Pacific Graphics (2014)
J. Keyser, Y. J. Kim, and P. Wonka (Editors)

Short Papers

Integrating Occlusion Culling into LOD on GPU

Chao Peng †1
[1]Computer Science Department
University of Alabama in Huntsville, USA
chao.peng@uah.edu

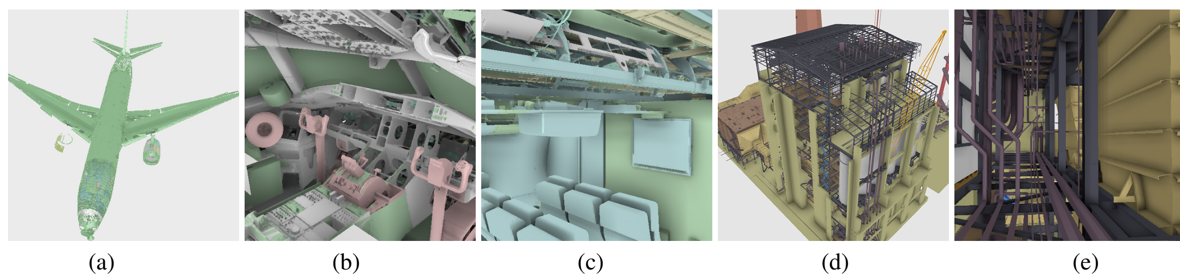

(a) (b) (c) (d) (e)

Figure 1: ***Complex 3D models rendered by our approach.*** *(a)-(c) are the rendered images of Boeing 777 model (332M triangles); (d) and (e) are the rendered images of Power Plant model (12M triangles).*

Abstract

Real-time rendering of complex 3D models is still a very challenging task. Recently, GPU-based level-of-detail (LOD) approaches have been proposed to fast decrease the complexity of a 3D model, but applying only LOD approaches is usually not sufficient to achieve highly interactive rendering rate for the complex model that contains hundreds of millions of triangles. Visibility culling, especially occlusion culling, needs to be introduced to further reduce the amount of triangles being rendered at each frame. In this paper, we present a novel rendering approach that seamlessly integrates occlusion culling with the LOD approach in a unified scheme towards the GPU architecture. The result shows that the integration significantly reduces the complexity of the 3D model and satisfies the demands of both memory efficiency and performance.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms

1. Introduction

As data complexity continues to increase due to the fundamental advances in modeling and simulation technologies, a complex 3D model may need several gigabytes in storage, and contains millions, even hundreds of millions of polygon primitives (see Figure 1). Rendering such complex models is computationally intensive. Recently, massive parallelism on GPU has become a major trend for high-performance applications. However, the requirement to interactively render gigabyte-scale models usually overburdens the computational power and memory capacity of the GPU. The size of GPU memory is limited. A model consuming several gigabytes can not fit into it. The common solution is to select a portion of the data and generate a simplified version. Towards this idea, parallel mesh simplification algorithms, such as [HSH09, DMG10, PC12]), have been proposed. But, without considering view-parameters, the occluded objects are undesirably rendered in high levels of details and consumes GPU resources. The wasted computing power and

† Chairman Eurographics Publications Board

© The Eurographics Association 2014.

DOI: 10.2312/pgs.20141246

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

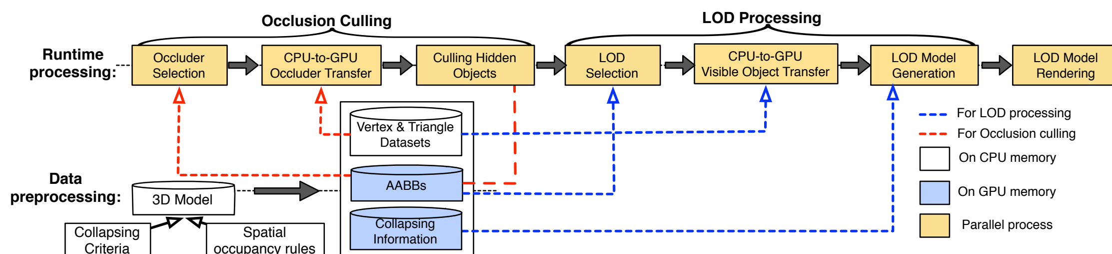


Figure 2: ***The overview of our approach.***

memory on rendering those occluded objects will definitely hurt overall performance and visual quality.

In this paper, we integrates occlusion culling into LOD processing towards GPU architectures. The hidden objects are removed by a conservative occlusion culling algorithm with a novel metric of dynamic occluder selection. GPU memory will not wasted on those hidden objects. Simplified versions will be generated only for visible objects.

2. Related Works

In this section, we discuss some previous works focusing on mesh simplification, occlusion culling and their integration.

Traditional simplification algorithms were based on a sequence of topological modifications, such as edge collapsing [Hop97, GH97]. However, collapsing edges one-by-one could be very time-consuming on complex models. To speed up the performance, GPU-based parallel implementations have been proposed. [HSH09] introduced a GPU-based approach for view-dependent Progressive Meshes, where vertex dependencies were not fully considered during a cascade of vertex splitting events. [DMG10] encoded the dependency information of Progressive Meshes into a GPU-friendly compact data structure. [PC12] eliminated the dependencies by simply using an array structure and supported triangle-level parallelism.

Occlusion culling aims to remove the hidden objects. Hierarchical Z-buffer (HZB) introduced in [GKM93] is an algorithm that uses an image-space Z-pyramid to quickly reject the rendering of hidden objects. [GSYM03] used a dual-GPU occlusion-switch approach to overcome the performance issues. But an additional latency was introduced when exchanging data between GPUs. [BWPP04] proposed an optimized version of hardware occlusion queries to improve the culling efficiency and performance.

To integrate them, [LT99] pre-computed occlusion information to guide the refinement process in the LOD algorithm. The authors replaced occlusive polygon set with a simplified virtual occluder. But the simplified occluders do not have correct silhouettes, and they may not be able to produce accurate depth information for the culling. [ASVNB00] used Visibility Octree to estimate the degree of visibility of each object, which was also contributed to the LOD selection in the integration. [YSM03] decomposed the 3D scene into a cluster hierarchy, where the simplification algorithm was applied to the set of visible clusters. [GM05] represented the data with a volume hierarchy, by which their approach tightly integrated the algorithms of LOD, culling and out-of-core for massive model rendering. [DG12] proposed the integration on GPU with a bounding volume hierarchy. The authors successfully removed the data-dependency to support the GPU parallel implementation, but the bounding volume hierarchy required a significant amount of memory. [SKK*14] used the integration for city generation and rendering. Since cities usually do not have spacial irregular shapes, occlusion culling and LOD approaches performed well on city models than other type of models.

3. Overview

We propose two geometric processing stages, data preprocessing and runtime processing, as illustrated in Figure 2. In preprocessing, we perform a sequence of edge-collapsing operations to simplify the input model. The order of these operations is used to re-arrange the storage of vertices and triangles. Also, to better prepare for occlusion culling, we examine the qualification of each object as an occluder by evaluating its spatial occupancy. We also generate Axis-Aligned Bounding Boxes (AABBs) of the objects.

During the runtime, occlusion culling and LOD processing components are performed based upon a series of parallel processing steps. we select a set of adaptive occluders, transfer them to GPU, and rasterize them into a Z-depth image. The objects hidden behind the occluders are then eliminated by testing against the depth image. After that, the remaining objects are passed through the component of LOD processing, where each object's geometric complexity is determined to be used for reforming the object into a new shape. At the end, the reformed objects are rendered with OpenGL Vertex Buffer Objects (VBO).

© The Eurographics Association 2014.

4. LOD Selection

Determining the desired level of detail (or the desired geometric complexity) of the objects is a very important step in LOD algorithms, which is knownn as *LOD Selection*. Conceptually, an object can be rendered at any level of detail. But because GPU memory is limited, the total number of polygon primitives must be budgeted based on the memory capability. [PC12] solved the LOD selection problem with a discrete optimization solution. In that paper, the detail level of an object is determined by the size of the screen region occupied by an object. The larger size of the region, the higher level the object's detail should be. However, objects may have dramatically different shapes with widely varying spatial ratios. It is possible that a far-away object, which deserves a lower level of detail, has a much larger projected area than a closer object. In addition, objects usually come with different number of triangles due to the nature of original design. An object that has more triangles should rationally be approximated with a higher detail level than those with fewer triangles, though the former one may be farther away from the viewpoint. By considering all these aspects, We re-evaluate the LOD selection problem and provide a closed-form expression.

LOD Selection Metric. The desired level of detail of the ith object is represented with a pair of vertex count and triangle count. We denote them as vc_i and tc_i, respectively. We compute vc_i using Equation 1.

$$vc_i = N\frac{w_i^{\frac{1}{\alpha}}}{\sum_{i=1}^{m} w_i^{\frac{1}{\alpha}}}, \text{where } w_i = \beta\frac{A_i}{D_i}P_i^{\beta}, \beta = \alpha - 1 \quad (1)$$

N is the user-defined maximal count. vc_i is computed out of total m objects. A_i denotes the projected area of the object on image plane. To compute A_i efficiently, we estimate it using the area of the screen bounding rectangle of the projected axis-aligned bounding box (AABB) of the object. The exponent, $\frac{1}{\alpha}$, is a factor aiming to estimate the object's contributions for model perception. D_i is the shortest Z-depth distance from the corner points of the AABB. P_i is the number of available vertices of the object. Then, tc_i can be easily retrieved from the information recorded during the edge-collapsing operation in preprocessing.

Pixel Error Threshold. No matter how large or complex an object is, the object's shape is scan-converted into pixels. At a position long-distance to the viewpoint, a very large object may be projected to a very small region of the screen (e.g. less than one pixel), so that the object might not be captured by people's visual perception. Based on this nature, we introduce a *Pixel Error Threshold* (PET) as a complementary criteria for our metric of *LOD selection*. If A_i in Equation 1 is below a given PET, vc_i is set to zero.

© The Eurographics Association 2014.

5. Parallel Occlusion Culling

In Equation 1, N is an important factor that impacts on overall performance and visual quality. N tells how many vertices and triangles will be processed by the GPU. We can decrease the value of N to ensure a desired performance, but a small N will result in the loss of visual quality. One of the reasons is that the hidden objects obtain cuts from N. It would be more reasonable that those cuts are assigned to the visible objects to increase their detail levels.

In this section, we introduce a novel parallel approach for occlusion culling that rejects the hidden objects before the step of LOD budget allocation.

5.1. Preprocessing

In a model, not all objects are suitable to be occluders. Most existing systems use simple criteria to examine an object's qualification, such as the size of the object's AABB and/or the number of triangles it has. But, in a complex model, some objects may be irregularly shaped (e.g., casually curved long wires in the Boeing 777 airplane model), which should not be qualified as occluders at any viewpoints. We use a spatial occupancy criteria to determine the qualification of an object by measuring its compactness in local space. A tight Object-Orientated Bounding Box(OOBB) is calculated for each object. Equation 2 returns the value of compactness that indicates how well the object fills the OOBB's space.

$$compactness = \frac{A_x\frac{P_x}{R_x} + A_y\frac{P_y}{R_y} + A_z\frac{P_z}{R_z}}{A_x + A_y + A_z}. \quad (2)$$

A is the orthogonally projected area of OOBB along its local axises; P is the number of pixels occupied by the object on the corresponding projection; R is the total number of pixels occupied by the OOBB on the projection.

Objects are then sorted based on their compactness values. Storage of the objects are re-arranged by moving those with higher compactness to the front. The higher a value is, the better the object is deserved to be an occluder. We use a *Compactness Threshold* (CT) to find out the objects suitable to be occluders. The CT defines the lower bound compactness. The objects whose compactness values are above the CT will be added into the *Candidate Occluder Set* (COS).

5.2. Occluder Selection

At the time a frame being rendered, a group of occluders from the COS is view-dependently selected. We denote it as *Active Occluder Set* (AOS). We perform three steps to decide AOS: (1) view-frustum culling: we test each object's AABB against the view frustum, so that the object is invisible if its AABB outside the frustum; (2) weighting candidate occluders: we weight the objects in COS to determine any one

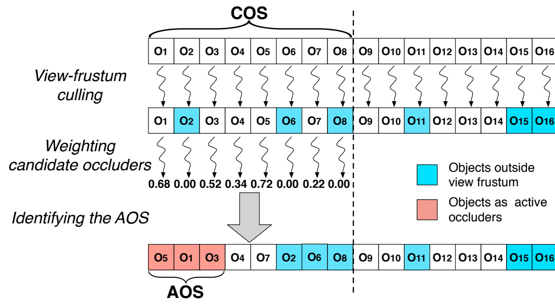


Figure 3: ***Occluder selection example.*** *8 out of 18 objects are classified into COS. The size of AOS is set to 3.*

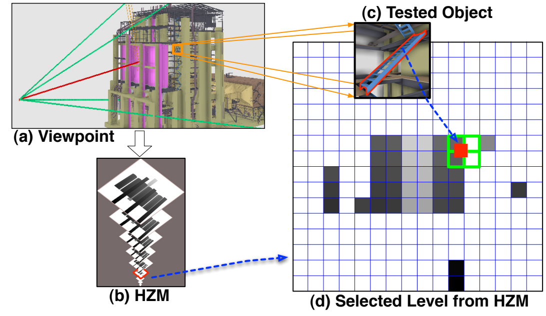


Figure 4: ***The concept of culling with HZM.*** *(a) is the viewpoint of the rendered frame; (b) is HZM constructed from occluders; (c) shows visibility testing with HZM; (d) is image representation of the selected level of HZM. The red square represents the projected size of the object on the screen. It overlaps with four green blocks representing four depth pixels.*

suitable to be in AOS. (3) identifying AOS: We identify the objects with higher weights and select them into the AOS. An example of these three steps is illustrated in Figure 3.

View-frustum culling. We ask one GPU thread to handle one object. By testing its AABB against the view frustum, the objects outside the frustum will not be passed into the rendering pipe. The reason that we use AABBs instead of OOBBs is because an AABB allows a faster execution and has less memory requirements.

Weighting candidate occluders. We certainly do not want to select all remaining objects of COS to AOS, because the number of them most-likely is large and many of them may actually be occluded. To select less but optimal candidates, We develop a weighting metric to ensure that the selected objects are spatially large and close to the viewpoint. we assign a direction vector to each object. The vector is perpendicular to the largest face of the object's OOBB. Then the object's weight is computed based on its viewing direction, its distance to the viewpoint and the size of its bounding volume, as shown in Equation 3.

$$weight = \frac{V||\overrightarrow{N} \cdot \overrightarrow{E}||}{D^3} \tag{3}$$

V is the volume size of the object's AABB; $\overrightarrow{N}$ is the direction vector of the object; E is the direction vector of the viewpoint; D is the closest distance between the AABB and the viewpoint position. Weighting the objects, as shown in Figure 3, can be executed with a object-level parallelism.

Identifying the AOS. We sort the COS based on the descending order of the weights. We fix the size of AOS so that the number of occluders will not be changed during runtime. We consider that the objects in AOS are significantly visible and should be represented with full details.

5.3. Conservative Culling with Hierarchical Z-Map

Conservative culling is to determine a set of objects that are potentially visible, which is commonly known as *Potentially Visible Set* (PVS). To identify the PVS, we build the Hierarchical Z-Map (HZM) with the depth image of active occluders.

Similar to [GKM93], the HZM is constructed by recursively down-sampling the depth image in an octree manner. Each level of HZM is an intermediate depth image, which is down-sampled from the one-level higher image by merging its 2×2 blocks of pixels.

we use the bounding square of an object's projected area as the approximation to test against an appropriate level of HZM. This appropriate level can be determined by Equation 4, where R represents the dimension of the bounding square in pixels; L represents the total levels of HZM; W is the dimension of the rendered frame.

$$level = L - log_2 \frac{W}{R}, (R \geq 1) \tag{4}$$

The area covered by the square is guaranteed in a range of pixel size of $(1,4)$ at the chosen level, see Figure 4. If the depth value of the tested object is larger than all of the four overlapped pixels, this object is surely occluded; otherwise, it will be labeled as a potentially visible object into PVS.

Table 1: ***Parameter Configurations.***

Model	α	N	PET	CT	Size of AOS
Boeing 777	3.0	12.2M	1 pixel	0.55	20
Power Plant	3.0	3.5M	1 pixel	0.88	15

6. Experimental Results

We implemented our approach on a workstation with 8 GB RAM and a Nvidia Quadro 5000 card with 2.5 GB memory.

© The Eurographics Association 2014.

Table 2: ***The results of preprocessing.***

Model	Data File		Collapsing		Occupancy	
	Tris /Vers Numbers	Memory Occupied	Memory Occupied	Time	Memory Occupied	Time
Boeing 777	332M/223M	6.7GB	582.5MB	952min	2.9MB	38min
Power Plant	12M / 6M	0.5GB	14.2 MB	40min	0.6MB	5min

We used Nvidia CUDA Toolkit v4.2 on a 64-bit Windows system. The parameters are listed in Table 1.

6.1. Preprocessing Performance

The preprocessing stage includes two parts: the simplification to record the collapsing information and the computation of the objects' spatial occupancy. We performed them on a single CPU core for both test models. The performance results are shown in Table 2. The simplification costs more time than the computation of spatial occupancy. The average throughput performance of our preprocessing method is 5K triangles/sec. Comparing to other approaches, [YSGM04] computed the CHPM structure at 3K triangles/sec; [CGG*04] constructed the multiresolution of the static LODs at 30k triangles/sec on a network of 16 CPUs; [GM05] built their volumetric structure at 1K triangles/sec on a single CPU. Our method is at least 66.7% faster in single CPU execution. In terms of memory complexity, we generated the addition data that is 8.7% and 3.0% of the original data size for the Boeing model and the Power Plant model, respectively.

6.2. Runtime Performance

We created a navigation path to evaluate runtime performance. Table 3 shows the breakdown of runtime performance. For rendering the Boeing model, transferring frame-different data from CPU to GPU leads to a high cost on the PCIe bus on the LOD processing component. Occlusion culling component is efficient, since the size of AOS is fixed and small, transferring the occluders is not expensive.

Comparison with previous approaches. Our runtime method can reach an average throughput of 110M triangles/sec. In contrast, [CGG*04] performed an average of 70M triangles/sec using their TetraPuzzle method. [GM05] sustained an average of 45M voxels/sec with their Fast Voxel method. Thus, we gain the advantages of using GPUs in terms of the performance of processing triangles. We also compare the runtime performance to the performance of [PC12]'s approach. We set the same value of N in both approaches, so that their GPU workloads are identical. Figure 6 plots the frame rates over 350 frames.

Effectiveness of occlusion culling. We evaluate the effectiveness of our occlusion culling method by comparing it

Table 3: ***Runtime Performance.*** *We show the average breakdown of frame time. The results are averaged over the total number of frames on the camera navigation paths. The models are rendered to a 512×512 image.*

Model	FPS	Occlusion Culling	LOD Processing	Rendering
Boeing 777	14.5	7.9 ms (11.4%)	34.6 ms (50.1%)	26.5 ms (38.4%)
Power Plant	78.3	4.8 ms (37.5%)	2.9 ms (22.7%)	5.1 ms (39.8%)

Table 4: ***Occlusion culling effectiveness on Boeing model.***

Object Number	Objects culled by OC		OC Accuracy	Memory
	Our Approach	Exact	(Our Approach)	Released
718K	63K	108K	58.3%	348.6MB

to the exact culling. In terms of the implementation of the exact culling, we first render the original model into a depth buffer, and then AABBs of the objects are tested against the depth buffer to identify the occluded objects. For the purpose of accuracy, the LOD algorithm is not applied to either of the phrases. This implementation renders one frame in 3-4 seconds. Table 4 shows that our heuristic method achieves 58.3% of the result of the exact culling. "Memory Released" indicates the memory was occupied by the hidden objects, and now is released to add more details to the simplified versions of visible objects. Figure 5 demonstrates the results of occlusion culling for the Boeing model.

7. Conclusions and Future Work

We have presented a GPU-based approach to integrate occlusion culling into the LOD algorithm on GPU. The GPU

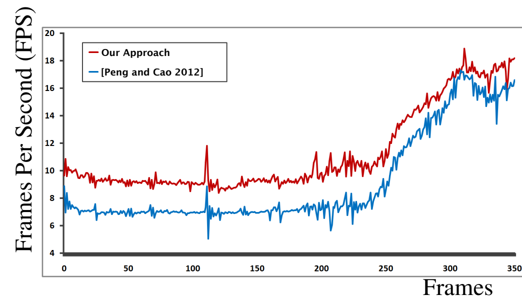
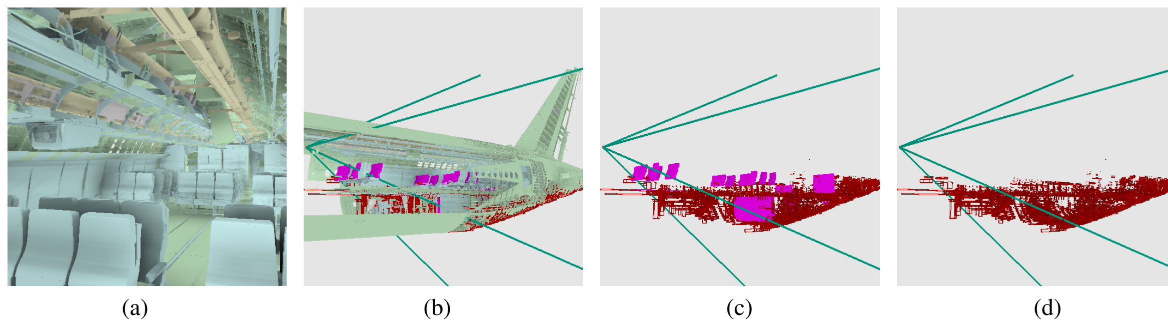


Figure 6: ***Runtime performance.*** *Our approach performs better than [PC12]. Theoretically, our performance should be worse than [PC12] because of the extra cost on occlusion culling. But the removed hidden objects decrease the amounts of vertices and triangles transferred to the GPU, so that our approach spends less time on the CPU-to-GPU data transfer, which more significantly impacts the performance.*

© The Eurographics Association 2014.



(a) (b) (c) (d)

Figure 5: *The occlusion culling result with Boeing 777 model. (a) is the rendered frame. (b) is the reference view, where the dark green line indicates the view frustum. (c) shows the active occluders, marked in purple. (d) shows the occluded objects, where each red box represents one object blocked by the occluders.*

memory occupied by hidden objects are released and used by the visible objects to increase geometric details.

There are several aspects to strengthen our work in the future. Rendering quality is sensitive to the metrics of LOD selection and occluder selection. We would like to explore other metrics that can deliver better performance and rendering qualities. We also want to further improve our occluder selection method to cull more hidden objects.

References

[ASVNB00] ANDÚJAR C., SAONA-VÁZQUEZ C., NAVAZO I., BRUNET P.: Integrating occlusion culling and levels of detail through hardly-visible sets. *Computer Graphics Forum 19*, 3 (2000), 499–506. 2

[BWPP04] BITTNER J., WIMMER M., PIRINGER H., PURGATHOFER W.: Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum 23*, 3 (2004), 615–624. 2

[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 multiresolution polygonal models. In *ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), SIGGRAPH '04, ACM, pp. 796–803. 5

[DG12] DERZAPF E., GUTHE M.: Dependency-free parallel progressive meshes. *Comp. Graph. Forum 31*, 8 (Dec. 2012), 2288–2302. 2

[DMG10] DERZAPF E., MENZEL N., GUTHE M.: Parallel view-dependent refinement of compact progressive meshes. In *Eurographics Symposium on Parallel Graphics and Visualization* (2010), pp. 53–62. 1, 2

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 209–216. 2

[GKM93] GREENE N., KASS M., MILLER G.: Hierarchical z-buffer visibility. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1993), SIGGRAPH '93, ACM, pp. 231–238. 2, 4

[GM05] GOBBETTI E., MARTON F.: Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, ACM, pp. 878–885. 2, 5

[GSYM03] GOVINDARAJU N. K., SUD A., YOON S.-E., MANOCHA D.: Interactive visibility culling in complex environments using occlusion-switches. In *Proceedings of the 2003 symposium on Interactive 3D graphics* (New York, NY, USA, 2003), I3D '03, ACM, pp. 103–112. 2

[Hop97] HOPPE H.: View-dependent refinement of progressive meshes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 189–198. 2

[HSH09] HU L., SANDER P. V., HOPPE H.: Parallel view-dependent refinement of progressive meshes. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 169–176. 1, 2

[LT99] LAW F.-A., TAN T.-S.: Preprocessing occlusion for real-time selective refinement. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics* (New York, NY, USA, 1999), I3D '99, ACM, pp. 47–53. 2

[PC12] PENG C., CAO Y.: A gpu-based approach for massive model rendering with frame-to-frame coherence. *Computer Graphics Forum 31*, 2 (2012). 1, 2, 3, 5

[SKK*14] STEINBERGER M., KENZEL M., KAINZ B., WONKA P., SCHMALSTIEG D.: On-the-fly generation and rendering of infinite cities on the gpu. *Computer Graphics Forum 33*, 2 (2014), 105–114. 2

[YSGM04] YOON S.-E., SALOMON B., GAYLE R., MANOCHA D.: Quick-vdr: Interactive view-dependent rendering of massive models. In *Proceedings of the conference on Visualization '04* (Washington, DC, USA, 2004), VIS '04, IEEE Computer Society, pp. 131–138. 5

[YSM03] YOON S.-E., SALOMON B., MANOCHA D.: Interactive view-dependent rendering with conservative occlusion culling in complex environments. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), VIS '03, IEEE Computer Society, pp. 22–. 2

© The Eurographics Association 2014.