Optimized HLOD Refinement Driven by Hardware Occlusion Queries

Jean Pierre Charalambos[1,2], Jiří Bittner[3], Michael Wimmer[1], and Eduardo Romero[2]

[1]Vienna University of Technology
[2]National University of Colombia
[3]Czech Technical University in Prague

Abstract. We present a new method for integrating hierarchical levels of detail (HLOD) with occlusion culling. The algorithm refines the HLOD hierarchy using geometric criteria as well as the occlusion information. For the refinement we use a simple model which takes into account the possible distribution of the visible pixels. The traversal of the HLOD hierarchy is optimized by a new algorithm which uses spatial and temporal coherence of visibility. We predict the HLOD refinement condition for the current frame based on the results from the last frame. This allows an efficient update of the front of termination nodes as well as an efficient scheduling of hardware occlusion queries. Compared to previous approaches, the new method improves on speed as well as image quality. The results indicate that the method is very close to the optimal scheduling of occlussion queries for driving the HLOD refinement.

1 Introduction

Interactive visualization of complex models comprising millions of polygons is one of the fundamental problems in computer graphics. In order to achieve interactive rendering of such models, the amount of processed data has to be substantially reduced. Level-of-detail methods allow aggressive reduction of the amount of data sent to the GPU at the expense of sacrificing image quality. Particularly, hierarchical level-of-detail (HLOD) methods proved capable for interactive visualization 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 [1].

An orthogonal approach of reducing the amount of rendered primitives is occlusion culling [2]. Occlusion culling methods aim to quickly cull the invisible part of the model and render only its visible part. In order to achieve this task, most recent methods employ hardware occlusion queries (HOQs) [3,4].

The effects of HLODs and occlusion culling can be effectively combined [5,6]. Moreover, it was shown that HOQs can also be used to drive the HLOD refinement [7]. 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 a HOQ and the availability of its result, the direct use of HOQs for refinement criteria causes CPU stalls and GPU starvation.

In this paper we introduce a novel traversal algorithm for HLOD refinement driven by HOQs. The algorithm minimizes CPU stalls and GPU starvation by predicting the HLOD refinement conditions using spatio-temporal coherence of visibility. As a result, it provides substantial speedup over previous methods while maintaining comparable image quality.

2 Related Work

Rendering very large models has received serious attention in recent years. Among other techniques, HLOD-based methods proved efficient for handling these datasets. HLOD systems either use polygonal representation of LODs [8], point-based representations [6], or a combination of both [9]. They commonly employ a *screen space error* (SSE) to drive the HLOD refinement [8,5]. As an alternative, Charalambos [7] proposed the *virtual multiresolution screen space error* (VMSSE) which also considers the degree of occlusion.

Occlusion culling methods are another popular technique for handling large models [2]. Recent methods employ HOQs mainly due to their efficiency and simplicity. The main goal of the recent techniques is to cope with the latency between issuing the query and availability of its results as well as reducing the number of issued queries [3,4].

The combination of occlusion culling and discrete LODs was addressed by Andújar *et al.* [10], who introduced the concept of *hardly visible sets.* In the context of view-dependent LOD El-Sana *et al.* [11] proposed *cell solidity values* with the same purpose in mind. Gobetti *et al.* [6] presented a method for integrating hardware occlusion queries into an HLOD-based system. This method copes with the query latency by using temporal coherence as proposed by Bittner *et al.* [3]. It does not, however, exploit the results of HOQs for driving the HLOD refinement.

Charalambos proposed a different method which also integrates occlusion culling into an HLOD-based system, and it additionally exploits the results of HOQs to drive HLOD refinement using the VMSSE [12]. In order to minimize the effect of latency of HOQs, this technique performs several simplifications, which may degrade the visual quality as well as the performance of the resulting algorithm. Moreover, these effects are more noticeable in the case of scenes with higher depth complexities [12]. In this paper we focus on these problems and propose an optimized method for efficiently scheduling the occlusion queries, which improves both the running time as well as the visual quality compared to the previous methods.

3 Integrating HLODs and HOQs

This section overviews the problem of integrating HLODs and HOQs, and briefly describes previous approaches. We first describe the visibility-based traversal of an HLOD hierarchy and then we focus on the computation of the visibility-based HLOD refinement criterion.

3.1 Visibility-Based HLOD Traversal

The HLOD algorithm recursively traverses the hierarchy starting at the root node. At every traversed node it evaluates a *refinement condition*. If this condition indicates that the node should be refined, the traversal continues with the children of the node. Otherwise, the traversal terminates and the LOD associated with the node is rendered. The set of nodes where the refinement stops forms the *front of termination nodes*. The refinement condition commonly uses the SSE, which corresponds to the projection of the model space error onto the screen [8,5]. The SSE for the given node is compared to a user-defined threshold given in pixels (τ), known as *pixels-of-error* [5].

The HLOD algorithm can be improved by integrating occlusion culling into the traversal of the hierarchy. Firstly, we can cull nodes which are completely invisible [6]. Secondly, for visible nodes we can use the results of HOQs in the refinement condition. This can be achieved by using the VMSSE [12,7], which modulates the SSE using the result of the HOQ.

By using the VMSSE we can reduce the number of rendered primitives. However, the HLOD refinement becomes dependent on the result of the HOQ, which is not readily available at the time when the refinement criterion is evaluated, due to the latency between issuing the HOQ and the availability of the result. Waiting for the result of the query would cause a CPU stall and in turn a GPU starvation. A way to cope with this problem is to predict the result of the HOQ and use the predicted value in the evaluation of the refinement condition. If the prediction were perfectly correct, we would only stop the refinement where it would have been stopped if the result of the query had already been available. A simple prediction method which uses the processing order of the nodes has been proposed by Charalambos [12]. However, this prediction is rather simplistic and inaccurate, leading to two main drawbacks: (1) When the prediction is too conservative, more nodes get refined and in turn more primitives are rendered, (2) when the prediction is too aggressive, the node is rendered, but the same node is then also refined when the result of the query is available. That means that the refined children are rendered together with their parent, which can cause visual artifacts.

In order to cope with this problem, our new algorithm predicts the refinement condition based on the criteria determined in the previous frame. The prediction is used to decide whether to immediately refine the node or stop the refinement, or delay the decision till the result of the query becomes available.

3.2 Virtual multiresolution screen space error

The classical HLOD refinement criterion is based on the SSE which bounds the error of projecting the simplified geometry onto the screen. The SSE only considers the distance of the object from the viewpoint, and disregards the occlusion of the object. However if the object is largely occluded, we most likely do not perceive subtle details in the remaining visible part of the object. This suggests that the computed SSE could be reduced if occlusion is considered [10,7].

In this paper we use a modification of the VMSSE proposed by Charalambos [7]. This method evaluates the *relative visibility* μ, which is the ratio of unoccluded pixels to the number of pixels an object projects to. The number of unoccluded pixels is computed by an HOQ, while the number of projected pixels is calculated analytically.

The method presented in [7] used the relative visibility μ to scale the SSE linearly. However a simple linear scaling of SSE might lead to visual artifacts because it does not take the possible distributions of visible pixels into account: For larger relative visibility, it is likely that larger *visibility windows* appear, which make errors due to decreased LOD levels more perceptible. A proper way for handling this issue would be to analyze the distribution of visible pixels in conjunction with visual masking analysis. However both techniques would be too costly for evaluating the HLOD refinement criterion in real time. A simple model which aims to reflect the possible clustering of visible pixels was proposed in [13]. It assumes that the likelihood of larger visibility windows to appear is proportional to relative visibility μ, and therefore modulates the SSE using a sigmoid *bias* function $B(\mu)$ (see Figure 1-a):

$$VMSSE = B_{s,t}(\mu) \cdot SSE, \tag{1}$$

where the bias $B_{s,t}(\mu)$ is a function of μ with user-defined parameters s and t. $t \in [0, 1]$ is the *movable turn over* point and $s \in [0, 1]$ is the *smoothness*. The closer the movable turn over point t is to 0, the more conservative the SSE modulation is, i.e. the SSE will be modulated strongly just if only a few unoccluded pixels exist. The *smoothness* parameter s linearly interpolates between μ and the unsmoothed sigmoid. Particularly, if $s = 0$ then the sigmoid function becomes equal to relative visibility (μ).

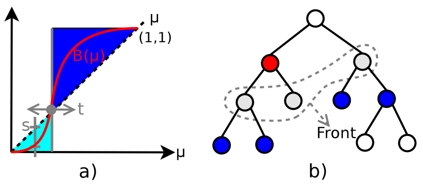

Fig. 1. a) Sigmoid bias function $B_{s,t}(\mu)$ used to compute the VMSSE. In the cyan region ($\mu \leq t$) the function provides aggressive scaling, while in the blue region ($\mu > t$) the scaling is rather conservative. The s value is used to control the smoothness of the curve. b) Candidates for update of the front of termination nodes. In red we show nodes for which refinement has stopped for all its children. In blue we show nodes one level below the termination nodes

4 Coherent HLOD culling algorithm

This section describes the proposed HLOD culling algorithm. We first describe the complete traversal algorithm and then we focus on its crucial part, which predicts the HLOD refinement condition by using temporal coherence.

4.1 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 the HLOD refinement. 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 result of occlusion is costly as it stalls the CPU and in turn causes GPU starvation. 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. The exceptions are 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: we start the traversal of the HLOD hierarchy at the root node. At each traversed node, we predict the refinement condition for the node based on the refinement conditions and the results of occlusion queries from the previous frame (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 is known.

In case (1), the children of the node are traversed by putting them in the priority queue. In case (2) and (3), we issue a HOQ for the node and put it in the query queue. In case (2), we also render the geometry associated with the node immediately and stop the refinement. In case (3), we delay the processing of the node until the result of the HOQ 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 in the priority queue.

The new traversal algorithm is outlined in Figure 2. Note that the differences to the traversal algorithm of Gobetti *et al.* [6] were colorised. Also note that the function $CalcSSE()$ computes the node SSE [8], which is a quick operation.

4.2 Predicting the HLOD refinement condition

The crucial part of the new method is the prediction of the refinement condition based on the relative visibility. As stated in the previous section, the prediction suggests either refine, stop, or delay. Let us first analyze the consequences of these actions for the traversal algorithm:

```
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);
            $\mu \leftarrow visiblePixels/BBoxPixels(node)$;
            node.bias ← $B_{s,t}(\mu)$;
            VMSSE ← node.bias*CalcSSE(node);
            node.stopRefinement ← VMSSE $\leq \tau$;
            Traverse(node);
    if ¬PriorityQueue.Empty() then
        node←PriorityQueue.Dequeue();
        if InsideViewFrustum(node) then
            stopMode←PredictRefinement(node);
            node.stopRefinement ← ¬(stopMode=Refine);
            node.visible ← false;
            node.lastVisited ← frameID;
            if node.stopRefinement then
                IssueOclussionQuery(node);
                QueryQueue.Enqueue(node);
            if ¬(stopMode=Delay) then
                Traverse(node);

Traverse(node);
if node.stopRefinement then
    Render(node);
else
    PriorityQueue.EnqueueChildren(node);
```

Fig. 2. Coherent HLOD Culling

– *Refine*. The children of the node are traversed immediately. No HOQ nor 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) we end up rendering more geometry than necessary.
– *Stop*. An HOQ is issued and the node is rendered immediately. When the result of the query is available we compute the VMSEE of the current node. If the prediction was *too aggressive*, we have to continue by traversing the children of the node. Note that in this case we end up rendering geometry of (some) child nodes over the geometry of the parent node, which increases the rendering cost and can also lead to visual artifacts.
– *Delay*. In this case wait for the result of the query to decide on the refinement condition. Thus for a node which was delayed and for which refinement

should have stopped we have induced a latency in passing its geometry to the GPU.

From this analysis we designed a prediction technique which aims to minimize the number of incorrect predictions by assuming coherence of the front of termination nodes. It primarily aims to predict either *refine* or *stop* conditions with high accuracy. If we expect a stop condition, but with lower confidence, the predictor returns *delay*. We also return *delay* for nodes which have been previously invisible and thus we expect the refinement will terminate without rendering the geometry of these nodes. The main idea of the prediction is to estimate the VMSSE by combining the SSE of the current frame with the *cached* bias values from the previous frame:

$$VMSSE_i^{est} = SSE_i * bias_{i-1} \tag{2}$$

The prediction works as follows (see also the pseudocode in Figure 3):

- *Node was invisible in the previous frame.* In this case the prediction returns *delay*.
- *Refinement stopped for the node in the previous frame.* We calculate $VMSSE^{est}$, 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.* In this case, the node is a good candidate for pulling up the termination front (see the red node in Figure 1-b). We verify this by first checking whether refinement for all children would still stop in the current frame based on their estimations $VMSSE^{est}$. If any of these indicates continue refinement, then the predictor returns *refine*. Otherwise, since the node itself doesn't have a cached bias value, we approximate $bias_{i-1}$ for the node by taking the average cached bias from all children. If the resulting $VMSSE^{est}$ is above the threshold, the predictor returns *refine*. Otherwise the predictor returns *delay*.

5 Results and discussion

We implemented the proposed algorithm using C++ and OpenGL under Linux. The HLODs use an octree with a single discrete LOD per node consisting of about 2000 triangles. We used the quadric error metric [8,5] to construct the HLODs and to derive the model space errors. For efficient caching of the geometry on the GPU´, we employed vertex buffer objects. The measurements were performed on two scenes with different depth complexities. For all tests we used a resolution of 640*480 pixels and the error threshold $\tau = 1$. The tests were evaluated on a PC with Intel-Core 2 Duo (2.4GHz) and nVidia GeForce 8800 GTX.

```
if $\neg$(node.lastVisited=frameID-1) $\vee$ node.visible=false then
    return Delay;
if $\neg$node.stopRefinement then
    candidateToShiftUp $\leftarrow$ true;
    forall child $\in$ node.children do
        if $\neg$child.stopRefinement $\vee$ $\neg$(child.bias*CalcSSE(child) $\leq \tau$) then
            candidateToShiftUp $\leftarrow$ false;
    if candidateToShiftUp then
        if AvgBias(node.children)*CalcSSE(node) $\leq \tau$ then
            return Delay;
else if node.bias*CalcSSE(node) $\leq \tau$ then
    return Stop;
return Refine;
```

Fig. 3. Function $PredictRefinement(node)$

5.1 Tests

We have used two scenes with middle and high depth complexities, respectively named as *scene 1* and *scene 2* (see Figure 5). For each scene we have designed a session representing typical inspection tasks. Depending on the traversal algorithm and the metric used to refine the hierarchy, we have evaluated the following scenarios:

- *Bool*: the hierarchy was traversed with the coherent culling algorithm version of Gobetti *et al.,* [6] (see Section 2). For this method we used the SSE metric for HLOD refinement, i.e., HOQs were used as if their result were boolean. Note that this configuration gives the ideal image quality for our tests.
- $SW(B)$: the hierarchy was traversed with the hierarchical *stop-and-wait* method referenced in Bittner *et al.* [3], using VMSSE to refine the hierarchy. (B) stands for using the $B_{s,t}(\mu)$ bias to compute VMSSE. Note that this configuration gives the ideal number of nodes to be drawn when using VMSSEs.
- $Simp(B)$: the hierarchy was traversed with the coherent HLOD culling algorithm proposed in [12] using VMSSE to refine the hierarchy.
- $Coh(B)$: the hierarchy was traversed with our new coherent HLOD culling algorithm (see Section 4) using VMSSE to refine the hierarchy.

5.2 Speedup

Figure 4 shows the whole sequence of drawn nodes together with the frame rates for the two scenes. All scene statistics have been summarized in Table 1.

It can be seen that our new prediction algorithm significantly reduces the number of drawn nodes compared to both *Bool* and $Simp(B)$, which also translates directly into higher framerates. The main reason for the reduction of drawn

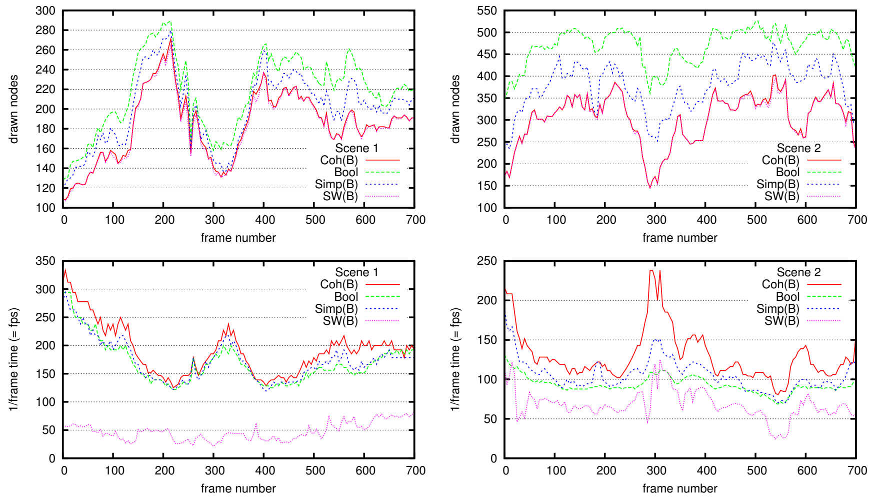

Fig. 4. Drawn nodes and frame rates for the test scenes

nodes with respect to *Bool* was the use of VMSSE instead of SSE within refinement conditions. The speedup obtained by using the VMSSE is clearer perceived in scenes with higher depth complexity in which the savings in number of drawn nodes are greater. The reduction of number of drawn nodes with respect to $Simp(B)$ follows from the tighter approximation of the ideal number of nodes to be drawn, which relies on the method to approximate the VMSSE *bias*. The comparison to $SW(B)$ shows that our new approach is within 1% from the ideal number of nodes to be drawn, whereas the $Simp(B)$ method draws up to 25% more nodes. It is also worth noting that the precision of $Simp(B)$ depends on the scene depth complexity: it behaves poorly in scenes with higher depth com-

Table 1. Statistics for the test scenes. 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 and D is the number of nodes delayed for rendering. FPS is the number of frames per second, and Speedup is the relative speedup of $Coh(B)$ with respect to the given method. All presented values are averages over all frames

Stats	*scene 1* *full resolution model* $\approx$ *5M* $\triangle$*'s* *number of HLOD nodes* $\approx$ *12k*					*scene 2* *full resolution model* $\approx$ *12M* $\triangle$*'s* *number of HLOD nodes* $\approx$ *21k*				
Scenario	DN	RARN	D	FPS	Speedup	DN	RARN	D	FPS	Speedup
Bool	217	-	-	169.9	1.13	468.5	-	-	92.9	1.40
$SW(B)$	181.4	-	-	47.3	4.04	302.1	-	-	68.2	1.91
$Simp(B)$	200	5	-	173.6	1.10	377.8	4.3	-	105.4	1.23
$Coh(B)$	183.3	0.1	4.7	191.6	-	303.9	0.4	4.2	130.2	-

plexities (scene 2), whereas the new method handles this types of scenes very well.

The only source for visual artifacts inherent in the traversal algorithm (as opposed to the VMSSE calculation) is the case when there are some nodes that need to be refined even though they have already been rendered in the same frame (RARN). Fortunately, unlike for $Simp(B)$, for $Coh(B)$ we have found this value to be always negligible. The reason is that our delay strategy for the nodes where the stop refinement condition is predicted to be shifted up effectively minimizes RARN. Additionally, the RARN reduction is achieved without hindering performance: the average number of nodes that are delayed for rendering out of the total number of drawn nodes for the two scenes are only 2.56% and 1.39%, respectively.

5.3 Image quality

We have measured the difference between the final ideal image obtained by $Bool$ and the one obtained by $Coh(B)$ by randomly selecting 20 frames of each inspection sequence and computing the peak signal-to-noise ratio difference ($psnr$). This measure has been traditionally used as an estimator of the distortion introduced by compression algorithms [14] and corresponds to the ratio of the power of a particular signal and the corrupting noise. The average and standard deviation $psnr$ values (luminance (l) and chrominance (C_b and C_r) components of the colors, repectively) for the 20 frames are: $l = 42.84 \pm 3.22$, $C_b = 67.04 \pm 3.68$ and $C_r = 56.2 \pm 3.65$ for scene 1; and $l = 35.51 \pm 1.4$, $C_b = 59.34 \pm 1.55$ and $C_r = 48.69 \pm 1.58$ for scene 2. The fact that $psnr > 30$ for all color components indicates that the proposed method practically does not alter the final image quality [14].

To emphasize the influence on image quality caused by $Coh(B)$, for each node in the front we have colored the geometry from blue to magenta to red according to the severity level of the modulation introduced by the bias: 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 Figure 5). Note that the use of $Coh(B)$ attenuates the bias, i.e., the stronger the bias is, the less likely it is that the node is actually visible (see the last column in Figure 5). On the other hand, whilst in $Simp(B)$ the appearance of visual artifacts is common, in $Coh(B)$ we practically eliminated this problem (see the first two columns in Figure 5 and also the accompanying video).

5.4 Summary of results

The results show that the proposed method is superior with respect to the previous state-of-the-art methods both in the framerate as well as in the image quality obtained. In comparison to the method of Gobetti et al. [6] ($Bool$), the reference solution for image quality measurements, we obtain a speedup of $1.13 - 1.4$,

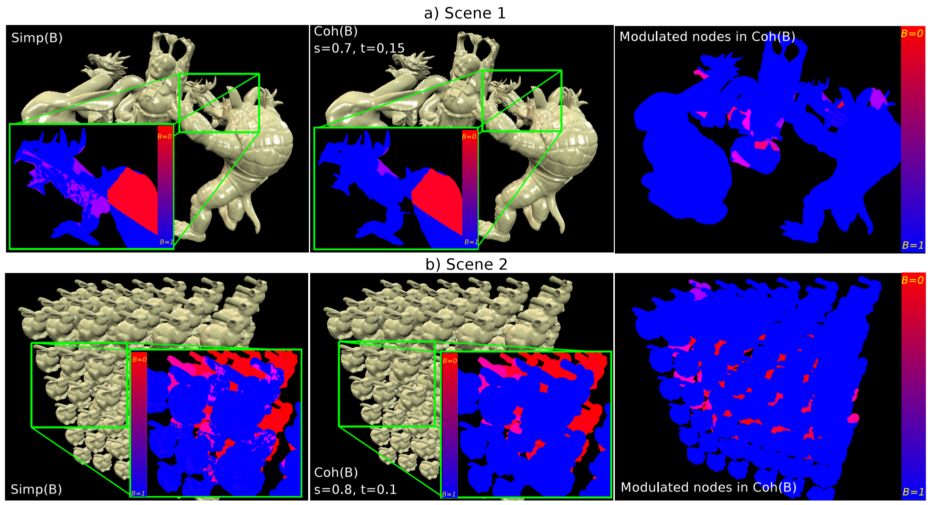

Fig. 5. Test scenes: selected frame of the visualization sequences when using $Simp(B)$ and $Coh(B)$. The last column corresponds to a visualization (from the user viewpoint) of the introduced modulation of the nodes selected to be drawn due to the VMSSE *bias* $(B_{s,t}(\mu))$ in $Coh(B)$. The small frames in the first two columns correspond to a detail in the scenes to show the possible appearance of visual artifacts due to RARN. Models courtesy of Standford Graphics Group.

which is significant, while the visual quality of our method does not incur a perceivable penalty. In comparison to the method of Charalambos [7] $(Simp(B))$, the speedup is about $1.1 - 1.2$. However, that technique shows frequent visual artifacts which might not be acceptable in walkthrough or inspection applications, and which the new method avoids. Therefore, the proposed solution is qualitatively superior while still managing to be faster.

6 Conclusions

We have presented an algorithm to integrate HLOD and occlusion culling. The main contribution is that the algorithm closely approaches the optimal set of primitives to render while avoiding visual artifacts exhibited by previous methods. We demonstrate significantly improved performance when compared to previous approaches. The main idea is to exploit temporal coherence and make use of the visibility information returned by hardware occlusion queries to determine the simplification degree of nodes. The method also has a straightforward implementation.

Acknowledgments

This work has been supported by the following resources: an Ernst Mach grant from the Austrian Exchange Service – Agency for International Cooperation in Education and Research (ÖAD), Academic Cooperation and Mobility Unit (ACM); the EU project no. IST-2-004363 (GameTools); and by the Ministry of Education, Youth and Sports of the Czech Republic under the research program LC-06008 (Center for Computer Graphics).

References

1. Guthe, M., Borodin, P., Balázs, Á., Klein, R.: Real-time appearance preserving out-of-core rendering with shadows. In: Rendering Techniques. (2004) 69–80
2. 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)
3. Bittner, J., Wimmer, M., Piringer, H., Purgathofer, W.: Coherent hierarchical culling: Hardware occlusion queries made useful. Computer Graphics Forum **23** (2004) 615–624
4. Guthe, M., Balázs, Á., Klein, R.: Near optimal hierarchical culling: Performance driven use of hardware occlusion queries. In Akenine-Möller, T., Heidrich, W., eds.: Eurographics Symp. on Rendering 2006, The Eurographics Association (2006)
5. Yoon, S.E., Salomon, B., Gayle, R., Manocha, D.: Quick-vdr: Interactive view-dependent rendering of massive models. In: VIS '04: Conference Proc., Washington, DC, USA, IEEE Computer Society (2004) 131–138
6. Gobbetti, E., Marton, F.: Far Voxels – a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. ACM Trans. on Graphics **24** (2005) 878–885 Proc. SIGGRAPH 2005.
7. Charalambos, J.P.: Virtual multiresolution screen space errors: Hierarchical level-of-detail (hlod) refinement through hardware occlusion queries. In: GMAI, IEEE Computer Society (2006) 221–227
8. 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 Trans. on Graphics **23** (2004) Proc. SIGGRAPH 2004.
9. 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). (2003)
10. 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) (2000) 499–506
11. El-Sana, J., Sokolovsky, N., Silva, C.T.: Integrating occlusion culling with view-dependent rendering. In: VIS '01: Conference Proc. (2001) 371 – 378
12. Charalambos, J.P.: Coherent hierarchical level-of-detail (hlod) refinement through hardware occlusion queries. In: SIACG 2006 - Ibero-American Symp. on Computer Graphics, Univ. Santiago de Compostela, Spain, Eurographics Association (2006)
13. Anonymous: A visibility information-based metric for performing the refinement of a hlod-based system. Submitted to Computer Graphics Forum (2007)
14. Gonzalez, Woods: Digital Image Processing, 2nd edition. Prentice Hall (2001)