PERGAMON

Computers & Graphics 27 (2003) 189–204

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Rendering complex scenes using spatial subdivision and textured LOD meshes ☆

Chih-Chun Chen, Jung-Hong Chuang*, Bo-Yin Lee, Wei-Wen Feng, Ting Chiou

Department of Computer Science and Information Engineering, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 30050, Taiwan, ROC

Accepted 27 November 2002

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 IBR 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. [1] 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

☆ Work supported partially by NSC of ROC under Grant NSC 90-2213-E-009-126.

*Corresponding author. Tel.: 886-3-573-1829; fax: 886-3-572-4176.

E-mail addresses: chihchun@csie.nctu.edu.tw (C.-C. Chen), jhchuang@csie.nctu.edu.tw (J.-H. Chuang), bylee@csie.nctu.edu.tw (B.-Y. Lee), fengww@csie.nctu.edu.tw (W.-W. Feng), tchiou@csie.nctu.edu.tw (T. Chiou).

0097-8493/03/$ - see front matter
doi:10.1016/S0097-8493(02)00276-5

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 LOD 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.

1.1. System overview

A hybrid rendering scheme that aims to render complex scenes in a constant and high frame rate with only a little or an acceptable quality loss is presented in this paper. To this end, view space is partitioned into cells to explore the locality of visibility, and for a view cell, each object outside the cell is represented by a LOD mesh together with textures that are derived with respect to the view cell. All these are done in a preprocessing. In contrast with IBR or depth mesh approach, the object-based LOD mesh derivation avoids hole problems due to occlusion among objects. In the meantime, to reduce hole problems due to self-occluding, the LOD mesh is classified into either single-view LOD mesh (termed as SVMesh) or multi-view LOD mesh (termed as MVMesh), depending on the object's self-occluding error (w.r.t. the viewcell). The SVMesh is chosen if the object's self-occluding error is smaller than a user-specified tolerance, otherwise MVMesh is chosen. Such a condition on SVMesh ensures that the potential holes possibly found in the images viewed from any point inside the cell will have size less than the user-specified tolerance. Hence all the information necessary to guide the derivation of SVMesh and the texture associated with the SVMesh come from the captured image and captured depth image of the cell's center. On the other hand, the MVMesh presents geometry and texture necessary to avoid holes on images viewed from some points in the cell. Therefore, the derivation of MVMesh and its texture associations are based on captured images and depth images from the cell's center as well as the centers of adjacent cells. In the proposed scheme, prefetching is also implemented to preload the data necessary for the following cells such that sudden drops in the frame rate at the cell transition can be avoided.

The proposed approach explores locality of visibility at the cost of extra storage and prefetching, and makes a tradeoff between image quality and rendering efficiency by using the SVMesh and MVMesh together with textures. Our experiments have shown that for a scene of 8 million polygons we have achieved higher than 200 frames/s with a little loss of image quality (average PSNR 37.34 dB). The polygons and textures require about 1260 MB hard disk storage and about 287 MB run-time memory on average. With such high frame rates, the overhead of prefetching is hardly noticeable.

2. Related work

There have been extensive research in the field of real-time rendering, ranging from geometry-based rendering, IBR, 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 [2], a sublinear 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 [3].

Several run-time methods have been proposed for occlusion culling; for example, shadow frusta [4], hierarchical Z-buffer [5], and hierarchical occlusion map [6]. To overcome the inevitable overhead doing occlusion culling at run-time, some recent results focused on regional conservative occlusion culling. Cohen-Or et al. [7] proposed a pre-processing algorithm for regional occlusion culling, but its performance depends heavily on a single strong occluder. Durand et al. [8] proposed *extended projection* operations and Schaufler et al. [9] proposed *blocker extension* to handle occluder fusion of multiple occluders.

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* [10–12], *vertex decimation* [13], *edge collapsing*, *progressive mesh* [14,15], *and view-dependent* LOD [16,17]. View-independent LOD can be incorporated with texture map to recover surface details as proposed in [18]; 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 [19].

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 [20,21]. 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 [1] 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 [22] and light field rendering [23] 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 [24,25] 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 [26–28] 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 [29], 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 [30], 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. The method proposed in [31] integrates LOD and visibility computation and is suitable for scenes with high-depth complexity and very dynamic scene.

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 or MVMesh, 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. Moreover, they also differ in how the vertex's weights are derived for mesh simplification using edge collapsing and how textures are associated with simplified polygons. For SVMesh, the weight associated with each polygon vertex and the texture associated with each simplified polygon are derived only from the object's depth image viewed from the cell's center. On the other hand, for MVMesh, the derivation of vertex's weight and polygon's textures also takes into account the depth images viewed from centers of nearby cells.

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. Pre-processing phase

The steps in the preprocessing phase are (see Fig. 1):

(1) Hexagonal spatial subdivision.
(2) For each cell, for each object outside the cell:
  (a) perform regional conservative back-face test;
  (b) perform self-occluding-error test and select single-view LOD mesh (SVMesh) or multi-view LOD mesh (MVMesh);
  (c) derive SVMesh or MVMesh and texture(s) association;
  (d) perform regional conservative back-face culling.
(3) (Optional) Perform regional conservative occlusion culling.

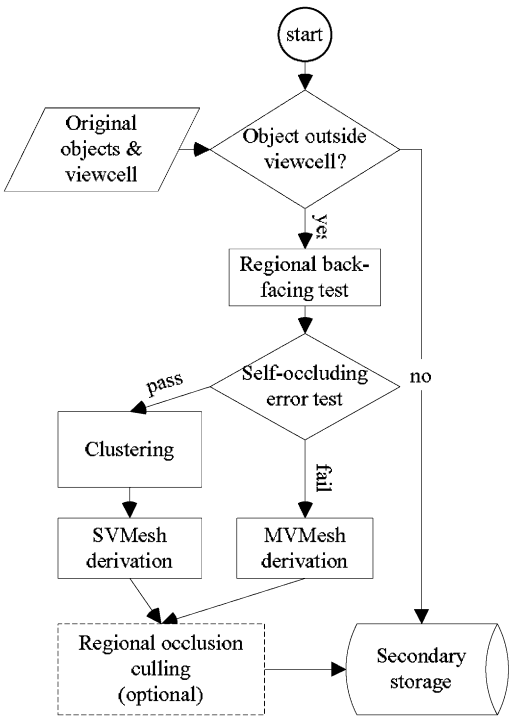


Fig. 1. Preprocessing.

3.1.1. 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. Compared to four for rectangular subdivision, hexagonal subdivision requires that data of only three adjacent cells need to be loaded when navigating across the cell boundary. Table 1 depicts the maximum ratio of side faces that can be seen from a point inside the hexagonal or the rectangular cell under different fields of view (FOVs). We can see that hexagonal subdivision is better than rectangular one in most cases, except that they are equal for the 45°.

3.1.2. 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. Here we describe a conservative estimation of self-occluding error.

As shown in Fig. 2, the maximum error occurs at the farthest view position V' from the cell center V. Let the cell size, i.e., the length of $\overline{VV'}$, be c, the distance between object and the cell center, i.e., the length of $\overline{VO}$, be d, and the depth of the object itself, i.e., the length of $\overline{OP}$, be l. The length of $\overline{OC}$ is $l \tan\theta$, the angle θ between $\overline{VP}$ and $\overline{V'P}$ is $\theta = \tan^{-1} c/(d+l)$, and s, the projected size of $\overline{OP}$ or $\overline{OC}$, is

$$s = (\overline{AB}/c)\ \text{ImageRes}$$

Since

$$\overline{AB} = \frac{\left(\sqrt{3}/2\right)c}{d}\ \overline{OC} = \left(\frac{\left(\sqrt{3}/2\right)c}{d}\right)\left(\frac{cl}{d+l}\right) = \frac{\sqrt{3}c^2 l}{2d(d+l)},$$

we have

$$s = \frac{\sqrt{3}cl}{2d(d+l)}\ \text{ImageRes}.$$

The self-occluding error of an object O, denoted as self-occluding-error (O), is approximated by s, derived based on those polygons that are front-facing w.r.t. the cell.

Table 1
Maximum ratio of side faces seen from a point inside the cell under different FOVs

FOV(°)	120	90	60	45	30
Hexagonal	5/6	2/3	1/2	1/2	1/3
Rectangular	4/4	3/4	3/4	1/2	1/2

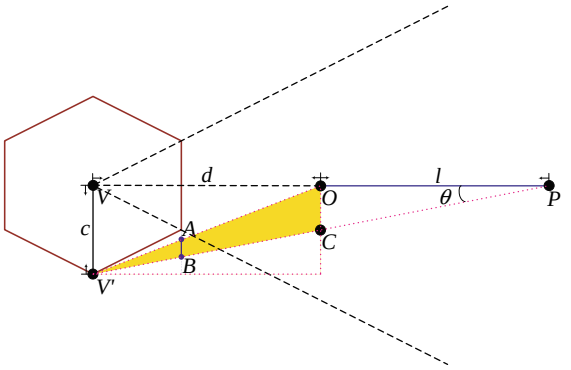


Fig. 2. The maximum self-occluding error occurs at the position V'.

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.

3.1.3. SVMesh derivation

SVMesh intends to provide a textured LOD model for the portions of an object that are 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.

The derivation of the SVMesh of an object O with respect to a cell C is outlined as follows.

(1) Capture the image and depth image of O using cell's face as the window and cell's center as the center of projection.
(2) Categorize pixels on the depth image as *exterior silhouette*, *interior silhouette*, *sharp edge*, and *interior*, and assign each category a weight.
(3) Assign weights to object's vertices:
   - vertices that are back-facing with respect to the center of C: vertex weight is 0.5;
   - other vertices: vertex weight is the weight of the pixel gets projected by the vertex.
(4) Perform edge collapsing in increasing order of edges' cost.

Fig. 3(a) presents the flowchart for the derivation of SVMesh. Fig. 4 depicts the SVMeshes of a bunny model.

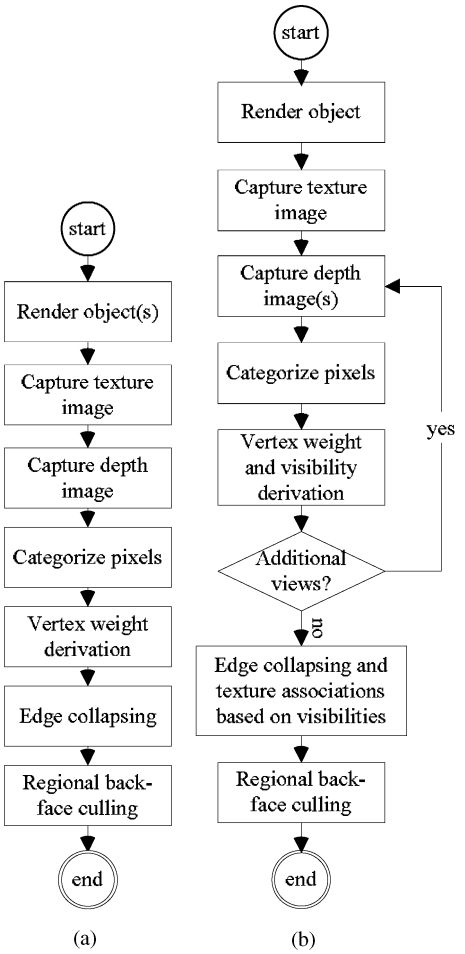


Fig. 3. Derivations of SVMesh (a) and MVMesh (b).

3.1.3.1. Categorizing pixels on the depth image. Pixels on the depth image are categorized into four categories:

- *Exterior silhouette*: a pixel on the external silhouette, which can be extracted using *contour extraction* techniques.
- *Interior silhouette* (C^0-discontinuity): a pixel Z whose value differs from adjacent pixels over a user-specified tolerance T_{C^0}; that is, $Z_{i+1}-Z_i > T_{C^0}$ or $Z_{i-1}-Z_i > T_{C^0}$ (see Fig. 5).
- *Sharp edge* (C^1-discontinuity): a pixel whose Z variation differs from Z variation of an adjacent pixel over a user-specified tolerance T_{C^1}; that is, $|(Z_{i-1}-Z_i)-(Z_i-Z_{i+1})| > T_{C^1}$ (see Fig. 5).
- *Interior:* other pixels whose Z values are different from the background Z value.

Each category corresponds to a weight. We have derived from our experience that 0.5 is for *exterior*

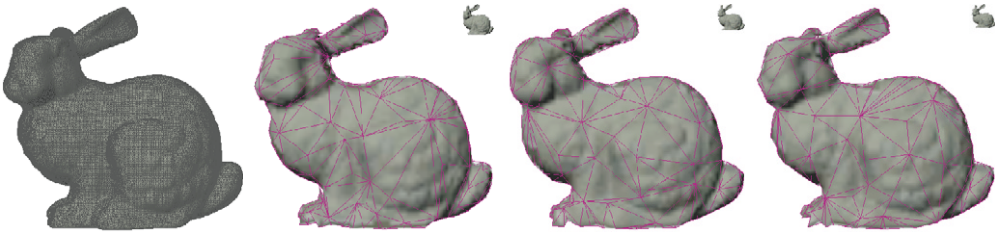


Fig. 4. (a) is the original mesh (65,491 polygons) of a bunny viewed at one cell away (cell size 50). (b–d) are SVMeshes for the bunny at 7(259 polygons), 8(254), and 9(239) cells away. The upper-right bunnies are the projected images.

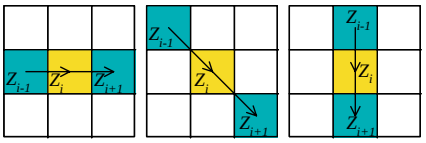


Fig. 5. Testing depth variation.

silhouette, 0.4 for *interior silhouette*, 0.25 for *sharp edge*, and 0.125 for *interior*.

3.1.3.2. Assigning vertex weights. The vertex weight indicates how important the vertex is, which is usually determined by the local geometry and the viewing parameters. Here we propagate the weight derived for pixels on the depth image to corresponding vertices. We first distinguish back-facing and front-facing vertices. A vertex is back-facing (w.r.t. the cell's center) if all polygons incident to it are back-facing (w.r.t. the cell's center), otherwise it is front-facing. Each back-facing vertex is assigned with the weight 0.5 (same as that for the exterior silhouette vertex). For a front-facing vertex, we do the projection and check to see if it is visible to the cell's center by checking its Z-value against the Z-value of the pixel that gets projected. If it is, the pixel's weight is the weight of the vertex, otherwise it is invisible and assigned with the weight 0.05, which is smaller than vertices corresponding to the pixel category *interior*.

3.1.3.3. Edge collapsing. To perform edge collapsing [14], the cost of collapsing an edge (v_i, v_j) is defined as

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

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.

Edges are first maintained in an increasing order according to their costs, and stored in a heap. In each edge collapsing, the edge at top of the heap is removed and the vertex of smaller weight gets collapsed to the other. Such a collapsing order ensures that the edge with smaller cost gets collapsed first. The costs of some edges may be altered as a result of an edge collapsing, and must be updated afterwards. The edge collapsing is repeated until the edge on the top of the heap has cost higher than a user-specified value T_l, where T_l is a tolerance on the edge's projected length w.r.t. the cell's center.

3.1.4. MVMesh derivation

The derivation of the MVMesh is an extension of that for SVMesh, as shown in Fig. 3(b) [32]. 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.

In addition to the weight, each vertex is also associated with a set of views to which the vertex is visible. The views associated with a simplified polygon is determined by the intersection of view sets associated with the polygon's vertices. Since the views associated with a vertex cannot propagate in the course of edge collapsing, we place one more condition on edge collapsing. Namely, for an edge $\overline{uv}$, u can be collapsed to v if the weight of u is smaller than or equal to that of v and both u and v are either visible to some common views or associated with empty view sets. Note that determining polygon's set of views based on that of vertices is not able to reflect the cases in which the polygon is partially occluded, but its vertices are not, by other polygons. Such exceptions should be handled carefully by considering general visibility problems.

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

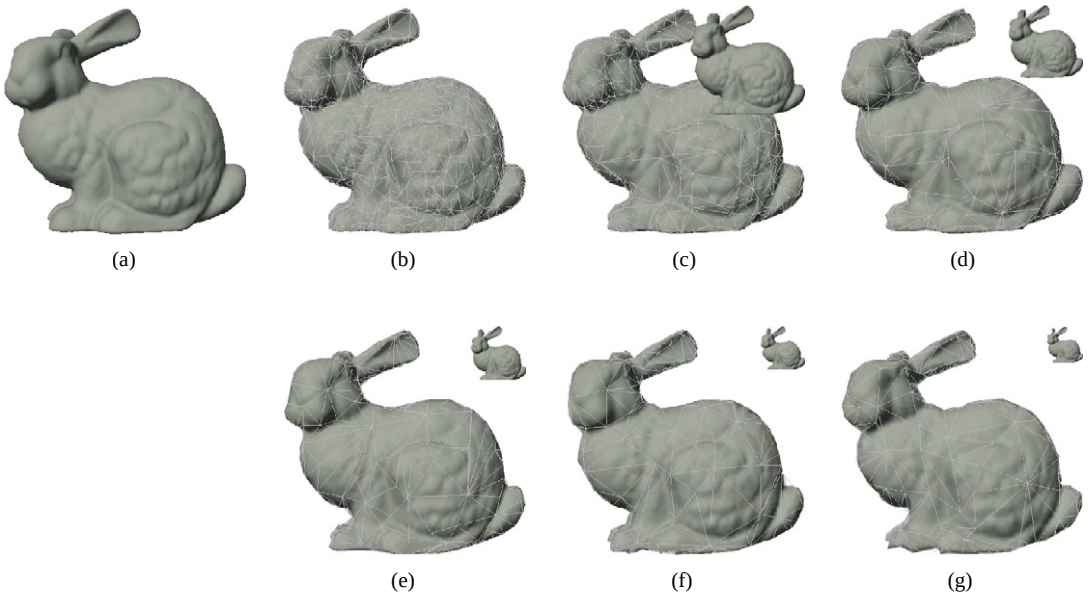


Fig. 6. (a) is the original mesh (65, 491 polygons) of a bunny viewed at one cell away (cell size 50). (b–g) are MVMeshes of the bunny at 1(1,605 polygons), 2(945), 3(554), 4(392), 5(330), and 6(306) cells away. The upper-right indicates actual projected images.

object, however, varies when we navigate in the cell. Fig. 6 depicts the MVMeshes of the bunny model.

3.1.5. 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 the C_i's are the corners of C. A simple proof for the 2D case is as follows: If a polygon P is back-facing with respect to both A and B, P's normal will be bounded in the dark green area, as shown in Fig. 7(a). Given a point G on the line $\overline{AB}$, vector $\overrightarrow{GP}$ is bounded by $\overrightarrow{AP}$ and $\overrightarrow{BP}$. As a result, P is shown to be back-facing with respect to G. An interior point I of the cell C is on a line $\overline{C_iE}$, for some i, and E on $\overline{C_jC_{(j+1)\bmod 6}}$ for some j. Since P is back-facing with respect to all corners, P is back-facing with respect to E and therefore I; (see Fig. 7(b)).

3.1.6. 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

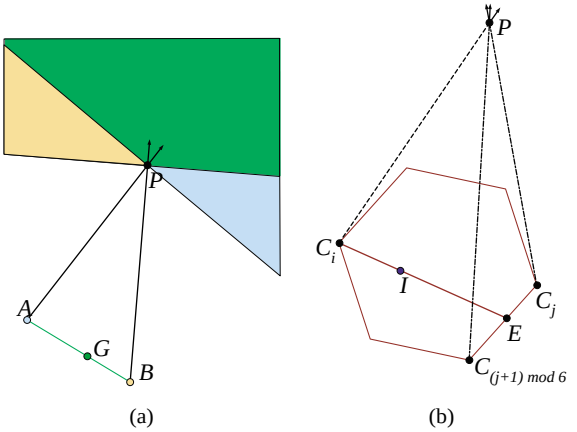


Fig. 7. Regional back-face culling.

amounts to the coloring problem, and itself is an NP-complete problem. Before getting into the details of the proposed greedy approach, several terms are first described.

- *Cluster-able*: Object or cluster M is cluster-able with cluster C if the texture size of $M \cup C$ is less than the sum of texture sizes of M and C, and self-occluding-error$(M \cup C) < T_s$.
- *Overlapping size*: Overlapping size of an object M and a cluster C is the size of the intersection of projected areas of M and C.

The greedy approach 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. 8(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 $M \cup C_1$, however, is no longer cluster-able with C_2; but still cluster-able with C_3 (see Fig. 8(b)). Finally, M is clustered with C_1 and C_3 (see Fig. 8(c)).

The clustering is performed after the self-occlusion-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.

3.1.7. 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 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 following steps when navigating inside the cell: (see also the followchart in Fig. 9.)

(1) Ensure that the geometry and image data for the current navigation cell has been loaded into memory.
(2) Perform window culling and view-frustum culling for the whole scene.
(3) (Optional) Perform a run-time occlusion culling for all meshes.
(4) (Optional) Perform a run-time back-face culling for the meshes inside the current cell.
(5) Render the meshes outside the current cell using *projective texture mapping*, followed by rendering meshes inside the cell as normal.
(6) Prefetch data for neighboring cells when the CPU load is relatively low.

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 precalculation 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 [33] while the rendering of MVMeshes involves texture blending as part of view-dependent projective texture mapping [34]. As mentioned previously in MVMesh derivation, a simplified polygon is associated with a set of views to which it is visible. If the set is empty, normal map approach [18] can be applied to that polygon. If the set contains only one view, then the polygon is rendered by standard projective texture mapping. If the set contains three or more views, two views are chosen from the set according to the vector defined from the viewer to the polygon. The textures corresponding to these two views are then mapped onto the polygon using projective texture mapping with blending.

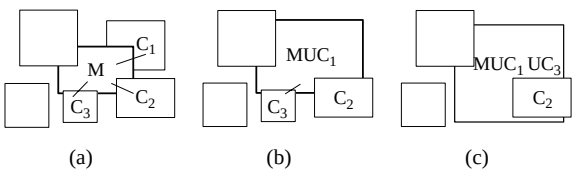


Fig. 8. Repeat clustering.

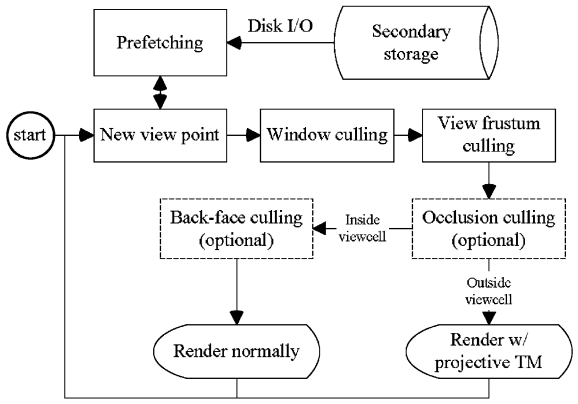


Fig. 9. Run-time phase.

One of the major problems arising 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 relatively 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.

The proposed scheme runs in a lower priority thread and prefetches data of neighboring cells that will be possibly visited in a short time. We set a *timestamp* for the navigation system. The timestamp is initially 0 and gets increased by 1 whenever the viewer has moved by a predefined distance or have turned by a predefined angle. When the timestamp gets increased by 1, we identify those cells that need to be prefetched and add them to a priority queue that maintains those cells waiting for prefetching, and then begin the prefetching. The cell is added to the priority queue according to its *t-priority*, which is the sum of the current timestamp and a priority value. The addition of timestamp in the *t-priority* allows us to distinguish the freshness of the cells in the priority queue. The priority value of a cell conceptually indicates how urgent it is for prefetching and is in the range of [0, 1). In principle, cells that are within the view frustum get higher priorities than those outside, the cells closer to the aiming vector get higher priorities, and cells closer to the viewpoint get higher priorities. The prefetching repeatedly removes the cell of the highest *t-priority* from the priority queue and loads the cell's data from disk to main memory, and removes those cells which are out of date by checking if *t-priority* values are smaller than the current timestamp minus 2.

Disk I/O can run in parallel; however, the system bus that loads texture data from main memory to texture memory can hardly run in parallel. To this end, textures that have been loaded from disk to main memory are put into a *texture queue* and get loaded in FIFO order. The loading of texture from main memory to texture memory runs in main thread, in which an amount of texture constrained by a budget is loaded before each frame. One practical concern is that the size of texture varies a lot. A texture that is not the first in the texture queue and is of size larger than the remaining budget is put back as the first in the texture queue.

4. Experiments

4.1. Setup

The test platform is a PC with an AMD AthlonXP 1800+CPU, 512 MB main memory, and an *n*VIDIA GeForce4 Ti 4400 with 128 MB 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).

For efficiency consideration, polygons and objects are represented by vertex IDs and object IDs, respectively. The original meshes are loaded into main memory before the navigation. In prefetching objects, SVMeshes, and MVMeshes, only their object IDs and vertex IDs are loaded.

4.1.1. 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 2 lists data statistics for the objects that compose the scenes, including polygon number, dimension, and distribution of polygon numbers for the scenes. A bird's eye view of the 8M-scene is shown in Fig. 10.

4.1.2. 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 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 navigation path shown in red in Fig. 10 with a maximum speed of 30/s., a maximum rotation of 45°/s., and an FOV of 60°.

Table 2
Object and scene statistics

Object name	Polygon no.	Dimension ($w \times d \times h$)	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

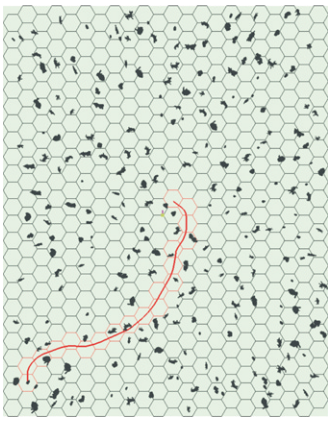


Fig. 10. A Bird's eye view of the 8M-scene.

4.2. Image quality measurement

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

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

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 [35] is

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

4.3. Mesh simplification

4.3.1. Self-occluding-error tolerance

The value of self-occluding-error tolerance T_s determines the distribution of SVMesh and MVMesh. For the scene 4M-50-T_s-4.5, where $T_s = 3, 5, 7$, Table 3 shows the averaged percentages of objects that are represented by SVMesh and MVMesh and their averaged polygon counts over all cells. Larger T_s implies higher percentage of SVMesh, more objects are clustered, higher simplification rate, less texture size, and finally higher frame rate. Note that numbers in the parenthesis under Avg. polygon count inside view frustum are Avg. polygon count for SVMesh and MVMesh inside view frustum. Fig. 11 depicts the distribution of SVMesh and MVMesh for the particular cell at the scene's center, on which MVMeshes are colored in blue, SVMeshes from single objects are in purple, and SVMeshes from clustered objects are in other colors.

4.3.2. Projected edge-length tolerance

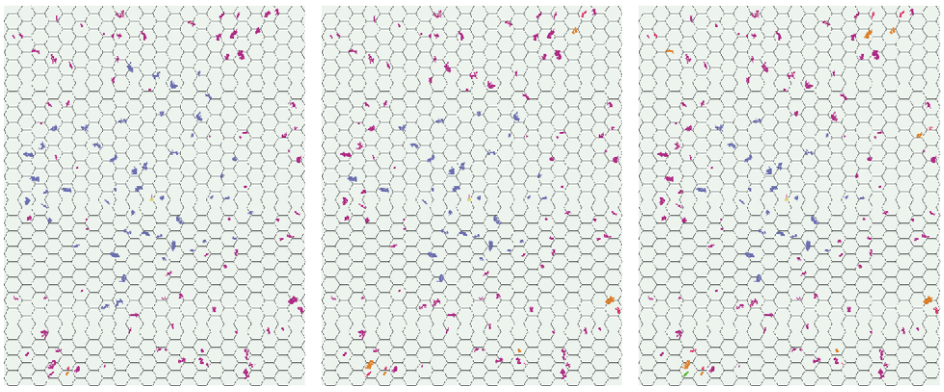
Through projected edge-length tolerance T_l, the edge collapsing can be tested for termination. Fig. 12 shows the MVMeshes of bunny derived by setting $T_l = 3.0$, 4.5, 6.0. Table 4 depicts the average polygon counts of SVMesh and MVMesh and simplification ratio for all cells. As we can see, larger T_l implies higher simplification rate, larger texture size, and finally higher frame rate.

4.3.3. Cell size consideration

Setting an optimal cell size is in general difficult. To test the effect of cell size, we continue to use the same

Table 3
Simplification performance under different self-occluding-error tolerance T_s

4M-50-T_s-4.5	$T_s = 3$	$T_s = 5$	$T_s = 7$
Statistics for object's represenatations and polygon counts			
Avg. percentage of SVMeshes from clustered objects (%)	9.8	18.7	25.5
Avg. percentage of SVMeshes from a single objects (%)	64.8	64.9	62.4
Avg. percentage of MVMeshes (%)	25.3	16.3	12.1
Avg. polygon no. inside a viewcell	9,308	9,308	9,308
Avg. polygon no. for SVMesh & MVMesh	40,742	39,981	39,360
Avg. polygon no. for a viewcell	50,050	49,290	48,669
Simplified:original	1:83.7	1:85.0	1:86.1
Performance statistics			
Avg. FPS	271.5	278.0	281.9
Avg. PSNR (dB)	39.64	39.54	39.46
Avg. texture size (KB) inside view frustum	592.0	544.1	516.6
Avg. polygon count inside view frustum	13,235 (11,532)	13,102 (11,418)	13,026 (11,367)



(a) (b) (c)

Fig. 11. Distribution of SVMesh and MVMesh for the scenes 4M-50-T_s-4.5: (a) $T_s = 3$; (b) $T_s = 5$; and (c) $T_s = 7$.

(a) (b) (c)

Fig. 12. MVMeshes of bunny for different T_l: (a) $T_l = 3.0$ (2,353 polygons), (b) $T_l = 4.5$ (1,605), and (c) $T_l = 6.0$ (1,227).

Table 4
Simplification performance under different projected edge-length tolerance T_l

4M-50-5-T_l	$T_l = 3.0$	$T_l = 4.5$	$T_l = 6.0$
Statistics for polygon counts			
Avg. polygon no. inside a viewcell	9,308	9,308	9,308
Avg. polygon no. for SVMesh & MVMesh	46,249	39,981	35,226
Avg. polygon no. for a viewcell	55,557	49,290	44,535
Simplified:original	1:75.4	1:85.0	1:94.1
Performance statistics			
Avg. FPS	265.4	278.0	289.2
Avg. PSNR (dB)	39.98	39.54	39.13
Avg. texture size (KB) inside view frustum	538.4	544.1	545.4
Avg. polygon count inside view frustum	14,741 (13,008)	13,102 (11,418)	11,701 (10,048)

data set and set $T_s = 5$ and $T_l = 4.5$ for cell sizes 50 and 100. Table 5 depicts the average polygon counts of SVMesh and MVMesh and simplification ratio for all cells. Larger cell size in general results in smaller simplification ratio and, in turn, lower frame rate, since the number of polygons inside a cell may increase dramatically.

4.4. 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.

Table 5
4M scene under different cell sizes 50 and 100

Cell size	50	100
Viewcells	22×24	11×12
Statistics for polygon counts		
Avg. polygon no. inside a viewcell	9,308	38,986
Avg. polygon no. for SVMesh & MVMesh	39,981	45,356
Avg. polygon no. for a viewcell	49,290	84,342
Simplified:original	1:85.0	1:49.7
Performance statistics		
Avg. FPS	278.0	255.5
Avg. PSNR (dB)	39.54	39.62
Avg. texture size (KB) inside view frustum	544.1	387.9
Avg. polygon count inside view frustum	13,102 (11,418)	17,457 (12,319)

- **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. All simulations follow the navigation path shown in Fig. 10.

Table 6 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 37.34 dB.

Fig. 13 represents the images rendered at views that are far from the cell center by configurations **B** and **C**. In Figs. 13(c) and (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 7 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.

Fig. 14 shows the run-time statistics of running configuration **C** on the scene 8M-50-5-4.5. In Fig. 14(a), 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. Along this particular navigation path, among 7 cells that contain objects, two of them contain the massive models such as bunny and dragon, respectively, as shown also in the plot for polygon count. Polygon count, texture size, and PSNR follow the FPS plots which are shown in Fig. 14(b). Note that most frames have PSNR above 37 dB.

Table 6
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

4.5. Discussions

On problems and potential of the proposed hybrid rendering scheme, we address the storage requirement and loading time for very complex scenes. Since each polygon and each object are presented by vertex IDs and object ID, respectively, original meshes must be available in main memory during navigation, and meshes are prefetched by loading their vertex IDs. Original meshes of 2M-, 4M- and 8M-scenes account for 74, 141, and 261 MB main memory, respectively. Taking into account the prefetched data as well, the main memory requirement is 82, 153, and 287 MB for 2M-, 4M- and 8M-scenes respectively.

As shown in Table 8, storage requirement for 2M-, 4M-, and 8M-scene are 311, 659, and 1260 MB, respectively, which are roughly 11 times that for original geometries. For each cell, the average size of polygons

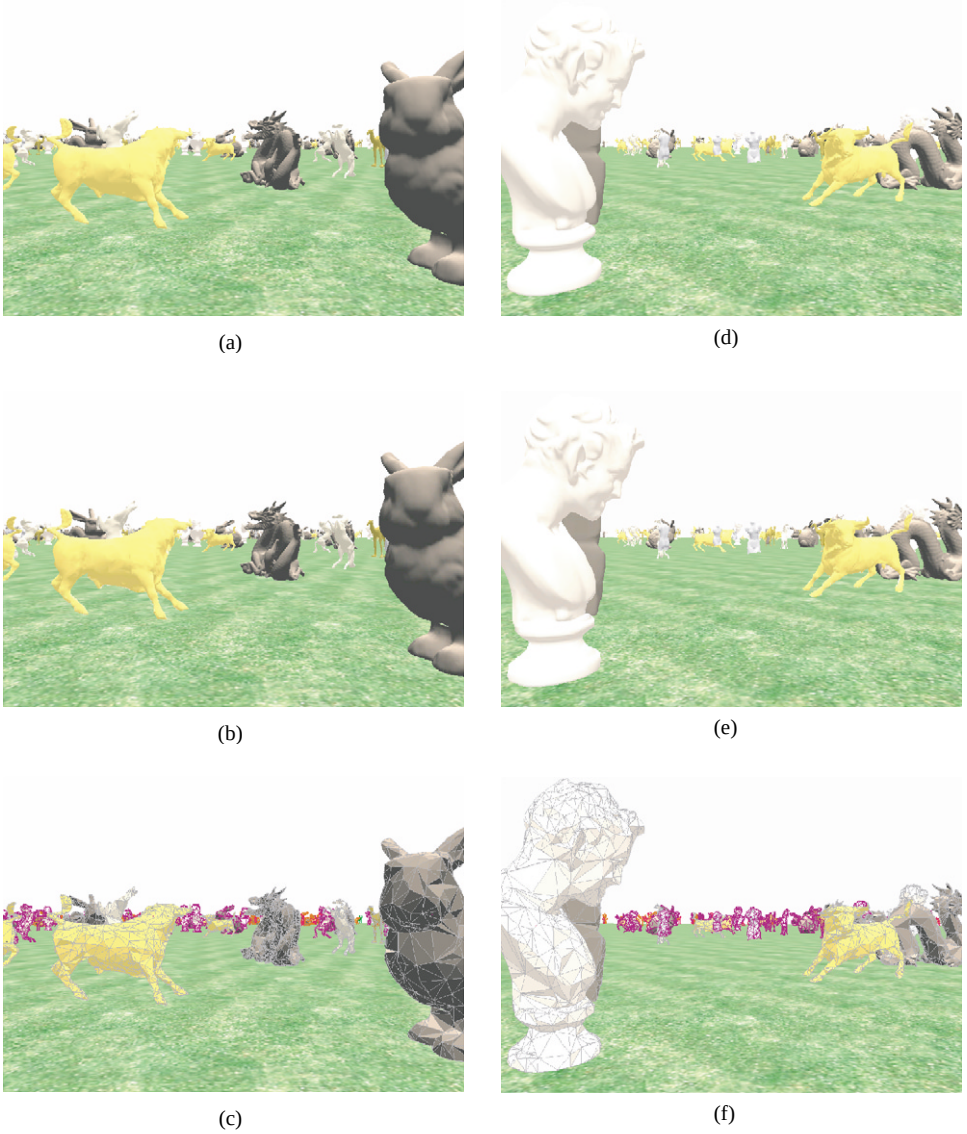


Fig. 13. Rendered images by configuration **B** and **C**: (a) Configuration **B**: 5,207,350 polygons in view frustum; (b) Configuration **C**: 35,216 polygons, PSNR 35.33 dB, 178.8 FPS; (c) Configuration **C**: flat shaded with wireframes; and (d) Configuration **B**: 4,693,355 polygons in view frustum; (e) Configuration **C**: 29,641 polygons, PSNR 36.39 dB, and 199.3 FPS; (f) Configuration **C**: flat shaded with wireframes.

and textures that needs to be in memory is 553, 1165, and 2252 KB for 2M-, 4M-, and 8M-scene, respectively. Let us take as an example the hard disks having reading speeds at 35 MB/s. For the 8M-scene, the loading time for each cell requires 64.4 ms. Under the assumption that the maximum navigation speed is 30/s and the cell size is 50, the time between cell transitions will be 2887 ms for the path 1 shown in Fig. 15. The ratio of the average loading time for a cell over the time between transitions is only 2.23%. On the other hand, for the case of path 2, the ratio is 4.46%. Several implementation details can be included to smooth out the loading time. First of all, the loading can be easily amortized into in-cell frames without notice since disk I/O can run in parallel. Secondly, when navigating in high speed, user perception is more sensitive to smooth frame rate than image quality. In this case, the texture can be mapped with lower resolutions, which implies smaller size for loading.

In the experiment, we found that textures can only be loaded from the hard disk to main memory, and then to texture memory. Transferring data to texture memory, however, has to compete with the data transferring

Table 7
Performance of configuration **C** for 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)

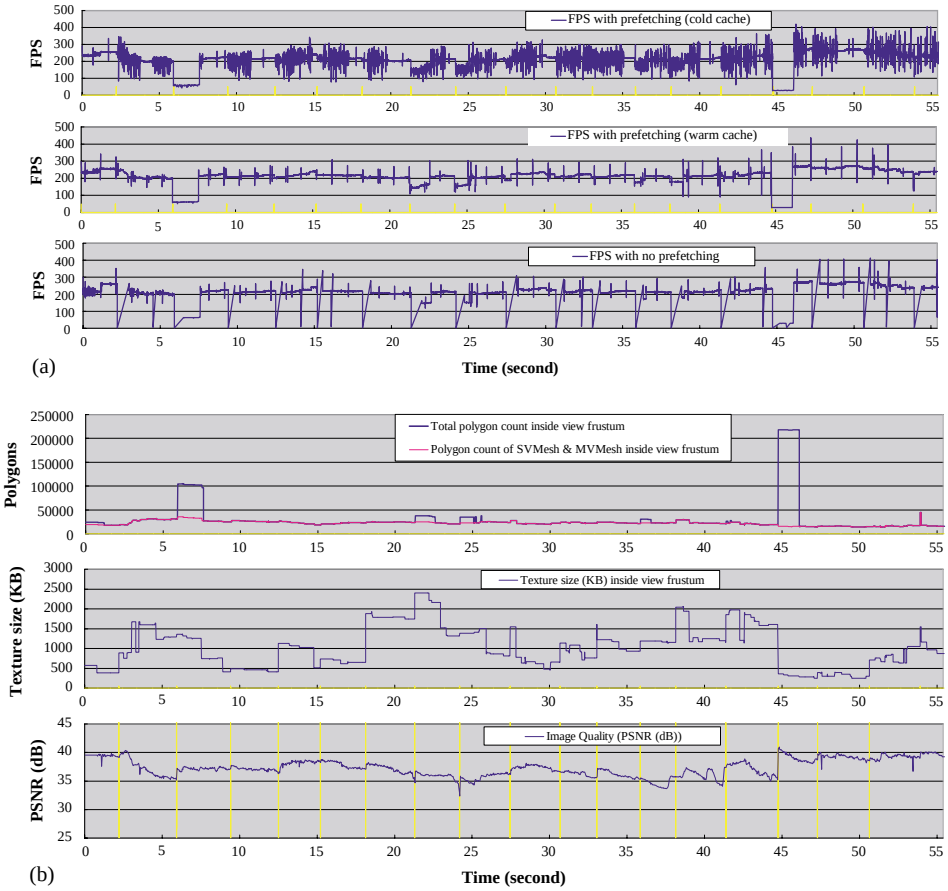


Fig. 14. Run-time statistics of configuration **C** on scene 8M-50-5-4:5: (a) the frame rates with prefetching under a cold cache and a warm cache, and without prefetching and (b) the polygon count, texture requirements, and image quality.

between CPU and graphics accelerator. Another problem appeared is that textures in general have size in a wide range, for example, from several bytes to hundred kilobytes. In consequence, the prefetching of textures cannot be easily amortized effectively to in-cell frames.

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

Table 8
Storage and loading time for different scene complexities ($T_s = 5$, $T_l = 4.5$)

Scene complexity	2M	4M	8M
Total secondary storage requirement (*MB*)			
SVMesh & MVMesh	101.4	218.6	408.8
Textures	209.3	440.5	850.8
Total	310.7	659.1	1,260
Original meshes	27.8	59.6	112.6
Average run-time requirement (*KB*) *for a cell*			
Potentially visible polygons	71.4	154.2	287.2
Textures	481.2	1,011	1,965
Total	552.6	1,165	2,252
Average loading time (*ms*) *for a cell*			
Hard disk (55 MB/s)	10.0	21.2	41.0
Hard disk (35 MB/s)	15.8	33.3	64.4

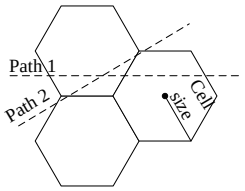


Fig. 15. Cases of cell transition.

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 neighboring 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/s with a little loss of image quality (average PSNR 37.34 dB). The polygons and textures require about 1260 MB secondary storage space and about 294 MB main memory on average.

References

[1] Shade JW, Gortler SJ, He L-W, Szeliski R. Layered depth images. In: Proceedings of SIGGRAPH '98, Orlando, Florida, July 1998. p. 231–42.

[2] Kumar S, Manocha D, Garrett B, Lin M. Hierarchical back-face culling. In: Proceedings of the Seventh Eurographics Workshop on Rendering, Porto, Portugal, 1996. p. 231–40.

[3] Zhang H, Hoff III KE. Fast backface culling using normal masks. In: Proceedings of 13th Symposium on Interactive 3D Graphics, Providence, Rhode Island, 1997. p. 103–6.

[4] Hudson T, Manocha D, Cohen J, Lin M, Hoff K, Zhang H. Accelerated occlusion culling using shadow frusta. In: Proceedings of the 13th Symposium on Computational Geometry, Nice, France, 1997. p. 1–10.

[5] Greene N, Kass M, Miller G. Hierarchical Z-buffer visibility. In: Kajiya JT, editor. Computer Graphics (SIGGRAPH '93 Proceedings), ACM Press, Anaheim, August 1993. p. 231–38.

[6] Zhang H, Manocha D, Hudson T, Hoff K. Visibility culling using hierarchical occlusion maps. Computer Graphics 1997;31:77–88.

[7] Cohen-Or D, Fibich G, Halperin D, Zadicario E. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. Computer Graphics Forum 1998;17(3):243–53.

[8] Durand F, Drettakis G, Thollot J, Puech C. Conservative visibility preprocessing using extended projections. In: Akeley K, editor. Computer Graphics (SIGGRAPH 2000 Proceedings), July 2000. p. 239–48.

[9] Schaufler G, Dorsey J, Decoret X, Sillion FX. Conservative volumetric visibility with occluder fusion. In: Akeley K, editor. Computer Graphics (SIGGRAPH 2000 Proceedings), New Orleans, Louisiana, July 2000. p. 229–38.

[10] Rossignac J, Borrel P. Multi-resolution 3-D approximations for rendering complex scenes. In: Falcidieno B, Kunii TL, editors. Modeling in Computer Graphics. Springer: Berlin, 1993. p. 455–65.

[11] Zhou K, Pan Z, Shi J. A new mesh simplification algorithm based on vertex clustering. Chinese Journal of Automation 1999;25(1):1–8.

[12] Pan Z, Zhang M, Zhou K, et al. Level of detail and multi-resolution modeling for virtual prototyping. Internal Journal of Image and Graphics, 2001;1(2):329–43.

[13] Schroeder WJ, Zarge JA, Lorensen WE. Decimation of triangle meshes. Computer Graphics, Chicago, Illinois, 1992;26(2):65–9.
[14] Hoppe H. Progressive meshes. In: Rushmeier H, editor. Proceedings of SIGGRAPH '96, Addison Wesley, New Orleans, August 1996. p. 99–108.
[15] Yang S-K, Chuang J-H. Material-preserving progressive mesh using vertex collapsing simplification. Journal of Virtual Reality, to appear.
[16] Hoppe H. View-dependent refinement of progressive meshes. In: Proceedings of SIGGRAPH '97. Los Angeles, California, 1997. p. 189–98.
[17] Xia J, Varshney A. Dynamic view-dependent simplification for polygonal models. In: Proceedings of IEEE Visualization '96, San Francisco, California, October 1996. p. 327–34.
[18] Cignoni P, Montani C, Rocchini C, Scopigno R. A general method for recovering attribute values on simplified meshes. In: Proceedings of IEEE Visualization '98, Research Triangle Park, North Carolina, October 1998. p. 59–66.
[19] Sander PV, Gu X, Gorlter SJ, Hoppe H, Snyder J. Silhouette clipping. In: Proceedings of SIGGRAPH 2000, New Orleans, Louisiana, July 2000. p. 327–34.
[20] Chen SE, Williams L. View interpolation for image synthesis. In: Kajiya JT, editor. Computer Graphics (SIGGRAPH '93 Proceedings), ACM Press, Anaheim, August 1993. p. 279–88.
[21] McMillan L, Bishop G. Plenoptic modeling: an image-based rendering system. In: Cook R, editor, Proceedings of SIGGRAPH '95, ACM SIGGRAPH, Addison-Wesley, Reading, MA, August 1995. p. 39–46.
[22] Gortler SJ, Grzeszczuk R, Szeliski R, Cohen MF. The lumigraph. In: Rushmeier H, editor. Proceedings of SIGGRAPH '96, ACM SIGGRAPH, Addison-Wesley, Reading, MA, August 1996. p. 43–54.
[23] Levoy M, Hanrahan P. Light field rendering. In: Rushmeier H, editor. Proceedings of SIGGRAPH '96, Addison Wesley, New Orleans, August 1996. p. 31–42.
[24] Schaufler G, Stürzlinger W. A three-dimensional image cache for virtual reality. In: Proceedings of Eurographics '96, Poitiers, France, August 1996. p. 227–36.
[25] Shade J, Lischinski D, Salesin DH, DeRose T, Snyder J. Hierarchical image caching for accelerated walkthroughs of complex environments. In: Rushmeier H, editor. Proceedings of SIGGRAPH '96, Addison Wesley, New Orleans, August 1996. p. 75–82.
[26] Darsa L, Silva BC, Varshney A. Navigating static environments using image-space simplification and morphing. In: Proceedings of the 13th Symposium on Interactive 3D Graphics, Providence, Rhode Island, 1997. p. 25–34.
[27] Darsa L, Costa B, Varshney A. Walkthroughs of complex environments using image-based simplification. Computers & Graphics, 1998;22(1):55–69.
[28] Sillion F, Drettakis G, Bodelet B. Efficient impostor manipulation for real-time visualization of urban scenery. In: Proceedings of Eurographics '97, Budapest, Hungary, September 1997. p. 207–18.
[29] Decoret X, Schaufler G, Sillion FX, Dorsey J. Multi-layered impostors for accelerated rendering. Computer Graphics Forum, Blackwell Publishers, Milano, 1999;18(3):61–73.
[30] Aliaga D, Cohen J, Wilson A, Baker E, Zhang H, Erikson C, Hoff K, Hudson T, Stuerzlinger T, Bastos R, Whitton M, Brooks F, Manocha D. MMR: an interactive massive model rendering system using geometric and image-based acceleration. In: Proceedings of 1999 ACM Symposium on Interactive 3D Graphics, Atlanta, Georgia, 1999. p. 199–206.
[31] Zhang M, Pan Z, Heng P-A. A near constant frame-rate rendering algorithm based on visibility computation and model simplification. In: Proceedings of VSMM 2002, Gyeongju, Korea, 2002. p. 387–98.
[32] Chen C-C, Chuang J-H. Viewcell-dependent geometry simplification using depth. In: Computer Graphics Workshop 2002, Taiwan, June 2002.
[33] Segal M, Korobkin C, Widenfelt R, Foran J, Haeberli P. Fast shadows and lighting effects using texture mapping. In: Computer Graphics (SIGGRAPH '92 Proceedings), July 1992. p. 249–52
[34] Debevec PE, Taylor CJ, Malik J. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In: Rushmeier H, editor. Proceedings of SIGGRAPH '96, Addison Wesley, New Orleans, August 1996. p. 11–20.
[35] Gonzalez RC, Woods RE. Digital image processing, Addison-Wesley, Reading, MA, September 1993, p. 228. [chapter 4].