行政院國家科學委員會專題研究計畫　成果報告

虛擬實境中結合幾何與影像之顯像技術研究(III)

計畫類別：個別型計畫
計畫編號：NSC91-2213-E-009-065-
執行期間：91 年 08 月 01 日至 92 年 07 月 31 日
執行單位：國立交通大學資訊工程學系

計畫主持人：莊榮宏

報告類型：精簡報告

處理方式：本計畫可公開查詢

中　華　民　國 92 年 12 月 18 日

中文摘要

我們發展了一套顯像系統利用可見性的區域性 以及多層次精細度幾何模型配合貼圖來大幅提昇顯像效率，而代價爲些微的影像品質降低、額外的儲存空間和預先讀取。 首先場景會被切割成蜂巢狀，蜂巢內部的物體 會以一般的方式顯像，而蜂巢外部的物體則以多層次幾何模型配合投射式貼圖來顯像。 多層次精細度貼圖幾何模型是以原始幾何模型爲基礎依據在蜂巢及其鄰近蜂巢中心 所取得的深度影像資料所簡化而來。 多層次精細度貼圖幾何模型可以避免或減少影像式顯像法所遭遇的問題— 如因物體間遮擋所產生的破洞問題、解析度不協調所造成的裂縫問題； 在我們的系統中, 因自我遮擋所產生的破洞大小能被侷限在使用者所給定的數值之內。 我們測試了數個有數百萬多邊形的場景，在只有些微影像品質的損失下, 成功地將顯像頻率提升到每秒 200 張以上。

關鍵字: 即時瀏覽，混合式顯像，多層次精細度，多層次精細度貼圖幾何模型

Abstract

We present a hybrid rendering scheme that explores the locality of visibility at the cost of extra storage and prefetching, and makes a tradeoff between image quality and rendering efficiency by using textured level-of-detail (LOD) meshes. The space is first subdivided into cells. For each cell, inside objects are rendered as normal while outside objects are rendered as textured LOD meshes using projective texture mapping. The textured LOD meshes are object-based and derived from the original meshes based on the captured depth images viewed at the centers of the cell and its adjacent cells. With such a textured LOD mesh, problems commonly found in image-based rendering, such as the hole problem due to occlusion among objects and the gap problems due to resolution mismatch, can be avoided. The size of holes due to self occlusion is constrained to be within a user-specified tolerance. Several scenes with millions of polygons have been tested and higher than 200 FPS has been achieved with a little loss of image quality.

Keywords: Interactive walkthrough, hybrid rendering, level-of-detail, textured LOD mesh, image-based rendering

1 Introduction

In order to achieve an immersive visual effect during the VR navigation, rendering with photo-realistic scene images in high frame rate has been an ultimate goal of real-time rendering. In the traditional geometry-based rendering, very complex scenes often consist of numerous polygons that cannot be rendered at an acceptable frame rate even using a state-of-the-art hardware. Many techniques have been proposed in last decades on reducing the polygon count while preserving the visual realism of the complex scenes, including visibility culling, level-of-detail (LOD) modeling, and image-based rendering (IBR). Although image-based rendering is capable of rendering complex scenes with photo-realistic images in the time that is independent of the scene complexity, it has been suffered from the static lighting, the limited viewing degree of freedom, and some losses of image quality due to gaps and holes. As a consequence, hybrid rendering that combines geometry- and image-based technique has become a viable alternative.

As a representation for an object or a region of the scene, several image-based or hybrid representations have been proposed. Shade et al. [22] described a paradigm in which regions or objects could be represented by environment map, planar sprite, sprite with depth, layered depth image (LDI), and polygonal mesh, depending on their distances to the viewer. Although the scheme integrates several existing representations, each individual form has its own problems. For example, sprites in general have gap problem due to resolution mismatch, and have to be re-computed once the viewer is outside the safe-region. LDI can only be drawn using software rendering with splatting. Finally, transition between different representations may produce noticeable popping effects.

To reduce gap problems due to resolution mismatch and to improve the efficiency of pixel-based rendering, depth meshes are extracted from the sprite with depth based on depth variation. However, rubber artifacts between disjoint surfaces are often encountered, and re-projecting pixel coordinates back to 3D coordinates may result in precision problems. The depth mesh approach can be incorporated by space subdivision, in which, when navigating inside a cell, distant objects are rendered using depth meshes with textures while near objects are rendered by selected LOD models. With such approaches, the polygon count of a complex scene can be still high and, most importantly, the transition between LOD and depth mesh with texture will generally results in visually noticeable popping effects.

Another more uniform representation is level-of-detail modeling, which can be incorporated with texture mapping for recovering surface details. View-independent LOD modeling has no control over silhouette during navigation. View-dependent LOD modeling, however, has to deal with silhouette problems at run-time by maintaining a mesh of fine resolution along silhouettes. Silhouette clipping that incorporates LOD modeling and normal/texture map needs to extract fine silhouettes at run-time, which is in general time consuming.

2 Related work

There have been extensive research in the field of real-time rendering, ranging from geometry-based rendering, image-based rendering, and hybrid rendering. Although culling, including back-face culling, view-frustum culling, and occluding culling, is a classical technique to clip out invisible polygons, many new approaches have been proposed. In [13], a sub-linear algorithm has been proposed for hierarchical back-facing culling. Zhang et al. improved this by introducing *normal mask* which reduces the per polygon back-face test to only one logical AND operation [25].

LOD modeling has been very useful in further reducing the number of polygon that are visible and inside the view frustum. Distant objects get projected to small areas on the screen and hence can be represented with coarse meshes. On the other hand, nearby objects share larger screen areas and should be modeled by meshes of higher resolution. Many LOD techniques have been proposed; for example, *vertex clustering* [16] *vertex decimation* [19], *edge collapsing*, *progressive mesh* [11] and *view dependent LOD* [12][24]. View-independent LOD can be incorporated with texture map to recover surface details as proposed in [4]; however, the silhouette cannot be recovered since it is view dependent. View-dependent LOD preserves silhouettes; but at the cost of fine mesh resolution along the silhouettes as well as complicated texture mapping. Silhouette clipping took different approach that clips an enlarged

coarse mesh by the exact exterior silhouettes derived at run-time [17].

Geometry-based rendering based on visibility culling and LOD modeling alone usually still cannot meet interactive requirement for very complex scenes. IBR has been a well-known alternative. IBR takes parallax into account, and renders a scene by interpolating neighboring reference views [3][15]. IBR has efficiency that is independent of the scene complexity, and can model natural scenes using photographs. It is, however, often constrained by the limited viewing degree of freedom, and may result in problems like folding, gap, and hole. LDI [22] is a good try to eliminate hole problems due to the visibility changes. LDI structure is more compact in the sense that redundant information has been reduced when several neighboring reference images are composed into a single LDI. However, a splatting is necessary for overcoming the gap problem. Lumigraph [10] and light field rendering [14] have been proposed to reduce the $7D$ *plenoptic function* to a $4D$ function for static scenes. However, both require storage for the extremely large number of images.

Hierarchical image caching proposed in [18][21] is the first approach that combines geometry-based rendering and IBR, aiming to achieve an interactive frame rate for complex static scenes. The cached texture possesses no depth and, in turns, limits its life cycle. The image simplification schemes proposed in [5][23] represent background or distant scene using depth meshes derived from the captured depth images. Such depth meshes are rendered by re-projection and texture mapping. In such approaches, folding problems and gaps resulting from the resolution changes can be eliminated; however, the hole problems due to occlusion among objects and self-occluding still remain. Moreover, disjointed surfaces might be rendered as connected, and depth meshes derived from the depth images are in pixel resolution, which might lead to geometric inaccuracy when re-projected into 3D space. In [7], Decoret et al. proposed multi-layered impostors to constrain visibility artifacts between objects to a given size, and a dynamic update scheme to improve the gap due to resolution mismatch. However, it still encountered hole problems due to self occlusion, and for an efficient dynamic update, a special hardware architecture is needed. In [1], an interactive massive model rendering system using geometric and image-based acceleration is proposed, in which distant objects are represented by textured depth meshes and near objects by LOD models.

3 Proposed hybrid rendering scheme

The proposed hybrid scheme consists of a preprocessing phase and a run-time phase. In the preprocessing phase, the x-y plane of the given 3D scene is first partitioned into equal-sized hexagonal cells. Then for each cell, we derive object-based textured LOD meshes, called SVMesh (single-view LOD mesh) or MVMesh (multi-view LOD mesh), for each object outside the cell. Note that with object-based LOD meshes, the holes due to occlusion among objects can be avoided. Furthermore, substituting original meshes with textured SVMeshes or MVMeshes allows us to make a tradeoff between image quality and rendering efficiency. The SVMesh is a LOD mesh associated with the object whose potential self-occluding error is within a user-specified tolerance. Such a constraint ensures that the potential holes found in the image of an SVMesh viewed from any point inside the cell will have size less than the user-specified tolerance. The MVMesh will be associated with objects who fail to pass the self-occluding-error test. Before deriving SVMesh, those objects legitimate to SVMesh are tested for a possible clustering operation. Such an operation clusters those objects whose union is still legitimate to SVMesh and possesses a reduced texture size. After SVMesh or MVMesh is derived for each object outside the cell, an optional cell-based occlusion culling can be performed to further reduce the polygon count.

Both the SVMesh and MVMesh are derived from object's original meshes, with emphasis on preserving interior and exterior silhouettes. SVMesh is derived from polygons in original mesh that are front-facing to the cell's center while MVMesh comes from polygons that are front-facing to the whole cell.

At run-time phase, window culling and view-frustum culling are performed for the whole scene, followed by a back-facing culling for all objects inside the current navigation cell and a run-time occlusion culling for all meshes. SVMeshes and MVMeshes with associated textures are then texture mapped by hardware-accelerated projective texture mapping and meshes inside the cell are rendered as normal. To reduce the overhead of loading data from secondary storage when navigating across the cell boundary, a prefetching mechanism is applied to amortize the loading to previous frames.

3.1 Preprocessing phase

The steps of the preprocessing phase are shown in Fig. 1.

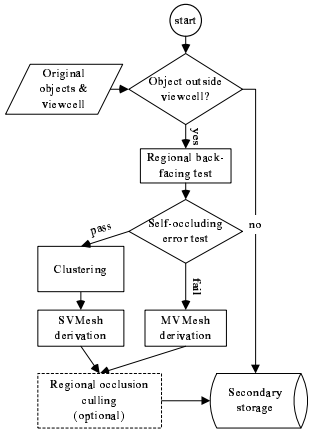


Figure 1: Preprocessing.

Hexagonal spatial subdivision
In order to utilize the spatial locality of visibility, we subdivide the x-y plane of the scene into $N \times M$ hexagonal cells. With the spatial subdivision, the viewpoint can be localized to cells, and, therefore, cell-based visibility culling, back-facing and occlusion culling can be performed in the preprocessing phase.

Self-occluding-error test
Since the SVMesh of an object represents only those polygons that are front-facing to the cell's center, the images derived from SVMesh for views other than the cell's center may have holes due to the self-occlusion. The self-occluding error of an object O, can be approximated by a value s, derived based on those polygons that are front-facing w.r.t. the cell. The self-occluding-error test is to

check if s is smaller than a predefined tolerance T_s specified in image resolution. If it is, the object is represented by an SVMesh; otherwise by an MVMesh.

SVMesh derivation

SVMesh intends to provide a textured LOD model for the portions of an object that is front-facing to the cell's center. The SVMesh is derived by simplifying the object using edge collapsing. The vertices are associated with weights derived from the depth variation found on the object's depth image captured at the cell's center. The cost of collapsing an edge is defined as a function of vertex's weights as well as the local geometry. The weight assignment is designed to distinguish important geometric features such as exterior silhouettes, interior silhouettes, and sharp edges such that those features can be preserved according to their importance during the simplification.

Fig. 2(a) presents the flowchart for the derivation of SVMesh. Fig. 3 depicts the SVMeshes of a bunny model.

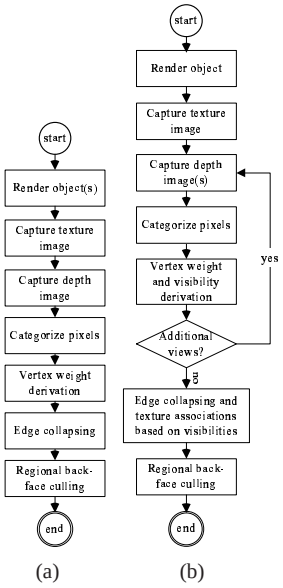


Figure 2: The derivations of SVMesh (a) and MVMesh (b).

Edge collapsing

To perform edge collapsing [11], the cost of collapsing an edge (v_i, v_j) is defined as

$$cost(v_i, v_j) = (1.5 - n_i \cdot n_j)^2 l(w_i + w_j),$$

where n_i and n_j are normals of v_i and v_j, respectively, l is the edge's projected length with respect to the cell's center, and w_i and w_j are the weights of v_i and v_j, respectively.

MVMesh derivation

The derivation of the MVMesh is an extension of that for SVMesh; as shown in Fig. 2(b) [2]. For MVMesh, we consider those polygons that are front-facing with respect to the cell, rather than cell's center. Furthermore, the derivation of the vertex's weight takes into account the captured depth images viewed at the centers of the cell and its adjacent cells. For each vertex, a weight is obtained from each depth image as we do for the SVMesh and the vertex is assigned with the maximum of all those weights.

For the cost function of an edge, we should replace l, the projected length of an edge with respect to the cell's center, by l', which is the projected length of the edge with respect to the cell. When the object is far from the cell, we have $l \approx l'$. The edge's projected length for a near object, however, varies when we navigate in the cell. Fig. 4 depicts the MVMeshes of the bunny model.

Regional conservative back-face culling

We claim that if a polygon is back-facing to all six vertices of the cell, the polygon is back-facing with respect to any point inside the cell. That is, a polygon P is back-facing with respect to the cell C if

$$dot\_product(P.\text{normal}, vector(C_i, P.\text{center})) < 0, \text{for } \; i = 0, \ldots, 5$$

where C_i's are the corners of C.

Object clustering

In order to reduce the texture size associated with LOD meshes and to reduce polygon count, objects that pass the self-occluding-error test and are close to each other can be clustered together, provided that certain conditions are satisfied. The clustering operation amounts to the coloring problem, and itself is an NP-complete problem. We propose a greedy approach that proceeds as follows. Firstly, objects that pass self-occluding-error test are sorted according to the size of their projected areas. Initially no cluster is formed. Secondly, for each object M removed from the sorted list, M itself forms a new cluster if there is no cluster or no cluster found to be cluster-able with M. Otherwise, M is repeatedly clustered with all the clusters that M is cluster-able with, in the order of decreasing overlapping size. As shown in Fig. 5(a), M is cluster-able with C_1, C_2 and C_3 in the order of decreasing overlapping size. M is clustered with C_1 first. The result $M \cup C_1$ is, however, is no longer cluster-able with C_2; but still cluster-able with C_3; see Fig. 5(b). Finally, M is clustered with C_1 and C_3; see Fig. 5(c).

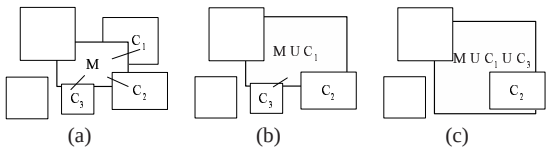


Figure 5: Repeat clustering.

The clustering is performed after the self-occluding-error test is applied for all objects, and before the derivation of SVMesh. The objects in the same cluster are considered as a single object that possesses an SVMesh. The SVMesh derivation can be slightly modified to construct an SVMesh for the clustered objects. In consequence, surfaces that are occluded by others in the cluster will be culled out in the simplification process. Such an SVMesh derivation for clustered objects implicitly performs occlusion culling among objects.

Regional conservative occlusion culling

Since SVMesh or MVMesh is object based and our scheme does space subdivision for utilizing view locality, it will be advantageous to do the regional conservative occlusion culling in the preprocessing phase. Such operations will enhance the rendering efficiency, especially for densely occluded scenes. Methods proposed recently can be used. For example, the extended projection [8] can be easily modified to fit into our system. This extended projection can also

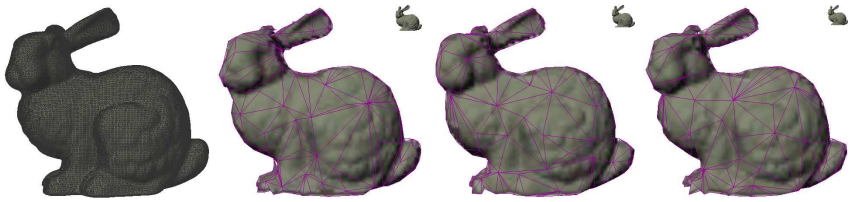


Figure 3: (a) is the original mesh ($65,491$ polygons) of a bunny viewed at one cell away (cell size 50), and (b-d) are SVMeshes for the bunny at 7 (259 polygons), 8 (254), and 9 (239) cells away. The upper-right bunnies are the projected images.

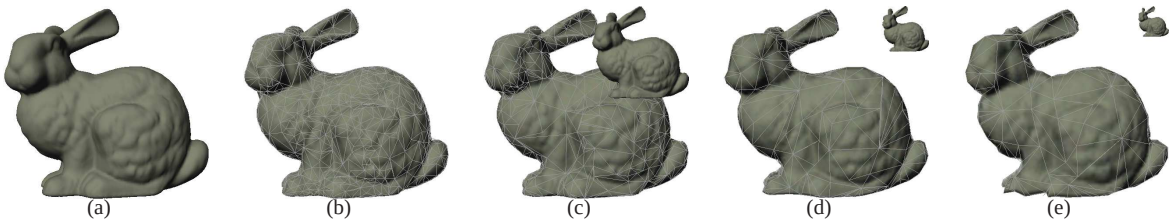


Figure 4: (a) is the original mesh ($65,491$ polygons) of a bunny viewed at one cell away (cell size 50), (b-g) are MVMeshes of the bunny at 1 ($1,605$ polygons), 2 (945), 4 (392), 6 (306) cells away. The upper-right indicates actual projected images.

handle the case of multiple occluders by using occluder fusion. The selection of occluders is based on the meshes' projected sizes. Only those meshes whose projected sizes are larger than a user-specified threshold are selected to be occluders.

3.2 Run-time phase

At the run-time phase, within the current navigation cell we first set up a lower priority thread for prefetching the geometry and image data belonging to neighboring cells, and then do the steps shown in the Fig. 6 when navigating inside the cell.

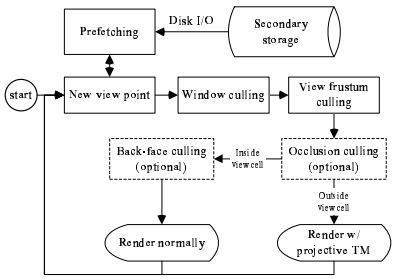


Figure 6: Run-time phase.

A view with an FOV sees through a fixed number of windows, which are faces of the navigating cell. Window culling can be considered as an effective pre-calculation of the view-frustum culling. As optional operations, the run-time back-face culling and occlusion culling can be applied to further reduce the polygon count. Back-facing culling is performed only for objects inside the navigation cell, while occlusion culling is applied to all meshes in the scene.

SVMeshes are simply rendered by projective texture mapping [20] while the rendering of MVMeshes involves texture blending as part of view-dependent projective texture mapping [6].

One of the major problems arises in our cell-based navigation is how to achieve smooth cell transition. When the view point moves across from one cell to its neighbors, the geometry and textures will be switched. The prefetching is a mechanism to preload the geometry and texture data of neighboring cells when CPU load is relative low during navigating inside the cell. It will amortize the loading time to several inside-cell frames and hence reduce the FPS gap between inside-cell frames and a cross-boundary frame.

4 Experiments

Setup
The test platform is a PC with an AMD AthlonXP 1800+ CPU, 512MB main memory, and an *n*VIDIA GeForce4 Ti 4400 with 128MB DDR RAM graphics accelerator. The OS is Windows XP Pro. The output image is in a resolution of $1024 \times 1024 \times 32$. S3's S3TC DXT3 is used to compress textures (in a ratio of $1/4$).

Scene statistics
The three scenes tested are statuary parks consisting of eight kinds of object that are randomly distributed in the same area of 1650×2035. The three scenes are called 2M-scene, 4M-scene, and 8M-scene, and have 2017700, 4188885 and 8004863 polygons, respectively. The scenes are generated such that 2M-scene is a subset of the 4M-scene, which in turn is a subset of 8M-scene. Table 1 lists data statistics for the objects that compose the scenes, including polygon number, dimension, and distribution of polygon numbers for the scenes.

Settings
Performance on frame rate and image quality may vary for different settings of parameters. We set $T_s = 3$, 5, or 7 pixels for self-occluding-error tolerance, $T_l = 3.0$, 4.5, or 6.0 for edge's project length tolerance, and 50 or 100 for cell

Table 1: Object and scene statistics.

Object name	Polygon no.	Dimension (w×d×h)	2M	4M	8M
dragon	202,520	$57.3 \times 25.6 \times 40.4$	4	10	18
bunny	69,451	$43.6 \times 33.8 \times 43.2$	7	12	26
statue	35,280	$11.8 \times 13.4 \times 23.4$	13	21	40
cattle	12,398	$40.0 \times 40.8 \times 30.7$	9	19	42
horse	7,257	$38.3 \times 57.2 \times 82.6$	13	29	51
easter	4,976	$12.4 \times 10.7 \times 30.8$	6	14	22
camel	3,969	$49.4 \times 16.8 \times 46.6$	4	14	26
venus	1,396	$10.2 \times 8.4 \times 21.9$	8	13	28
		Total object number	64	132	253

size. The parameters T_{C^0} and T_{C^1} for pixel categorizing are fixed in this experiment as 3.4×10^{-4} and 1.28×10^{-4}, respectively. For simplicity, we denote the kM-scene with cell size c, parameters T_s and T_l as kM-c-T_s-T_l; for example, the 4M-scene with cell size 50, $T_s = 5$, and $T_l = 4.5$ is denoted as 4M-50-5-4.5.

All experimental results are collected by following the same navigation path with a maximum speed of 30/s., a maximum rotation of 45^o/s., and an FOV of 60^o.

Image quality measurement

To identify how much is the quality-loss, we use the peak signal-to-noise ratio PSNR(dB) defined as

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{\frac{1}{HW} \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} \left[\hat{f}(x,y) - f(x,y) \right]^2},$$

where $f(x,y)$ and $\hat{f}(x,y)$ are the pixel colors of the original image and approximated image at position (x,y), respectively, W and H are the dimensions of the image. Before applying PSNR, the RGB color is mapped to a single luminance value Y since human eyes are more sensitive to the changes in luminance. Such a mapping [9] is

$$Y = 0.299 * R + 0.587 * G + 0.114 * B.$$

4.1 Mesh simplification

Self-occluding-error tolerance

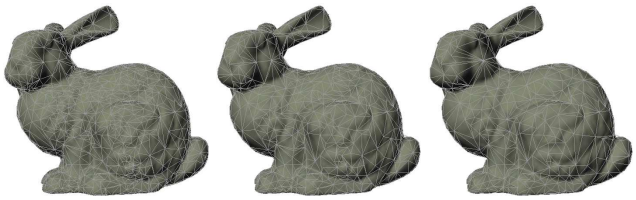
The value of self-occluding-error tolerance T_s determines the distribution of SVMesh and MVMesh. Our experimental results show that larger T_s implies higher percentage of SVMesh, more objects are clustered, higher simplification rate, less texture size, and finally higher frame rate (271.5, 279.0, and 281.9 frames/s. for $T_s = 3$, 5, and 7 respectively for the 4M scene).

Projected edge-length tolerance

Through projected edge-length tolerance T_l, the edge collapsing can be tested for termination. Fig. 7 shows the MVMeshes of bunny derived by setting $T_l = 3.0, 4.5, 6.0$. Larger T_l implies higher simplification rate, larger texture size, and finally higher frame rate (265.4, 278.0, and 289.2 frames/s. for $T_l = 3.0$, 4.5, and 6.0 respectively for the 4M scene).

Cell size consideration

Setting an optimal cell size is in general difficult. The results show that larger cell size in general results in smaller simplification ratio and, in turns, lower frame rate (278.0 and 255.5 frames/s. for cell size 50 and 100 respectively) since the number of polygons inside a cell may increase dramatically.



(a) $T_l = 3.0$ (2,353 polygons) (b) $T_l = 4.5$ (1,605) (c) $T_l = 6.0$ (1,227)

Figure 7: MVMeshes of bunny for different T_l.

4.2 Run-time performance

The three rendering configurations used to test the performance comparison are:

- **A: (Pure geometry)** render the original scene geometry using the traditional graphics pipeline.
- **B: (Pure geometry with view frustum culling)** same as **A**, but with software view frustum culling.
- **C: (Proposed hybrid scheme)** render the scene using proposed hybrid scheme, without regional occlusion culling, run-time back-face culling, and run-time occlusion culling.

The parameter setting for the following performance tests is $T_s = 5$, $T_l = 4.5$, and cell size 50.

Table 2 lists the run-time performance of three configurations on the scene 8M-50-5-4.5. Without regional occlusion culling, back-face culling, and run-time occlusion culling, configuration **C** achieves 274.8 gain factor over configuration **A**, and 76.9 gain factor over configuration **B**, with little quality-loss at PSNR 37.34dB.

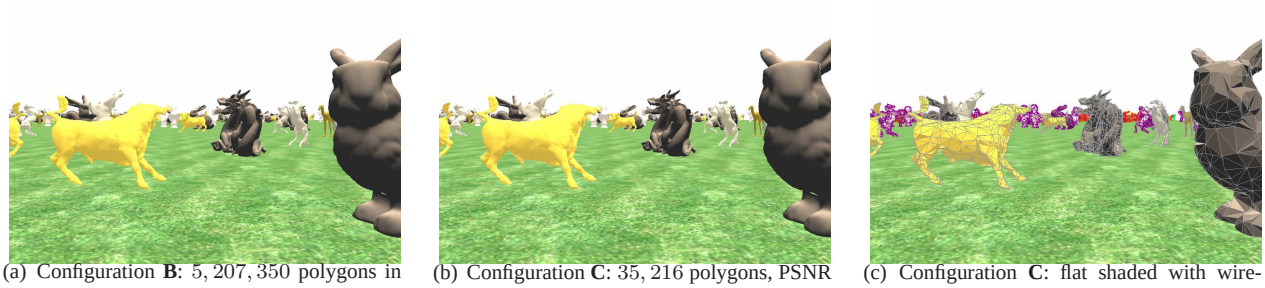
Table 2: Performance of the three configurations on a 8M-scene.

	A	**B**	**C**
Avg. polygon count	8,004,863	2,443,969	23,580
Avg. frame time (ms)	1,247	349.1	4.54
Avg. frame rate (FPS)	0.802	2.864	220.4
Speedup	1.0	3.57	274.8

Fig. 8 represents the images rendered at views that are far from the cell center by configuration **B** and **C**. In Figs. 8(c) and 8(f), the MVMeshes are flat shaded with gray wireframes, SVMeshes from single objects are in purple, and SVMeshes from clustered objects in other colors.

Table 3 depicts the performance of configuration **C** for different scene complexities 2M-50-5-4.5, 4M-50-5-4.5, and 8M-50-5-4.5. It reveals that as the scene complexity goes up from 2M, 4M, to 8M, the FPS goes down from 353, 278, to 220. This is due to the fact that all objects outside a navigation cell are in the form of SVMesh or MVMesh, which have much less varied polygon counts.

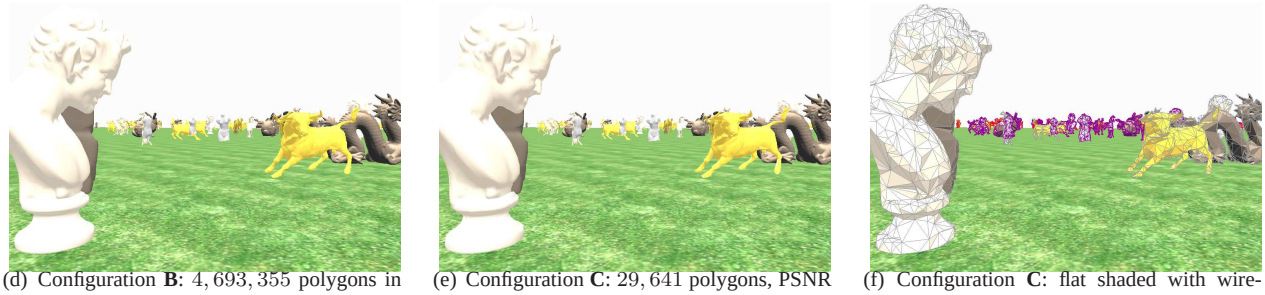
In Fig. 9, FPS plots are shown for different prefetching schemes. From the plot for prefetching under a cold cache, we can see that the frame rate changes rapidly after a cell transition (illustrated by yellow vertical line) and becomes more stable frame rate after a while. The frame rate for the prefetching under a warm cache is quite stable except some sudden decreases appear. The suddenly decreased FPS in the plots indicates the presence of objects inside the navigation cell. Note that most frames have PSNR above 37dB.



(a) Configuration **B**: $5,207,350$ polygons in view frustum.

(b) Configuration **C**: $35,216$ polygons, PSNR 35.33dB, 178.8 FPS.

(c) Configuration **C**: flat shaded with wireframes.



(d) Configuration **B**: $4,693,355$ polygons in view frustum.

(e) Configuration **C**: $29,641$ polygons, PSNR 36.39dB, and 199.3 FPS.

(f) Configuration **C**: flat shaded with wireframes.

Figure 8: Rendered images by configuration **B** and **C**.

Table 3: Performance of configuration **C** under different scene complexities.

Scene complexity	2M	4M	8M
Statistics for polygon counts			
Avg. polygon no. inside a viewcell	$4,145$	$9,308$	$18,302$
Avg. polygon no. for SVMesh & MVMesh	$18,829$	$39,981$	$75,891$
Avg. polygon no. for a viewcell	$22,974$	$49,290$	$94,193$
Simplified : original	$1:87.8$	$1:85.0$	$1:85.0$
Performance statistics			
Avg. FPS	353.3	278.0	220.4
Avg. PSNR (dB)	44.92	39.54	37.34
Avg. texture size (KB) inside view frustum	112.4	544.1	992.4
Avg. polygon count inside view frustum	4,548 (3,991)	13,102 (11,418)	23,580 (21,735)

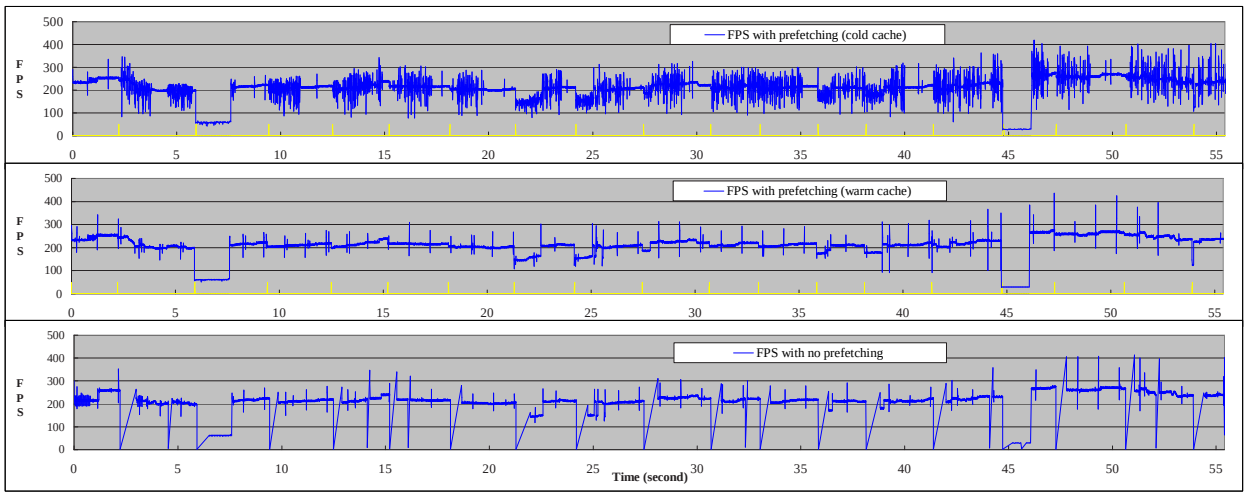


Figure 9: The frame rates with prefetching under a cold cache and a warm cache, and without prefetching of configuration **C** on scene 8M-50-5-4.5.

5 Concluding remarks

We have presented a hybrid rendering scheme for real-time display of complex scenes. The scheme partitions the model space into cells, thus explores the locality of visibility based on which the objects outside a cell are rendered as textured LOD meshes and inside objects are rendered as normal. Such a hybrid representation allows us to avoid problems that are commonly found in image-based rendering; such as the gap problem due to resolution mismatch and the hole problem due to occlusion among objects. The representation also constrains the hole due to self-occlusion to be within a user-specified tolerance. A prefetching mechanism has also been proposed to predict data of which neighbor-

ing cells will be needed shortly and how the loading can be amortized to frames before crossing the cell boundary. In the proposed scheme, acceleration techniques such as regional occlusion culling, back-facing culling, and run-time occlusion culling can be easily integrated. We have demonstrated our system on several scenes consisting of millions of polygons and observed very encouraging results. For a scene of 8 millions of polygons, we have achieved higher than 200 frames per second with a little loss of image quality (average PSNR 37.34dB). The polygons and textures require about 1260MB secondary storage space and about 294MB main memory on average.

Our results has been published on the Journal of *Computers & Graphics* 27(2):189–204, 2003.

References

[1] D. Aliaga, J. Cohen, A. Wilson, E. Baker, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stuerzlinger, R. Bastos, M. Whitton, F. Brooks, and D. Manocha. MMR: An Interactive Massive Model Rendering System Using Geometric and Image-Based Acceleration. In *Proceedings of 1999 ACM Symposium on Interactive 3D Graphics*, pages 199–206, 1999.

[2] C.-C. Chen and J.-H. Chuang. Viewcell-Dependent Geometry Simplification Using Depth. In *Computer Graphics Workshop 2002*, June 2002.

[3] S. E. Chen and L. Williams. View Interpolation for Image Synthesis. In J. T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 279–288, August 1993.

[4] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. A General Method for Recovering Attribute Values on Simplified Meshes. In *Proceedings of IEEE Visualization '98*, pages 59–66, October 1998.

[5] L. Darsa, B. Costa, and A. Varshney. Walkthroughs of Complex Environments using Image-based Simplification. *Computers & Graphics*, 22(1):55–69, 1998.

[6] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and Rendering Architecture from Photographs: A hybrid geometry- and image-based approach. In H. Rushmeier, editor, *Proceedings of SIGGRAPH '96*, pages 11–20, August 1996.

[7] X. Decoret, G. Schaufler, F. X. Sillion, and J. Dorsey. Multi-Layered Impostors for Accelerated Rendering. *Computer Graphics Forum*, 18(3):61–73, September 1999.

[8] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative Visibility Preprocessing using Extended Projections. In Kurt Akeley, editor, *Computer Graphics (SIGGRAPH 2000 Proceedings)*, pages 239–248, July 2000.

[9] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*, chapter 4, page 228. Addison-Wesley, September 1993.

[10] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. In H. Rushmeier, editor, *Proceedings of SIGGRAPH '96*, pages 43–54. ACM SIGGRAPH, Addison Wesley, August 1996.

[11] H. Hoppe. Progressive Meshes. In H. Rushmeier, editor, *Proceedings of SIGGRAPH '96*, pages 99–108, August 1996.

[12] H. Hoppe. View-Dependent Refinement of Progressive Meshes. In *Proceedings of SIGGRAPH '97*, pages 189–198, 1997.

[13] S. Kumar, D. Manocha, B. Garrett, and M. Lin. Hierarchical Back-face Culling. In *7th Eurographics Workshop on Rendering*, pages 231–240, 1996.

[14] M. Levoy and P. Hanrahan. Light Field Rendering. In H. Rushmeier, editor, *Proceedings of SIGGRAPH '96*, pages 31–42, August 1996.

[15] L. McMillan and G. Bishop. Plenoptic Modeling: An Image-Based Rendering System. In R. Cook, editor, *Proceedings of SIGGRAPH '95*, pages 39–46. ACM SIGGRAPH, Addison Wesley, August 1995.

[16] J. Rossignac and P. Borrel. Multi-resolution 3-D Approximations for Rendering Complex Scenes. In B. Falcidieno and T. L. Kunii, editors, *Modeling in Computer Graphics*, pages 455–465, Berlin, 1993. Springer-Verlag.

[17] P. V. Sander, X. Gu, S. J. Gorlter, H. Hoppe, and J. Snyder. Silhouette Clipping. In *Proceedings of SIGGRAPH 2000*, pages 327–334, July 2000.

[18] G. Schaufler and W. Stürzlinger. A Three-Dimensional Image Cache for Virtual Reality. In *Proceedings of Eurographics '96*, pages 227–236, August 1996.

[19] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of Triangle Meshes. *Computer Graphics*, 26(2):65–69, 1992.

[20] M. Segal, C. Korobkin, R. Widenfelt, J. Foran, and P. Haeberli. Fast Shadows and Lighting Effects Using Texture Mapping. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, pages 249–252, July 1992.

[21] J. Shade, D. Lischinski, D. H. Salesin, T. DeRose, and J. Snyder. Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In H. Rushmeier, editor, *Proceedings of SIGGRAPH '96*, pages 75–82, August 1996.

[22] J. W. Shade, S. J. Gortler, L.-W. He, and R. Szeliski. Layered Depth Images. In *Proceedings of SIGGRAPH '98*, pages 231–242, July 1998.

[23] F. Sillion, G. Drettakis, and B. Bodelet. Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery. In *Proceedings of Eurographics'97*, pages 207–218, Budapest, Hungary, September 1997.

[24] J. Xia and A. Varshney. Dynamic View-Dependent Simplification for Polygonal Models. In *Proceedings of IEEE Visualization '96*, pages 327–334, October 1996.

[25] H. Zhang and K. E. Hoff III. Fast Backface Culling Using Normal Masks. In *Proceedings of 13th Symposium on Interactive 3D Graphics*, pages 103–106, 1997.