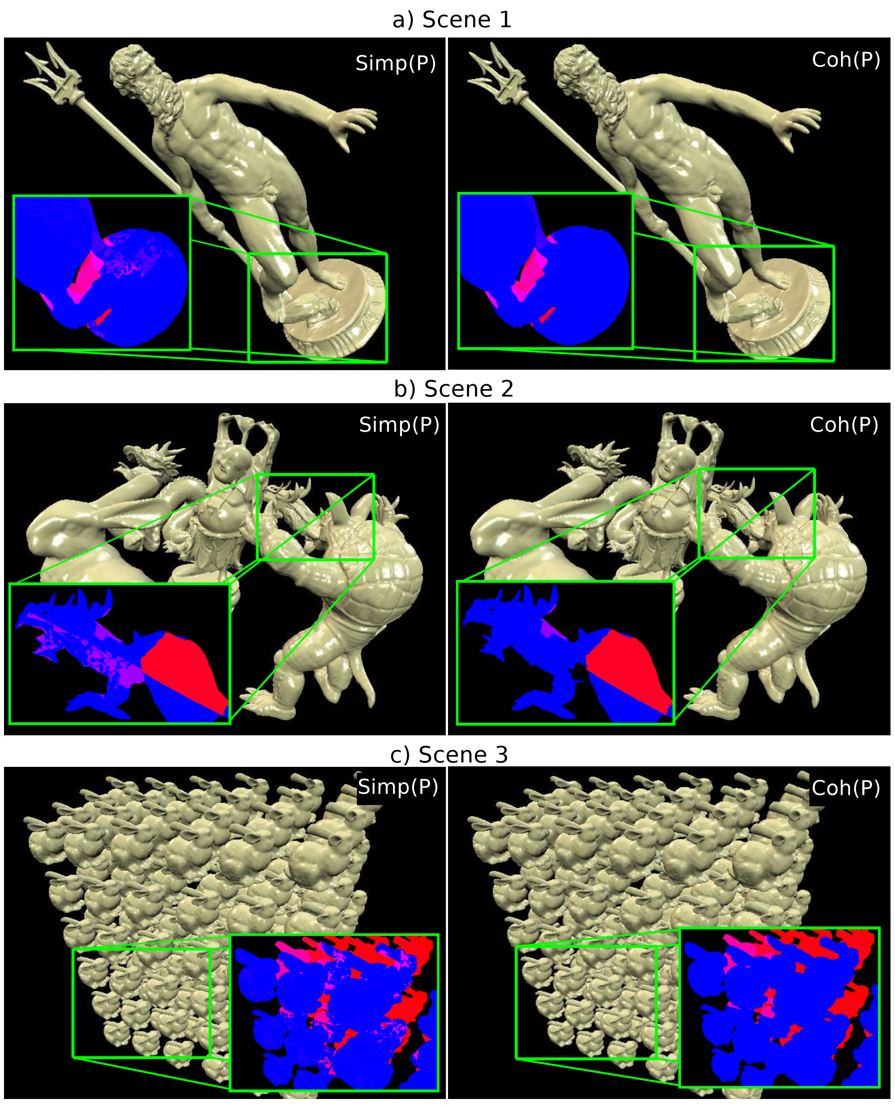

Figure 8.6: Detail in the scenes to show the possible appearance of visual artifacts due to RARN. Whilst the appearance of this visual artifacts is common in $Simp(P)$, this problem was practically eliminated in $Coh(P)$.

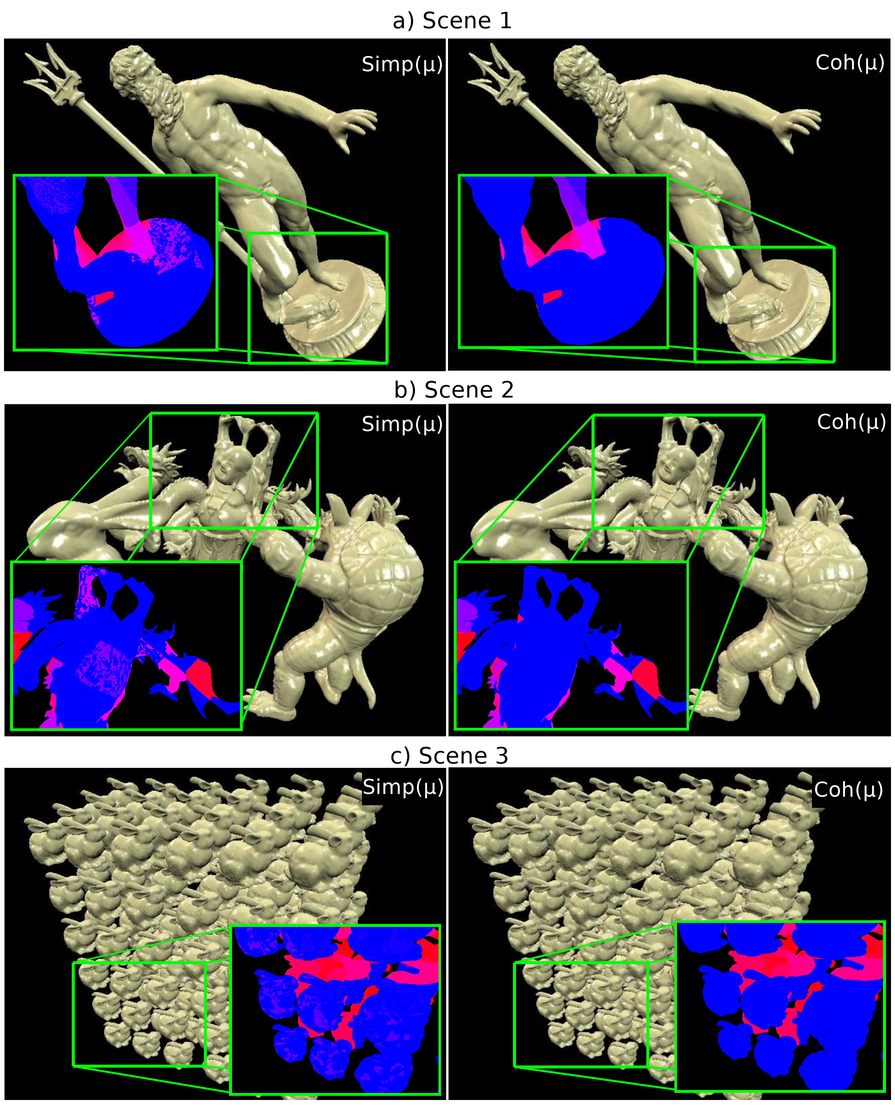

Figure 8.7: Detail in the scenes to show the possible appearance of visual artifacts due to RARN. Whilst the appearance of this visual artifacts is common in $Simp(\mu)$, this problem was practically eliminated in $Coh(\mu)$.

9 Conclusion

This thesis has presented a novel visibility-based refinement error metric which supports both polygon-based and point-based HLODs. A novel coherent HLOD culling algorithm was also introduced which employed the introduced error metric to perform HLOD refinement. Our traversal strategy allowed efficient updating of the front of termination nodes as well as efficient scheduling of HOQs (from which the visibility information was gathered). The main contributions found in our combined approach were:

1. Improved performance while keeping good visual quality: less primitives could be rendered (whilst our occlusion culling technique still remained conservative) with minimal loss in image quality. Moreover, the tradeoff between speedup and image quality could be fine tuned with the parameters provided by our proposed visibility-based error metric, according to user needs.

2. Full use of HOQ results: our error metric took full advantage of the information gathered in HOQs. To avoid HLOD refinement being stopped at a level where the geometric error was still apparent, a simple HOQ-based model was used which took the degree of visibility of a given node into account. Moreover, our model also attempted to exploit the visual masking perceptual phenomenon by taking into account the possible distribution of the visible pixels comprising the regions through which the geometry of the node was reachable (a feature not directly supported by HOQs). Previous approaches have treated HOQs restrictively as if their results were Boolean and they thus become too conservative.

3. Seamless integration between the common strategy for dealing with the latency presented in HOQs, and the use of visibility information within refinement criteria. Our traversal algorithm minimised CPU stalls and GPU starvation by predicting the HLOD refinement condition with high precision using spatio-temporal coherence of visibility. Moreover, our traversal algorithm prevented the appearance of visual artifacts (caused by rendering over nodes previously rendered in wrong LOD) and produced tight prediction for the front of termination nodes. The results indicated that the proposed method was very close to optimal HLOD and HOQ integration respecting the scheduling of queries and the rendering of HLOD geometry.

4. Straightforward implementation. Our approach can also be easily integrated into existing HLOD-based systems supporting HOQs.

Future work will be aimed at improving the proof-of-concept implementation and releasing the algorithm as an integral component of a visualisation system for complex models. Plans are also being made for investigating the use of tighter bounding volumes for HOQs.

Bibliography

[ARFPB90] Airey J. M., Rohlf J. H., Frederick P. Brooks J.: Towards image realism with interactive update rates in complex virtual building environments. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics* (New York, NY, USA, 1990), ACM Press, pp. 41–50.

[AS] Aim-Shape: The aim@shape shape repository. `http://shapes.aimatshape.net/`.

[ASVNB00] Andújar C., Saona-Vázquez C., Navazo I., Brunet P.: Integrating occlusion culling with levels of detail through hardly-visible sets. *Computer Graphics Forum (Proceedings of Eurographics '00)*, 3 (2000), 499–506.

[BH01] Bittner J., Havran V.: Exploiting coherence in hierarchical visibility algorithms. *Journal of Visualization and Computer Animation 12*, 5 (2001), 277–286.

[BM95] Bolin M. R., Meyer G. W.: A frequency based ray tracer. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1995), ACM Press, pp. 409–418.

[BMH98] Bartz D., Meissner M., Hüttner T.: Extending graphics hardware for occlusion queries in opengl. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware* (1998), ACM Press.

[BWPP04] Bittner J., Wimmer M., Piringer H., Purgathofer W.: Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum 23*, 3 (Sept. 2004), 615–624.

[CBWR07] Charalambos J. P., Bittner J., Wimmer M., Romero E.: Optimized hlod refinement driven by hardware occlusion queries. In *Advances in Visual Computing* (2007), Bebis G., Boyle R., Parvin B., Koracin D., Paragios N., Tanveer S.-M., Ju T., Liu Z., Coquillart S., Cruz-Neira C., Müller T., Malzbender T., (Eds.), vol. 4841 of *LNCS*, Springer, pp. 106–117.

[CGG*03] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., Scopigno R.: BDAM – batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum 22*, 3 (September 2003), 505–514.

[CGG*04] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., Scopigno R.: Adaptive TetraPuzzles – efficient out-of-core construction and visualization of gigantic polygonal models. *ACM Transactions on Graphics 23*, 3 (August 2004). Proc. SIGGRAPH 2004.

[Cha06a] Charalambos J. P.: Coherent hierarchical level-of-detail (hlod) refinement through hardware occlusion queries. In *SIACG 2006 - Ibero-American Symposium on Computer Graphics* (July 2006), Universidad Santiago de Compostela, Spain, Eurographics Association.

[Cha06b] Charalambos J. P.: Virtual multiresolution screen space errors: Hierarchical level-of-detail (hlod) refinement through hardware occlusion queries. In *GMAI* (2006), IEEE Computer Society, pp. 221–227.

[CKS03] Correa W. T., Klosowski J. T., Silva C. T.: Visibility-based prefetching for interactive out-of-core rendering. In *PVG '03: Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (Washington, DC, USA, 2003), IEEE Computer Society, p. 2.

[Cla76] Clark J. H.: Hierarchical geometric models for visible surface algorithms. *Commun. ACM 19*, 10 (1976), 547–554.

[COCSD02] Cohen-Or D., Chrysanthou Y., Silva C. T., Durand F.: A survey of visibility for walkthrough applications. *IEEE Transaction on Visualization and Computer Graphics* (2002).

[CT97] Coorg S., Teller S.: Real-time occlusion culling for models with large occluders. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics* (New York, NY, USA, 1997), ACM Press, pp. 83–ff.

[DDTP00] Durand F., Drettakis G., Thollot J., Puech C.: Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000* (July 2000). Held in New Orleans, Louisiana.

[Dur00] Durand F.: A multidisciplinary survey of visibility, 2000.

[EMWVB01] Erikson C., Manocha D., William V. Baxter I.: Hlods for faster display of large static and dynamic environments. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (New York, NY, USA, 2001), ACM Press, pp. 111–120.

[ESSS01] El-Sana J., Sokolovsky N., Silva C. T.: Integrating occlusion culling with view-dependent rendering. In *Proceedings of the conference on Visualization '01* (2001), pp. 371 – 378.

[FS93] Funkhouser T. A., Sequin C. H.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer* *graphics and interactive techniques* (New York, NY, USA, 1993), ACM Press, pp. 247–254.

[FSPG97] FERWERDA J. A., SHIRLEY P., PATTANAIK S. N., GREENBERG D. P.: A model of visual masking for computer graphics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 143–152.

[GBK03] GUTHE M., BORODIN P., KLEIN R.: Efficient view-dependent out-of-core visualization. In *The 4th International Conference on Virtual Reality and its Application in Industry (VRAI'2003)* (October 2003).

[GBK06] GUTHE M., BALÁZS Á., KLEIN R.: Near optimal hierarchical culling: Performance driven use of hardware occlusion queries. In *Eurographics Symposium on Rendering 2006* (June 2006), Akenine-Möller T., Heidrich W., (Eds.), The Eurographics Association.

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 209–216.

[GKM93] GREENE N., KASS M., MILLER G.: Hierarchical z-buffer visibility. In *Proceedings of SIGGRAPH '93* (1993), *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 231–238.

[GM05] GOBBETTI E., MARTON F.: Far Voxels – a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. *ACM Transactions on Graphics 24*, 3 (August 2005), 878–885. Proc. SIGGRAPH 2005.

[GW01] GONZALEZ, WOODS: *Digital Image Processing, 2nd edition.* Prentice Hall, 2001.

[HGJ03] HA H., GREGORSKI B., JOY K. I.: Out-of-core interactive display of large meshes using an oriented bounding box-based hardware depth query. *Computer Graphics Forum 22*, 3 (2003).

[HMC*97] HUDSON T., MANOCHA D., COHEN J. D., LIN M. C., HOFF III K. E., ZHANG H.: Accelerated occlusion culling using shadow frusta. In *Proceedings of the 14th ACM Symposium on Computational Geometry* (1997), pp. 1–10.

[Hop96] HOPPE H.: Progressive meshes. *Computer Graphics 30*, Annual Conference Series (1996), 99–108.

[Hop97] HOPPE H.: View-dependent refinement of progressive meshes. *Computer Graphics 31*, Annual Conference Series (1997), 189–198.

[KB05] KIM C. Y., BLAKE R.: Psychophysical magic: rendering the visible 'invisible'. *Trends Cogn Sci 9*, 8 (August 2005), 381–388.

[KL01] KIM J., LEE S.: Truly selective refinement of progressive meshes. In *Proceedings of Graphics Interface 2001* (2001), Watson B., Buchanan J. W., (Eds.), pp. 101–110.

[KS99] KLOSOWSKI J. T., SILVA C. T.: Rendering on a budget: a framework for time-critical rendering. In *VIS '99: Proceedings of the conference on Visualization '99* (Los Alamitos, CA, USA, 1999), IEEE Computer Society Press, pp. 115–122.

[KS00] KLOSOWSKI J. T., SILVA C. T.: The prioritized-layered projection algorithm for visible set estimation. *IEEE Transaction on Visualization and Computer Graphics* (2000), 108–123.

[KS01] KLOSOWSKI J. T., SILVA C. T.: Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics 7*, 4 (2001), 365–379.

[LE97] LUEBKE D., ERIKSON C.: View-dependent simplification of arbitrary polygonal environments. *Computer Graphics 31*, Annual Conference Series (1997), 199–208.

[Lin03] LINDSTROM P.: Out-of-core construction and visualization of multiresolution surfaces. In *SI3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics* (New York, NY, USA, 2003), ACM Press, pp. 93–102.

[LP01] LINDSTROM P., PASCUCCI V.: Visualization of large terrains made easy. In *VIS '01: Proceedings of the conference on Visualization '01* (Washington, DC, USA, 2001), IEEE Computer Society, pp. 363–371.

[LPC*00] LEVOY M., PULLI K., CURLESS B., RUSINKIEWICZ S., KOLLER D., PEREIRA L., GINZTON M., ANDERSON S., DAVIS J., GINSBERG J., SHADE J., FULK D.: The digital michelangelo project: 3d scanning of large statues. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 131–144.

[Lue01] LUEBKE D. P.: A developer's survey of polygonal simplification algorithms. *IEEE Comput. Graph. Appl. 21*, 3 (2001), 24–35.

[LWC*02] LUEBKE D., WATSON B., COHEN J. D., REDDY M., VARSHNEY A.: *Level of Detail for 3D Graphics.* Elsevier Science Inc., New York, NY, USA, 2002.

[MCC*99] Mirin A. A., Cohen R. H., Curtis B. C., Dannevik W. P., Dimits A. M., Duchaineau M. A., Eliason D. E., Schikore D. R., Anderson S. E., Porter D. H., Woodward P. R., Shieh L. J., White S. W.: Very high resolution simulation of compressible turbulence on the IBM-SP system.

[Net89] Netravali A.: *Digital Pictures, Representation and Compression*. Plenum, 1989.

[Pas02] Pascucci V.: Slow growing subdivision (sgs) in any dimension: Towards removing the curse of dimensionality. *Comput. Graph. Forum 21*, 3 (2002).

[PS97] Puppo E., Scopigno R.: Simplification, lod and multiresolution, principles and applications. *Computer Graphics Forum (Proceedings of Eurographics 1997)* (1997).

[RL00] Rusinkiewicz S., Levoy M.: Qsplat: a multiresolution point rendering system for large meshes. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 343–352.

[SA04] Segal M., Akeley K.: *The OpenGL Graphics System: A Specification (Version 2.0)*, 2004. `http://www.opengl.org`.

[SDDS00] Schaufler G., Dorsey J., Decoret X., Sillion F. X.: Conservative volumetric visibility with occluder fusion. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 229–238.

[SU] Standford-University: The stanford 3d scanning repository. `http://graphics.stanford.edu/data/3Dscanrep`.

[TM02] Taubman D., Marcellin M. W.: *JPEG2000 Image Compression, Fundamentals, Standards and Practice.* Kluwer Academic Publishers, 2002.

[TS91] Teller S. J., Séquin C. H.: Visibility preprocessing for interactive walkthroughs. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1991), ACM Press, pp. 61–70.

[VM02] Varadhan G., Manocha D.: Out-of-core rendering of massive geometric datasets. In *VIS '02: Proceedings of the conference on Visualization '02* (Washington, DC, USA, 2002), IEEE Computer Society.

[Wik07] Wikipedia: Rendering (computer graphics) — Wikipedia, the free encyclopedia, 2007. [Online; accessed 25-July-2007] `http://en.wikipedia.org/wiki/Rendering_(computer_graphics)`.

[XESV97] Xia J. C., El-Sana J., Varshney A.: Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (1997), 171–183.

[YSGM04] Yoon S.-E., Salomon B., Gayle R., Manocha D.: Quick-vdr: Interactive view-dependent rendering of massive models. In *VIS '04: Proceedings of the conference on Visualization '04* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 131–138.

[ZMHH97] Zhang H., Manocha D., Hudson T., Hoff III K. E.: Visibility culling using hierarchical occlusion maps. In *Proceedings of SIGGRAPH '97* (1997), *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 77–88.