Ibero-American Symposium on Computer Graphics - SIACG (2006)
P. Brunet, N. Correia, and G. Baranoski (Editors)

Coherent Hierarchical Level-of-Detail (HLOD) Refinement Through Hardware Occlusion Queries

Jean Pierre Charalambos[1,2]

[1]National University of Colombia
[2]Departament de LSI, Universitat Politecnica de Catalunya, Barcelona, Spain

Abstract

We present a coherent hierarchical level of detail (HLOD) culling algorithm that employs a novel metric to perform the refinement of a HLOD-based system that takes into account visibility information. The information is gathered from the result of a hardware occlusion query (HOQ) performed on the bounding volume of a given node in the hierarchy. Although the advantages of doing this are clear, previous approaches treat refinement criteria and HOQ as independent subjects. For this reason, HOQs have been used restrictively as if their result were boolean. In contrast to that, we fully exploit the results of the queries to be able to take into account visibility information within refinement conditions. We do this by interpreting the result of a given HOQ as the virtual resolution of a screen space where the refinement decision takes place. In order to be able to use our proposed metric to perform the refinement of the HLOD hierarchy as well as to schedule HOQs, we exploit the spatial and temporal coherence inherent to hierarchical representations. Despite the simplicity of our approach, in our experiments we obtained a substantial performance boost (compared to previous approaches) in the frame-rate with minimal loss in image quality.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Viewing algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Visualization of complex models comprising several million polygons is a very active research area [RL00, EMWVB01, GBBK04, YSGM04, CGG*04, GM05]. For the interactive rendering of such models, hierarchical level of detail (HLOD) methods have proven to be the most efficient approach. Typically, a HLOD-based system is built performing two recursive offline steps: 1. Without assuming anything about the topological genus of the underlying model, a scene hierarchy is built in a top-down manner using spatial subdivision driven solely by a user-defined target number of primitives per node. 2. For each node of this hierarchy a single, or a few levels of detail are generated using offline simplification in a bottom-up manner. HLODs support out-of-core algorithms in a straightforward way, and allow an optimal balance between CPU and GPU load during rendering [GBBK04]. The HLODs either consists of a point-based [RL00], or polygon-based approximation of the model [EMWVB01].

During runtime visualization of the model two of the main tasks needed to be accomplished are: the refinement of the HLOD hierarchy and performing the occlusion culling.

The quantity that has traditionally driven the refinement of the hierarchy is the *screen projection error*. This quantity corresponds to the *number of pixels* obtained when projecting onto the screen a given *model space error* related to a node (see Section 2.1).

The preferred method of performing the occlusion culling has been through hardware occlusion queries (HOQs), (see Section 2.2). To test the visibility of a node, a HOQ is issued on the node bounding volume. The returning value of the test corresponds to the precise *number of pixels* of the volume that would result as being visible (see Section 2.2.2). Since the main disadvantage of HOQs is that there is a latency between issuing a query and the availability of the result, the scheduling of the queries should be performed carefully. The preferred method for doing this has been the *coherent hierarchical culling* algorithm [BWPP04]. For the scheduling of

queries, the method fully exploits the spatial and temporal coherence of visibility (see Section 2.2.1).

Occlusion culling offers an excellent measurement for HLOD refinement criteria. According to the degree of partial occlusion of a node, it could be determined that there would be no *gain* in the final appearance of the image obtained if the node were further refined. However, to our knowledge there are no HLOD approaches which take visibility information as an integral part of the refinement condition (see Section 3). With this purpose in mind, in this paper we provide the means to relate the previously stated refinement criterion with HOQs (see Section 4). Moreover, in order to use our introduced metric to perform the hierarchical refinement, and to avoid latency times due to HOQs, we exploit the spatial and temporal coherence of visibility inherent to hierarchical representations (see Section 5).

Some of the properties of our approach are: 1. Improved performance with the same visual quality: we are able to render less primitives with minimal loss in image quality (see Section 6); 2. Generality: our metric supports both polygon-based as well as point-based HLODs; 3. Full use of the result of the HOQ: our metric takes full advantage of the information gathered in HOQs; 4. Full use of the spatial and temporal coherency inherent to hierarchical representations; and 5. Straightforward implementation.

2. Related Work

2.1. Refinement criteria: model and screen projection errors

2.1.1. HLOD refinement

The screen projection error is the quantity that has traditionally driven the refinement of a HLOD-based system. In a top down traversal of the hierarchy the refinement condition may be written as follows: if ($\varepsilon \leq \tau$) then *stop hierarchical refinement* (e.g., [CGG*04]), where: ε corresponds to the screen projection error, i.e., the projection onto the screen of a model space measurement λ, and τ corresponds to the user specified threshold given in pixels (sometimes known as *pixels of error* [YSGM04]). Observe that the proposition ($\varepsilon \leq \tau$) is referred as *StopHLODRefinement(node)* in the algorithms of Section 2.2.1.

In polygon-based HLODs, λ corresponds to the *model space error* due to the simplification of the geometry related to a given node in the hierarchy. Readers may refer to Lindstrom [Lin03], for a possible estimation of λ from the *quadric error metrics* [GH97]. An upper bound of ε could be obtained by measuring in pixels the projected diameter of a sphere with diameter equal to λ and centered at the node bounding sphere point closest to the viewpoint [CGG*04] (see Figure 1).

In point-based HLODs, λ corresponds to the bounding volume of a given node, e.g., the node bounding sphere [RL00], or the node bounding box [GM05].

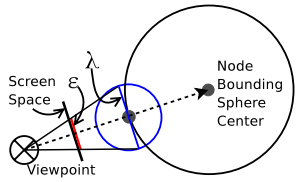

Figure 1: *Screen projection errors in polygon-based HLODs.*

2.1.2. Refinement within the nodes

During runtime inspection of the model, in polygon-based HLODs *popping artifacts* may appear when refining (or coarsing) in a given frame a previous visible node. Those visual artifacts are prominent in hierarchies with fast growing subdivision [Pas02]. To avoid their appearance, instead of a single level of detail, a few levels of detail are related to each node. Thus, an additional finer-grained refinement needs to be achieved within the nodes of the hierarchy [YSGM04]. To perform this refinement a range of model space errors $[\lambda_{min}, \lambda_{max}]$, is first (during the simplification step) related to each node. When visiting a node during runtime, τ is *mapped back* from screen to model space to obtain λ_τ, i.e., λ_τ simply corresponds to the value of the model space error that when projected onto the screen leads to $\varepsilon = \tau$. The value of λ_τ is then used to: 1. Refine the hierarchy by means of the following refinement condition: if ($\lambda_{min} \leq \lambda_\tau \leq \lambda_{max}$) then *stop hierarchical refinement*. 2. Refine the geometry within the node. Readers may refer to Hoppe [Hop97] for the details of how to accomplish the refinement of a geometric model when λ_τ is given.

2.2. Hardware occlusion queries

The HOQ scan converts a set of graphics primitives (but does not render them to screen), and determines whether or not any pixels in the frame buffer would be affected if the primitives were actually rendered to the screen [HGJ03]. HOQs have several advantages: generality of occluders, occluder fusion, generality of occludees, better use of GPU power, and easy use. For this reason, HOQs have become the preferred method in those HLOD-based systems implementing occlusion culling, e.g., [YSGM04, GM05]. However, their main disadvantage is that there is a latency between issuing a query and the availability of the result [BWPP04].

Currently, the two main supported properties of HOQs in OpenGL are:

Property 1. Multiple occlusion queries may be sent at once. See Section 2.2.1.

Property 2. The returning value corresponds to the number of visible pixels of the queried object, but without telling anything about their position. See Section 2.2.2.

2.2.1. Coherent HLOD culling

By means of the first property stated above, authors have focused their attention on avoiding CPU stalls due to latency. An elegant and powerful method to minimize them was introduced by Bittner *et al.,* [BWPP04]. The method fully exploits the spatial and temporal coherence of visibility, inherent to hierarchical representations.

This method could easily be adapted to any HLOD-based system, as it is done in Gobetti *et al.,* [GM05] (see Algorithm 1). The method, to be executed once per frame, comprises two parts: *process finished HOQs* (Algorithm 2), and *hierarchical traversal* (Algorithm 3).

Algorithm 1 CoherentHLODCulling

```
1: PriorityQueue.Enqueue(hierarchy.Root)
2: while (not PriorityQueue.Empty() or not
   QueryQueue.Empty( )) do
3:    ProcessFinishedHOQs
4:    HierarchicalTraversal
5: end while
```

In Algorithm 2, HOQs are processed once their result is available or when the priority queue is empty (see line 1 of Algorithm 2). Only in the latter case the algorithm waits for the result of HOQs. If a node is determined to be visible (see line 4 of Algorithm 2): 1. The node and its ancestors are marked as visible (see line 5 of Algorithm 2). 2. The node is attempted to be rendered immediately or their children are scheduled for traversal (see line 7 of Algorithm 2, and Algorithm 4).

Algorithm 2 ProcessFinishedHOQs

```
1: while (not QueryQueue.Empty() and (ResultAvail-
   able(QueryQueue.Front()) or PriorityQueue.Empty()))
   do
2:    node←QueryQueue.Dequeue()
3:    visiblePixels←GetOcclusionQueryResult(node)
4:    if (visiblePixels> 0) then
5:       PullUpVisibility(node)
6:       stopRefinement←StopHLODRefinement(node)
7:       TraverseNode(node, stopRefinement)
8:    end if
9: end while
```

In Algorithm 3 the hierarchy is traversed in a top-down and in an *approximated* front-to-back traversal order, i.e., the nodes are scheduled for traversal, using a priority queue [BWPP04]. While performing the traversal: 1. HOQs are issued (and enqueued in a query queue) only for the bounding boxes of previous invisible nodes or for the bounding boxes of those nodes where it was determined that the HLOD refinement had stopped (see line 9 of Algorithm 3). 2. Without waiting for the result of HOQs, previous visible nodes are attempted to be rendered immediately, or their children are scheduled for traversal (see line 14 of Algorithm 3, and Algorithm 4).

Algorithm 3 HierarchicalTraversal

```
 1: if (not PriorityQueue.Empty()) then
 2:    node←PriorityQueue.Dequeue( )
 3:    if (InsideViewFrustum(node)) then
 4:       wasVisible←node.visible and (node.lastVisited =
          frameID-1)
 5:       stopRefinement←StopHLODRefinement(node)
 6:       stopRefinementOrWasInvisible←not wasVisible
          or stopRefinement
 7:       node.visible←false
 8:       node.lastVisited←frameID
 9:       if (stopRefinementOrWasInvisible) then
10:          IssueOcclusionQuery(node)
11:          QueryQueue.Enqueue(node)
12:       end if
13:       if (wasVisible) then
14:          TraverseNode(node, stopRefinement)
15:       end if
16:    end if
17: end if
```

Algorithm 4 TraverseNode(node, stopRefinement)

```
1: if (stopRefinement) then
2:    Render(node)
3: else
4:    PriorityQueue.EnqueueChildren(node)
5: end if
```

In the following we will only discuss the parts of the algorithm that are most closely related to our work (see Section 5). For a thorough discussion on the subject readers may refer to Bittner *et al.,* [BWPP04] and Gobetti *et al.,* [GM05].

2.2.2. Returning values of HOQs

To our knowledge the second property stated above has received little or no attention. In all previous HLOD approaches to test the visibility of a node in the hierarchy, a HOQ is issued on its bounding volume. If there are no visible pixels, the object related to the node is culled away. If there is at least one visible pixel and if ($\varepsilon > \tau$), then the node is further refined [YSGM04, GM05], i.e., HOQs have been used as if their result were boolean. The main issue with this approach is that it results as being too conservative, particularly when the number of visible pixels is low. Therefore, our approach is to use the precise number of visible pixels returned from HOQs as integral part of the HLOD refinement criterion (see Section 4).

3. Overview

Few approaches exist that integrate LODs with occlusion culling. A *cell/object-precision* occlusion culling method was introduced in Andujar *et al.,* [ASVNB00] (readers may refer to Cohen *et al.*, [COCSD02] for a taxonomy of occlusion culling methods). The scene is divided into cells and for each cell the objects are classified into sets according to their visibility degree, i.e., a representative error that measures the possible contribution in pixels to the final image is assigned to each set. From the selection of some objects that act as occluders, the sets related to each cell are obtained during a preprocess. During runtime the error is used to: discard the objects of the sets not meeting a user defined threshold; or, for the objects having a multiresolution representation, to select the proper level-of-detail needed to display them. Unfortunately, this method is incompatible with HLODs: to perform a fixed preselection of objects as occluders in a HLOD-based system seems a very difficult task, i.e., every single node in the HLOD hierarchy represents potentially both, a good occluder and an occludee. Thus, the occlusion method should take into account the occlusions caused by all the *objects* in the scene [COCSD02].

Our idea of integrating occlusion culling directly into the refinement criterion of a HLOD-based system is based on the following observation. As previously stated, during rendering time, state-of-the-art hierarchical refinement corresponds solely to the comparison between a screen space error, given in number of pixels, and a user specified threshold, (see Section 2.1). However, because of its advantages the preferred method of performing the occlusion culling is through HOQs, (see Section 2.2). Nevertheless, the result of the queries is the number of visible pixels of a given object. Since occlusion culling also offers an excellent refinement criterion (see Section 1), it seems reasonable to integrate it into the above refinement condition. In this way we expect to obtain an increment in the average rendering frame-rate: once we determine that there is no gain in further refining a node, due to partial occlusion, we could stop the refinement in advance, sending less primitives to the graphics card while achieving the same approximated visual quality.

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

In Section 4 we introduce our proposed metric to use the result of HOQs directly within refinement conditions, and in Section 5 we show our approach to exploit the spatial and temporal coherence to avoid latencies, meanwhile using our introduced metric to perform the hierarchical refinement.

4. Virtual multiresolution screen space errors

In contrast to previous approaches our novelty is that we employ the result of HOQs as an integral part of the HLOD refinement criterion. We interpret the result of a given HOQ as the virtual resolution of a screen space where we are going to project a given model space error λ. In all previous approaches λ is directly projected onto the screen space S, at its full original resolution, (see Section 2.1). Similar to previous approaches we also use the value of the new error metric, dubbed *virtual multiresolution screen space error*, as the quantity that guides refinement.

Let M be the space on the screen defined as the subset of S that corresponds to the projection of a given node's bounding volume B. Observe that M bounds the projection onto S of the model space error related to B, λ; and also the (eventual) projection of its related geometry G. Let us say that the resolution of M, denoted by α, corresponds to the number of pixels of the projection of B onto S. Let q be the number of visible pixels on S obtained when a HOQ on B is issued, i.e., the result of the query; and q' be the number of invisible pixels. Observe that $q' = \alpha - q$, and that $0 \leq q \leq \alpha$ always holds. Also note that, if in the moment when the query was issued there were no objects between S and B, then $\alpha = q$.

Suppose that the result of the HOQ performed on B is available, i.e., we know q; and remember that we want to use its value as an integral part of the refinement criterion, i.e., we want to establish how q could affect ε. However, in polygon-based HLODs λ is not subject to occlusion: for the sole purpose of projecting λ we always need to assume that all the pixels of M are visible (see Section 2.1.1). Our approach is therefore to coarsen the resolution of M by means of q, i.e., we calculate from q a coarser virtual screen space resolution δ, $\delta = f(q) \leq \alpha$, for the screen space where we are going to project λ. The visual appearance obtained when projecting G due to ε, could then be approximated using δ in one of two ways:

1. Calculate ε in the original (real) resolution α, while projecting G at the coarse (virtual) resolution δ.
2. Calculate ε in the coarse (virtual) resolution δ, while projecting G at the original (real) resolution α.

In terms of the *biased* (see below) visual appearance obtained, the two statements above are *almost* equivalent. However, in the latter case we would be able to stop the refinement higher in the hierarchy allowing us to send less primitives to the GPU, i.e., effectively taking into account the result of the HOQ.

4.1. How much to coarsen a virtual resolution

To best decide how to calculate δ it would be necessary to measure the introduced bias, i.e., the loss in image quality. However, to characterize the bias it would be necessary to know the exact position of the pixels comprising q. Since

this functionality is not currently present in HOQs (see Section 2.2) in this paper we simply assume that $\delta = q$. That is, in order to project λ we coarsen the resolution of M from α to q. Observe that for extreme visibility conditions this assumption leads to *reasonable* results, i.e., if $q = 0$ then the value of ε at the coarse virtual resolution would be 0; and if $q = \alpha$ then the value of ε at the coarse virtual resolution would be the same as if no visibility information would have been gathered. Fortunately, for most practical purposes we have found this to be a good approximation. Most likely because the closer q gets to 0, the bias gets a higher chance to decrease (see Section 6).

4.2. Switching among multiple virtual resolutions

Since we can not allow an actual change in the resolution to take place, we now seek to compute ε at a virtual resolution q, from its computation at α. We can then take the value of the former to effectively refine the hierarchy, without actually modifying the original resolution. For this, we show how ε could be *virtually* switched among multiple resolutions.

We begin by extending the definition of the screen projection error according to a given screen resolution γ. First suppose that when projecting λ onto the screen space, each of the obtained pixels occupy a position (i, j). Let us say that the *virtual multiresolution screen projection error* ε^γ at a given screen resolution γ, is given by the following expression:

$$\varepsilon^\gamma = \sum_i \sum_j I^\gamma_{i,j}$$

Where $I^\gamma_{i,j}$ corresponds to the intensity value of the pixel at the screen position (i, j), and at the given screen resolution γ. Note that if $\gamma = \alpha$, then ε^γ reduces to the original definition of ε, i.e., in this case $I^\gamma_{i,j}$ are 0 or 1, and thus ε^γ simply corresponds to the number of pixels of the projected error.

Now let a^γ be the area occupied by a single pixel at a given screen resolution γ. To keep the estimation of the error consistent when *virtually* switching the resolutions, the total intensity due to ε^γ should be maintained constant, i.e., this will produce exactly the same effect on ε, as if we were to *actually* change the screen resolution. Therefore, since the total intensity due to ε^γ is given by $a^\gamma \sum_i \sum_j I^\gamma_{i,j}$, it follows that:

$$a^q \sum_i \sum_j I^q_{i,j} = a^\alpha \sum_i \sum_j I^\alpha_{i,j}$$

However, since $a^q * q = a^\alpha * \alpha$ and $\varepsilon = \varepsilon^\alpha$, we finally get:

$$\varepsilon^q = \varepsilon * (q/\alpha)$$

Thus, our hierarchical refinement condition that takes into account HOQs may be simply written as: if $((\varepsilon * (q/\alpha)) \leq \tau)$ then *stop hierarchical refinement.*

4.3. Estimation of the number of pixels obtained when projecting a node bounding box

It only remains to show how α could be calculated. The *exact* computation of α (see the previous observation regarding when $\alpha = q$) suffers from a double drawback: we would need to issue an additional HOQ on a reset z-buffer. While there are several ways to approximate the value of α, our approach is to do it as follows. As a part of the preprocess: 1. We first build a cube $\widehat{B}$ with the same volume as B. 2. On an orthogonal view direction from the center of one of the faces of $\widehat{B}$, we calculate the shortest distance $\widehat{d}$, at which the cube is completely visible and count the number of pixels $\widehat{f}$, of the face. 3. We keep at the node the values of $\widehat{f}$ and $\widehat{d}$. It is easy to see that the estimated value of α, to be used at runtime, is then simple given by:

$$\widehat{\alpha} = \widehat{f} * (\widehat{d}/d)^2$$

Where d is the same distance used to calculate ε, i.e., the distance from the viewer to the center of the node's bounding sphere, at a given frame. Observe that the above estimation leads *almost* always to an underestimated value of α, i.e., otherwise we would be underestimating the value of ε^q, something that is better avoided.

4.4. Refinement within the nodes

So far we have only focused our attention on hierarchical refinement. However, we can easily adapt our above framework for those hierarchies where it is necessary to further refine the model within the nodes (see Section 2.1.2). Since the user specified threshold is given in screen space, we can use the result of the query to adjust τ exactly in the same way as we did with ε, i.e., we can simply: 1. Compute τ^q as $\tau^q = \tau * (q/\alpha)$; 2. Compute λ_{τ^q} by mapping back τ^q to the model space; and, 3. Use λ_{τ^q} to refine the model.

5. Coherent HLOD culling algorithm

In order to exploit the spatial and temporal coherence of visibility inherent to hierarchical representations while using our above introduced metric, we have departed from Algorithm 1 (see Section 2.2.1). Remember that we use the result of HOQs (q) as integral part of the refinement condition, i.e., we use ε^q instead of ε as integral part of the hierarchical refinement condition (see line 6 in Algorithm 2 and line 5 in Algorithm 3). In order to do this, we will show our approach adapting both parts of this algorithm in the following.

5.1. Process finished HOQs

We process finished occlusion queries in the same way as in Algorithm 2, but in order to test the hierarchical stop condition (line 6 of Algorithm 2) we simply use ε^q instead of ε.

5.2. Hierarchical traversal

Since at the moment of performing the hierarchical traversal (Algorithm 3) we do not know the result of HOQs, we need to *predict* when the refinement should stop. Observe that the ratio q/α simply represents the visible fraction of the given node bounding box. We call this ratio the *visibility coefficient* of the node μ, $\mu = (q/\alpha)$. Since $0 \leq q \leq \alpha$ (see Section 4) then $0 \leq \mu \leq 1$ always holds. Now let $\widehat{\mu}$ be the *predicted* value of μ. Our approach is to use the value of $\widehat{\mu}$ to perform the hierarchical traversal, i.e., we use the value of $\widehat{\mu}$ as an integral part of the right hand side condition of line 5 of Algorithm 3. Before showing our approach to compute $\widehat{\mu}$, we have identified the following cases relating to the quality of the prediction:

1. If $\widehat{\mu} \cong \mu$ for all the nodes in the hierarchy then we are in the ideal case, i.e., we would be able to render the smallest number of nodes with minimal loss in image quality. In terms of visual quality, this case is equivalent to as if we were using the hierarchical stop-and-wait method [BWPP04], using our new screen space error metric ε^q (see Section 6.1).
2. If $\widehat{\mu} > \mu$ for all the nodes in the hierarchy, then in terms of visual quality a *conservative* case will follow, i.e., we would render more nodes than the ideal number, but *at least* with the same visual quality obtained in the first case stated above. Also observe that in this case we would never be rendering more nodes than if we used Algorithm 1, i.e., if $\widehat{\mu} = 1$ for all the nodes in the hierarchy then we would render exactly the same number of nodes as if we used Algorithm 1.
3. If for a given hierarchy node $\widehat{\mu} < \mu$ then a potential source of visual artifacts arises, i.e., if the node was determined visible in the previous frame, then it is immediately rendered in the current frame (see line 14 of Algorithm 3). However, later on in the same frame when the result of the HOQ performed on the node is available, the node is traversed and thus some of its descendants get the chance to be also rendered within the same frame (see line 7 of Algorithm 2).

We exploit the spatial coherency inherent to the traversal algorithm to compute the value of $\widehat{\mu}$. Our observation is that the position of the node in the priority queue gives a hint of its visibility coefficient. The reason is that the nodes in the priority queue are scheduled in an *approximated* front-to-back traversal order (see Section 2.2.1). Thus, our assumption to compute $\widehat{\mu}$ is that it linearly decreases from 1 to 0 with the position of the node in the priority queue, i.e., we simply employ the following expression $\widehat{\mu} = 1 - (s/S)$, where s is the scheduled position of the node in the priority queue in the current frame (as defined in line 4 of Algorithm 4) and S corresponds to the total number of scheduled nodes in the previous frame.

Fortunately, although its simplicity we have found this approach to be both effective and conservative in terms of visual quality (see Section 6).

6. Results

An experimental software supporting our new metric and our coherent HLOD culling algorithm has been implemented on Linux using C++ with OpenGL. To evaluate the performance boost and the image quality obtained in our approach, we have extensively tested our system with a number of scenes with different depth complexities. We have implemented a geometry-based HLOD with an octree with nearly 2000 triangles per node. We have also employed the quadric error metrics (see [GH97]) to simplify the geometry, and to derive the model space errors [Lin03]. All tests were run on a window size of 640*480 and $\tau = 1$, and on a Pentium M 1.7GHz with a nVidia GeForce Go 6200.

6.1. Tests

For our discussion in this paper we have built three scenes with low, middle and high depth complexities, respectively named as *scene 1*, *scene 2*, and *scene 3* (see Figures 8, 9 and 10). For each scene we have designed a session representing typical inspection tasks. Our inspection sequences include rotations and changes from overall views to extreme close-ups that heavily stress the system. For each scene we have played the sequence to collect data: frame rates and number of drawn nodes. Depending on the traversal algorithm and the metric used to refine the hierarchy, we have evaluated the following configurations:

1. CHLOD-ε^q: the hierarchy was traversed with our coherent HLOD culling algorithm (see Section 5) using our new screen space error metric (ε^q) to refine the hierarchy (see Section 4).
2. CHLOD-ε: the hierarchy was traversed with the coherent culling algorithm version of Gobetti *et al.,* [GM05] (see Section 2.2.1). The metric used to refine the hierarchy was ε.
3. SW-ε^q: the hierarchy was traversed with a top-down/front-to-back traversal algorithm based on bit toggling using our new screen space error metric (ε^q) to refine the hierarchy. Observe that under this configuration there is no avoidance of latency time (see Section 2.2.1), i.e., the traversal algorithm corresponds to the hierarchical *stop-and-wait* method referenced in Bittner *et al.*, [BWPP04]. Also note that this configuration gives the ideal number of nodes to be drawn.
4. SW-ε: the hierarchy was traversed with a top-down/front-to-back traversal algorithm based on bit toggling using ε as the screen space error metric to refine the hierarchy. Observe that under this configuration there is no avoidance of latency time (see Section 2.2.1).

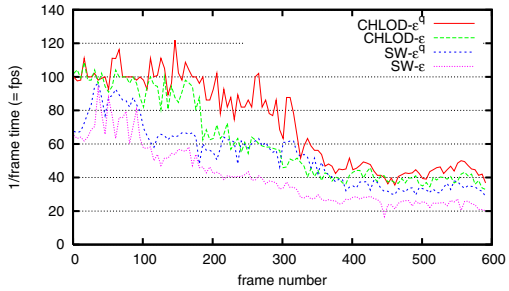

Figure 2: *Frame rates for scene 1 (low depth complexity).*

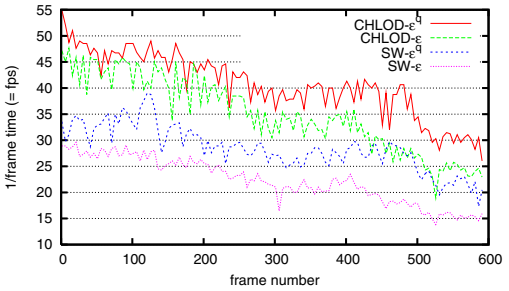

Figure 4: *Frame rates for scene 2 (middle depth complexity).*

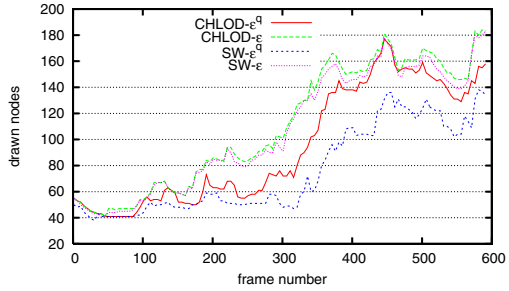

Figure 3: *Drawn nodes for scene 1 (low depth complexity).*

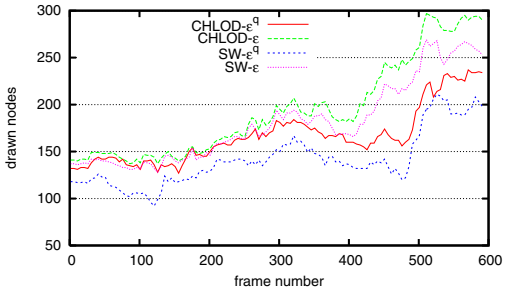

Figure 5: *Drawn nodes for scene 2 (middle depth complexity).*

6.2. Performance boost

6.2.1. Scenes with low depth complexities

In scene 1 we have obtained an average of 17%, 36%, and 79% of performance boost in the frame rate when using CHLOD-ε^q, respectively to CHLOD-ε, SW-ε^q and SW-ε. Figure 2 shows the whole frame rate values of the inspection sequence. The main reason for this improvement was that we were able to send less primitives to the GPU: in respect to CHLOD-ε, we saved an average of 14% of the total number of nodes that needed to be rendered. On average, we rendered only 25% more nodes than in SW-ε^q, which gives the ideal number (see Section 6.1). Figure 3 shows the whole sequence of drawn nodes. All scene statistics have been summarized in Table 1.

6.2.2. Scenes with middle depth complexities

In scene 2 we have achieved similar results. We obtained an average of 16%, 42%, and 82% of performance boost in the frame rate when using CHLOD-ε^q, respectively to CHLOD-ε, SW-ε^q and SW-ε. Figure 4 shows the whole frame rate values of the inspection sequence. In respect to CHLOD-ε, we saved an average of 14% of the total number of nodes that needed to be rendered. We were able to render only 18% more than the nodes found by SW-ε^q, which gives the ideal number. Figure 5 shows the whole sequence of drawn nodes. All scene statistics have been summarized in Table 1.

6.2.3. Scenes with high depth complexities

In scene 3 we have achieved more impressive results. We obtained an average of 31%, 44%, and 122% of performance boost in the frame rate when using CHLOD-ε^q, respectively to CHLOD-ε, SW-ε^q and SW-ε. Figure 6 shows the whole frame rate values of the inspection sequence. In average, we were able to save 21% of the total number of nodes that needed to be rendered, in respect to CHLOD-ε. However, we rendered 53% more than the nodes found by SW-ε^q. We found that this increment in the average arose under extreme close-up visibility conditions where the number of rendered nodes in CHLOD-ε^q was a lot higher than in SW-ε^q, see frames 350-500 in Figure 7 (the figure shows the whole sequence with the number of drawn nodes). All scene statistics have been summarized in Table 1.

In general, we have found that the higher the depth complexity of a scene is our metric performs better. However, we have found that our culling traversal algorithm performs too conservative in scenes with higher depth complexity under extreme close-up viewing conditions. Under these circumstances, even SW-ε^q could sometimes perform a little better, e.g., (see frames 450-500 in Figure 6). This is one of our main research avenues (see Section 7). For some frames of the inspection sequences (frames 200-400, 450-520, and 250-500 in Figures 2, 4 and 6, respectively) we have also found that SW-ε^q performs equally or even better than

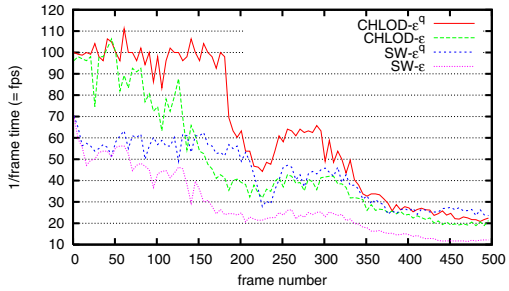

Figure 6: *Frame rates for scene 3 (high depth complexity).*

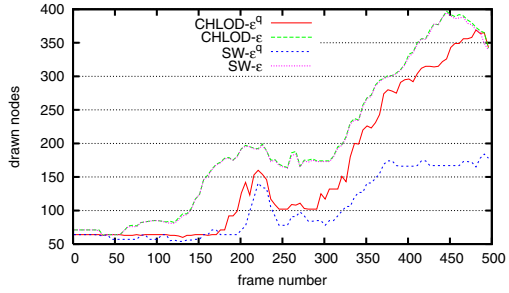

Figure 7: *Drawn nodes for scene 3 (high depth complexity).*

CHLOD-ε. SW-ε always gives the poorest performance, as expected. Since our algorithm is always able to render less nodes than CHLOD-ε, it *almost* always performs better than CHLOD-ε.

6.3. Image quality

Figures 8, 9 and 10 illustrate the image quality of the three scenes obtained when using CHLOD-ε and CHLOD-ε^q. Note that the use of our technique does not alter the final image quality perceived in these close-up frames of the inspection sequences.

To show the effect of the degree of correction of ε when refining the hierarchy using ε^q, the view-frustum view of the bounding boxes of the nodes selected to be drawn are shown in the top row of Figures 8, 9 and 10. Moreover, we have coloured the node bounding boxes from blue to magenta to red according to the degree of correction of ε: blue means low correction ($\varepsilon^q \cong \varepsilon$), magenta means middle correction ($\varepsilon^q < \varepsilon$) and red means high correction ($\varepsilon^q \ll \varepsilon$), see the top of Figures 8(b), 9(b) and 10(b). Because of the visibility information gathered with HOQs, observe that the errors of the nodes near to the view point, *tend* to get lower correction than those that are farther. This tendency is not always strictly respected, as it is illustrated in the nodes farthest to the view point that gets low correction in top of Figure 9(b). The reason for that is in CHLOD-ε^q (and in CHLOD-ε as well) we use a *compatible* (but not exact) version of a front-to-back traversal algorithm [BWPP04]. The only issue with this approach, in respect with our prediction task (see Section 5) is that it could result as being too conservative (see Sections 5 and 6.2). Thus, in CHLOD-ε^q as well as in SW-ε^q the introduced bias in the final image due to the correction of ε gets alleviated, i.e., the higher the correction of ε is, the harder the chance that the geometry related to the node would result as being visible.

The only source that could hinder the visual quality in CHLOD-ε^q is when there is an important number of nodes that need to be refined once they have been rendered in the same frame (see Section 5). Fortunately, we have found this value to be negligible when using our CHLOD-ε^q algorithm: the average number of drawn nodes that need further refinement once they have been rendered for the three scenes are only 0.9%, 1.19% and 1.85%, respectively (see Table 1).

7. Conclusions

We have introduced a coherent HLOD culling algorithm that employs a novel metric to perform the refinement of a HLOD-based system that takes into account visibility information. Our approach supports both polygon-based as well as point-based HLODs. Our main contributions are:

- Improved performance with the same visual quality: we are able to render less primitives with minimal loss in image quality.
- Full use of the result of the HOQ: our metric takes full advantage of the information gathered in HOQs.
- Full use of the spatial and temporal coherency inherent to hierarchical representations to avoid latency times in HOQs meanwhile using the introduced metric to perform the hierarchical refinement.
- Straightforward implementation.

For future work we are researching how to integrate other view-dependent parameters (different from the distance to the viewpoint) into the refinement criterion. Regarding our approach to perform the coherent HLOD culling, we plan to investigate means to better anticipate when the hierarchical refinement should stop.

Acknowledgements

We would like to thank the Stanford Graphics Group for the 3D models they provided. We also thank Eduardo Romero at the *Universidad Nacional de Colombia* for valuable criticisms. We are grateful to Gilles Debunne and Mario Botsch for their assistance with the *libQGLViewer* software library and the *OpenMesh* software library, repectively. This work is funded by a *Fundacion Carolina* grant.

References

[ASVNB00] ANDÚJAR C., SAONA-VÁZQUEZ C., NAVAZO I., BRUNET P.: Integrating occlusion culling

	scene 1			scene 2			scene 3		
Configuration	FPS	DN	RARN	FPS	DN	RARN	FPS	DN	RARN
CHLOD-ε^q	72.3	94.82	0.85	40.56	166.54	1.97	62.49	159.32	2.95
CHLOD-ε	61.57	110.69	-	34.99	193.92	-	47.69	200.9	-
SW-ε^q	53.23	75.71	-	28.48	141.25	-	43.27	103.8	-
SW-ε	40.39	107.24	-	22.24	180.56	-	28.19	199.48	-

Table 1: *Statistics for the test scenes. FPS are the number of frames per second, DN are the number of drawn nodes, and RARN is the number of nodes that once rendered in a given frame need further refinement within the same frame (only for CHLOD-ε^q). All values are average over all frames.*

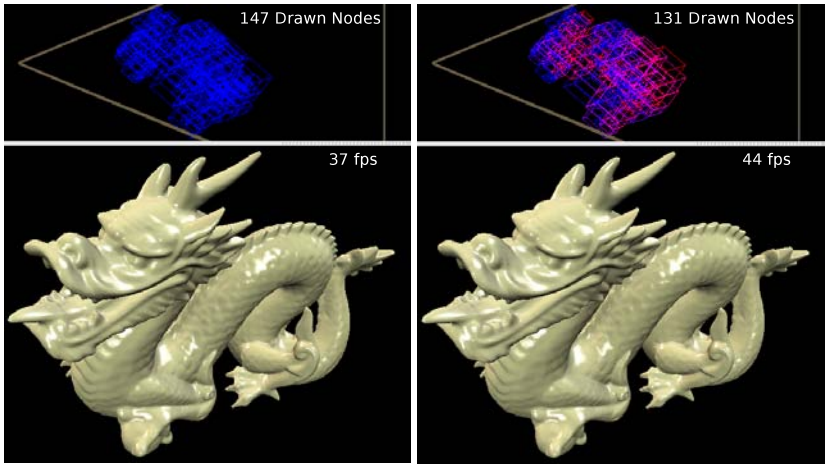

Figure 8: *Scene 1 (low depth complexity): Selected frame of the visualization sequence. The top images correspond to the view-frustum view of the bounding boxes of the nodes selected to be drawn.*

with levels of detail through hardly-visible sets. *Computer Graphics Forum (Proceedings of Eurographics '00)*, 3 (2000), 499–506.

[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.

[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.

[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).

[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.

[GBBK04] GUTHE M., BORODIN P., BALÁZS Á., KLEIN R.: Real-time appearance preserving out-of-core rendering with shadows. In *Rendering Techniques* (2004), pp. 69–80.

[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.

[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.

[HGJ03] HA H., GREGORSKI B., JOY K. I.: Out-of-core interactive display of large meshes using an oriented bounding box-based harware depth query. *Com-*

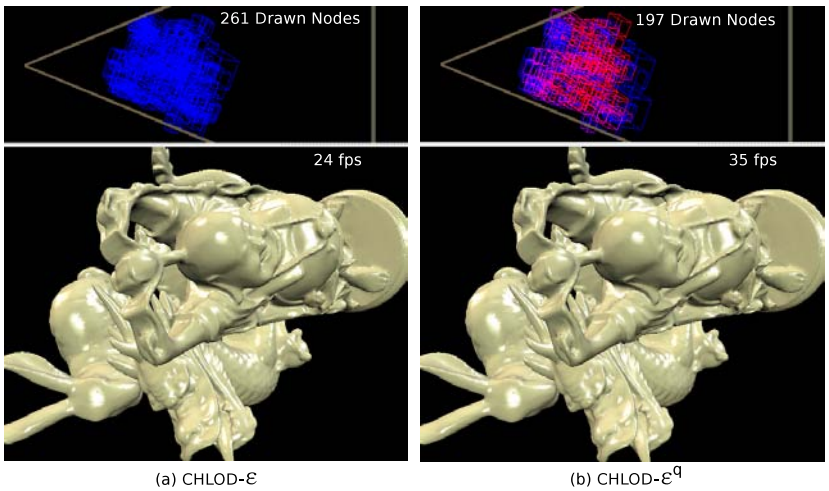

Figure 9: *Scene 2 (middle depth complexity): Selected frame of the visualization sequence. The top images correspond to the view-frustum view of the bounding boxes of the nodes selected to be drawn.*

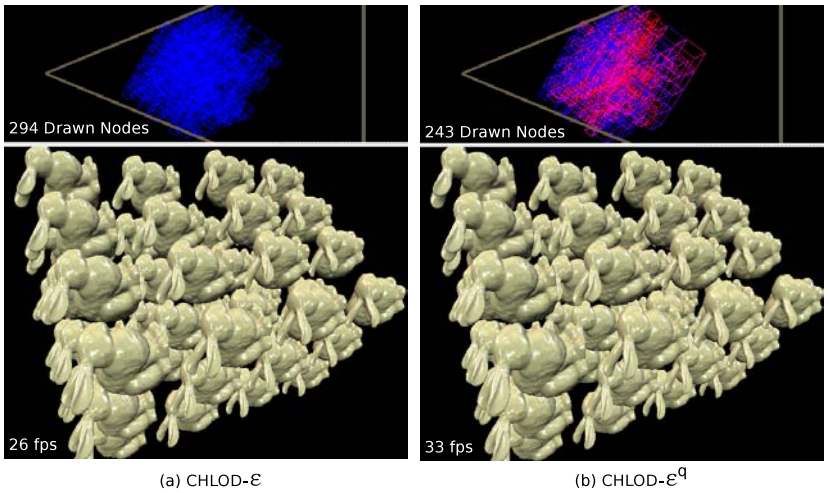

Figure 10: *Scene 3 (high depth complexity): Selected frame of the visualization sequence. The top images correspond to the view-frustum view of the bounding boxes of the nodes selected to be drawn.*

puter Graphics Forum 22, 3 (2003).

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

[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.

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

[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.

[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.