Rendering complex scenes based on spatial subdivision, object-based depth mesh, and occlusion culling

Chih-Chun Chen, Bo-Yin Lee, Jung-Hong Chuang, Wei-Wen Feng, Ting Chiou*
Department of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan

ABSTRACT

In this paper, we combine geometry-based and image-based rendering techniques to develop a VR navigation system that aims to have efficiency relatively independent of the scene complexity. The system has two stages. In the pre-processing stage, the *x-y* plane of a 3D scene is partitioned into equal-sized hexagonal navigation cells. Then for each navigation cell, we associate each object outside the cell with either a LOD mesh or an object-based depth mesh depending on its self-occlusion error. The object with the error larger than a user-specified threshold is associated with a LOD mesh of an appropriate resolution. For the object with error smaller than the threshold, we associated it with a depth mesh that is reduced from its original mesh based on the silhouette and depth information of its image rendered from the cell center. All LOD meshes are then culled by a conservative back-face computation, and then all LOD and depth meshes are culled by a conservative visibility computation, all aim to remove polygons that are invisible from any point inside the cell. At run-time stage, LOD meshes are rendered normally while depth meshes are rendered by texture mapping with their cached images. Techniques for run-time back-face culling and occlusion culling can be easily included. Our experimental results have depicted fast frame rates for complex environments with an acceptable quality-loss.

Keywords: Virtual Reality, Visibility, Image Caching, Image-Based Rendering, Hybrid Rendering

1. INTRODUCTION

In order to achieve an immersive visual effect during the VR navigation, rendering with photo-realistic scene images and high frame rate has been our ultimate goal. However, in the traditional geometry-based rendering, complex scenes always require numerous polygons, and, therefore, can be rendered at an unacceptably low 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 complex scenes, including visibility culling, level-of-detail (LOD) modeling, and more recently, image-based rendering. Although image-based rendering scheme is capable of rendering complex scenes with photo-realistic images in the time complexity that is independent of the scene complexity, it has been suffered from the limited viewing degree of freedom and some losses of image quality due to gaps and holes. Hybrid rendering that combines geometry- and image-based technique has become a viable alternative.

To take the advantages of both geometry- and image-based rendering techniques, we introduce a hybrid rendering scheme that aims to render a complex scene in a constant and high frame rate with only a little or an acceptable quality loss. The hybrid scheme consists of two stages: pre-processing stage and run-time stage. In the pre-processing stage, to exploit the spatial coherence, the *x-y* plane of a 3D scene is partitioned into equal-sized hexagonal navigation cells. To reduce hole problem due to self-occluding, each object outside a cell is represented either by a LOD mesh or by a *depth mesh* depending on its approximate *self-occluding error*. The object is represented by a LOD mesh of an appropriate resolution if its self-occluding error is over a user-specified tolerance; otherwise by an object-based depth mesh. The object-based depth mesh is derived from the object's original mesh based on the silhouette and depth variation on the rendered image viewed from the cell center. The resulting depth mesh is a view-dependent LOD model of the object's visible part that the resolution becomes coarser when the object is at farther distance while the silhouette is well preserved. In consequence, for each navigation cell, we have a set of LOD meshes and depth meshes (together with their cached images) for objects outside the cell. Since viewpoint is constrained in a particular cell during the navigation,

* Further author information:
All: Email: {chihchun, bylee, jhchuang, fengww, tchiou}@csie.nctu.edu.tw, address: Department of Computer Science and Information Engineering National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu, Taiwan 30050, ROC.

Third International Conference on Virtual Reality and Its Application in Industry,
Zhigeng Pan, Jiaoying Shi, Editors, Proceedings of SPIE Vol. 4756 (2003)

we can further remove back-facing polygons of LOD meshes and occluded polygons for both types of meshes for any viewpoint inside the cell. This is accomplished by the proposed conservative back-facing culling and a conservative occlusion culling.

At the run-time stage, LOD meshes and depth meshes associated with the navigation cell where current view-point lies are rendered. The depth meshes are texture mapped with the cached images by the proposed hardware accelerated *projective-alike texture mapping* which generates texture coordinates automatically. Run-time occlusion culling for the entire scene and back-facing culling for the objects inside the cell can be performed to further reduce the polygon count. To minimize the impact of the data loading while navigating across the cell boundary, a pre-fetching scheme is also developed to amortize the loading time to several previous frames.

2. RELATED WORK

View frustum culling that prevents the objects outside the view volume from being sent to the rendering pipeline is the basic visibility culling. Furthermore, back-face culling is also commonly used. A sub-linear algorithm[10] has been proposed for culling back-facing polygons, but it suffers from the requirements of model partitioning. Zhang and et al.[17] improved this by introducing *normal mask* which reduces the per polygon back-face test to only one logical AND operation.

Several run-time methods have been proposed to further cull out polygons that are occluded by others. There include occlusion culling using hierarchical Z-buffer[8], and hierarchical occlusion map[18]. However, there are inevitable overheads doing occlusion culling at run-time. Cohen-Or and et al.[2] proposed a preprocessing algorithm for regional occlusion culling, but its performance depends heavily on a single strong occluder. Durand and et al.[5] proposed *extended projection* operations to handle occluder fusion of multiple occluders. It subdivides a scene into volumetric cell, and computes potential visibility set (PVS) for each cell.

With those visibility culling techniques, the remaining polygons might be still too many to achieve interactive rate. Level-of-detail (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 methods have been proposed to obtain LOD meshes; e.g., *edge collapsing*[9], *vertex clustering*, *vertex decimation*, and etc, ...

Geometry-based rendering based on visibility culling and LOD modeling alone usually still cannot meet interactive requirement for very complex scenes. Image-based rendering (IBR) has been a well known alternative. IBR takes parallax into account, and renders a scene by interpolating neighboring reference views[1,13]. 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. IBR in general has problems like folding, gap, and hole. Lumigraph[7] and light field rendering[11] have been proposed to reduce the 7D *plenoptic function* $P(\theta,\psi,\lambda,V_x,V_y,V_z,t)$ to the 4D function *P'(s, t, u, v)* for static scenes. However, both require enormous storage for the extremely large number of images. Layered depth image (LDI)[15] 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.

Hierarchical image caching[14] is the first approach that combines geometry- and image-based rendering aiming to achieve an interactive frame rate for complex static scenes. The cached texture possesses no depth information and, in turns, limits its life cycle. The image simplification schemes[3,16] represent background or distant scene using 2D cached depth meshes derived from the rendered images for some specific views. 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 visibility and self-occluding still remain. Moreover, disjointed objects might be rendered as connected objects, and depth meshes derived on the 2D cached images are in pixel resolution, which might lead to geometric inaccuracy when re-projected into 3D space. Multi-layered impostors[4] are proposed to restrict visibility artifacts between objects to a given size, and as well as a dynamic update scheme to improve the resolution mismatch. However, it still encountered hole problem due to self occlusion, and an efficient dynamic update requires a special hardware architecture.

3. PROPOSED HYBRID SCHEME

A scene represented by traditional image-based representation would produce a lot of artifacts such as cracks and holes due to resolution mismatch and visibility change respectively. However, a scene represented by a single environment-based depth mesh (impostor) still produces rubber artifacts caused by incorrect connections between disjoint surfaces. Although, disjoint surfaces are identified, holes still appear in a new view caused by the lack of information of occluded parts. Such that, we propose an *object-based depth meshes* scheme to greatly reduce the hole artifacts produced from the occlusion between objects, as well as a *self-occlusion error estimation* to restrict the hole artifacts produced from object's self-occlusion in a given size. Such estimation decides the representation of objects. Those objects which will potentially result in holes smaller than a user-specified tolerance are represented by depth meshes; otherwise, by standard meshes of appropriate LOD resolutions.

To reduce the redundancy of a regular-grid meshing, as well as the precision error caused by the projection from object space to image space, the *depth mesh* is simplified by an edge collapsing from original mesh based on the depth characteristic while preserving most of the important visual appearances.

Furthermore, the regional back-face culling for each object from a cell could be performed before the self-occluding error estimation, and depth mesh construction as well. Such that, more objects are represented by depth mesh under the same self-occluding error tolerance, and depth mesh construction would be much faster. Lastly, a conservative occlusion culling can remove invisible polygons for any view inside the cell of all depth meshes and LOD meshes. In a result, the polygon count in scene navigation can be greatly reduced.

3.1. Pre-Processing Stage

The processing steps in the pre-processing stage are:

1. Hexagonal spatial subdivision.
2. Regional conservative back-face culling.
3. Selection of object's representation based on its self-occluding error estimation.
4. Depth mesh construction.
5. LOD mesh generation.

3.1.1. Hexagonal spatial subdivision

In order to utilize the spatial locality of a complex scene, we subdivide the x-y plane of the scene into $N \times M$ hexagonal navigation cells (Figure 1(a)). With the spatial subdivision, the scene data and viewpoints can be localized to cells, and, therefore, visibility culling, conservative back-facing and occlusion culling can be performed in preprocessing phase. A reason why the hexagonal subdivision, rather than rectangular subdivision, is used is that, in worse case, data of three adjacent cells need to be loaded, instead of four for rectangular subdivision, when navigating across the cell boundary. Table 1 depicts the maximum ratio of side faces could be seen from an inside point of the hexagonal and rectangular cell under different FOVs, we can see that hexagonal subdivision is better than rectangular one in most cases, except that they are equal under the 45° case.

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

3.1.2. Regional conservative back-face culling

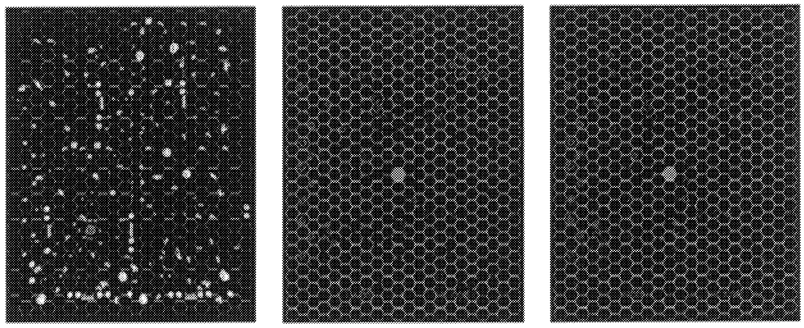
As shown in Figure 3(b), for each polygon, we obtain the vector from one of six corner vertices of the navigation cell to the center of polygon, and do the dot product of the vector with polygon's normal vector. If it is negative, the polygon is back-facing with respect to that corner vertex. If a polygon is back-facing for all six vertices of the cell, the polygon is back-facing wrt. any point inside the cell, and hence should be culled. In short, a polygon P is back-facing for a navigation cell C, if

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

where C_i's are the corners of the navigation cell. A simple proof for the 2D case is as follows:

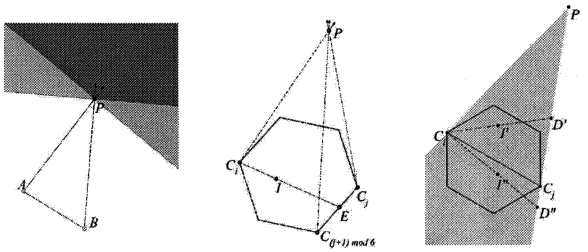
If a polygon P is back-facing wrt. both point A and B, P is back-facing wrt. any point on the line $\overline{AB}$ (see Figure 3(a).)
Any inner point I of a navigation cell is on a line $\overline{C_i E}$, where vertex E is on an edge $\overline{C_j C_{(j+1) \bmod 6}}$ (see Figure 3(b).)
Because, P is back-facing wrt. to all corners C_k, $k = 0, \ldots, 5$, P is back-facing wrt. E and therefore I.

Moreover, two corners are necessary and sufficient for this regional back-facing test for each polygon (see Figure 3(c).) For any point I' inside the navigation cell, $\overline{C_i I'}$ intersect $\overline{C_j P}$ at point D', because D' is on $\overline{C_j P}$ and P is back-facing with respect to both C_i and C_j; such that, P is back-facing with respect to D', as well as I'. Note that, each polygon would be required to test with different pair of corners.



(a) A bird's eye view (b) size 50, 1-pixel (c) size 50, 3-pixel

Figure 1. (a) shows the distribution of objects, (b) and (c) show the distribution of LOD meshes and depth meshes under two different self-occluding errors.



(a) P is back-facing with respect to any point on $\overline{AB}$, if P is back-facing wrt. to both A and B.

(b) P is back-facing wrt. any inner point I in a navigation cell, when P is back-facing wrt. all corners.

(c) P is back-facing wrt. to the navigation cell, if P is back-facing wrt. both C_i and C_j.

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

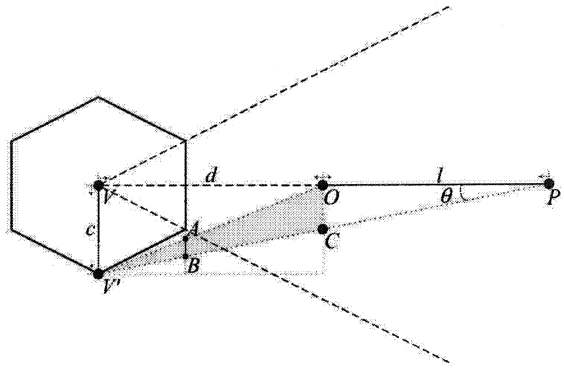


Figure 3. Regional back-face culling.

3.1.3. Self-occluding error estimation

The major problem of depth mesh representation is that it is only the visible part of the object viewed from the cell center hence has limited viewing degree of freedom. When a new view is far from the cell center, parts that are invisible originally might become visible and get rendered as holes. We propose a self-occluding error to estimate the maximum size of the hole that may appear when the object is represented by a depth mesh.

As shown in Figure 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 distant between object and the cell center, i.e., the length of $\overline{VO}$ be d, and the size in 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 $\theta = \tan^{-1}\frac{c}{d+l}$ and s which is the projected size of $\overline{OP}$ as well as $\overline{OC}$ is:

$$s = \frac{\overline{AB}}{c} * \text{ImageRes.}$$

Since

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

therefore,

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

The test on self-occluding error is to check if the object's maximum projected error s is smaller then a user-specified tolerance in image precision. If it is, the object is represented by a depth mesh; otherwise by a LOD mesh.

3.1.4. Depth mesh construction

The cached image of an object is obtained by rendering the object using the cell center as the center of projection and using the cell's side face as the projection window. The *depth image* is the cached image augmented with the depth values. The simplest way to construct a *depth mesh* for an object is to use the regular-grid triangulation[12] performed on the depth image, which would, however, results in too many redundant triangles and produces rubber effects caused by incorrect connections. Furthermore, since it is performed in the image space, it always suffers from the precision error caused by the projection (from floating-point precision in world space into integer precision in image space).

In order to reduce the number of the triangles on a depth mesh while preserving most of the visual appearances, several properties of the depth image could be adopted. The most important one is to use the depth coherence, by that we mean pixels of similar depth variation are likely to be on the same surface, and a pixel that has a sharp depth variation from adjacent pixels would have a high possibility to be on a contour edge. Moreover, the external contour edges of the rendered object on the image are the most important visual appearances, and hence must be included in the depth mesh. External contour edges can be easily derived by using the *contour extraction* in the field of image processing. On the other hand, if we can extract all the internal contour edges from the depth image, rubber effects caused by undetected gaps (C^0 discontinuity) between disjointed surfaces represented by a connected mesh and blur effects appearing at the sharp edges (C^1 discontinuity) represented by a flatted mesh could be greatly reduced.

In order to minimize the precision error caused by projection, we simplify the depth mesh in both the image and the object space in three steps. Firstly, we categorize image pixels on the depth image based on the importance of its visual appearance and its characteristic into four categories:

- *external contour*: a pixel on the external contour extracted by contour extraction.
- *internal contour* (gap) (C^0 discontinuity): a pixel whose next guessed Z value differs from neighboring pixels over a specified tolerance T_{C^0} (i.e., $|Z_{i+1} - (Z_i + (Z_i - Z_{i-1}))| > T_{C^0}$, see Figure 4.)[1]
- *sharp edge* (C^1 discontinuity): a pixel whose Z variation differs from neighboring pixels over a specified tolerance T_{C^1} (i.e., $|(Z_{i-1} - Z_i) - (Z_i - Z_{i+1})| > T_{C^1}$.)[2]
- *interior*: other pixel whose Z value is different from the background Z value.

Secondly, vertices of object's original mesh are projected again (but do not alter the value of depth image) with the same projection setup of the depth image to do the visibility test for each vertex. If Z value of a projected vertex equals to the Z value at the pixel on the depth image, the pixel on the depth image is representing the vertex of the original mesh; otherwise the vertex is behind other surfaces. For the former case, a weight rela-tive to the category of the pixel is assigned to the vertex (The highest weight is assigned for external contour category, a lower weight is assigned for internal contour category, and so on.) On the other hand, for the later case (invisible vertex), the vertex gets the weight zero.

Lastly, weight-based edge collapsing is performed based on the weights of vertices to simplify the object's original mesh. Moreover, to obtain a proper resolution of the simplified depth mesh and preserve the visual appearances, the edge with one of or both vertices' weight smaller than the weight of the sharp edge category and whose projected size smaller than a user-specified length tolerance (in pixels) is simplified. The collapsing order is based on the area of the

[1] If we use depths of only two neighboring pixels to test C^0 discontinuity, surfaces that are nearly parallel to the viewing direction would treated to be discrete pixels.

[2] Note that both equations for testing C^0 and C^1 discontinuity are equivalent, except that $T_{C^0} > T_{C^1}$.

projected triangle, i.e., a smaller projected triangle is simplified earlier. As a result, a more optimized triangle aspect ratio is obtained and tiny triangles with respect to the view contain no important visual appearances are removed. Figure 5(b-e) show the simplified depth meshes at different distances.

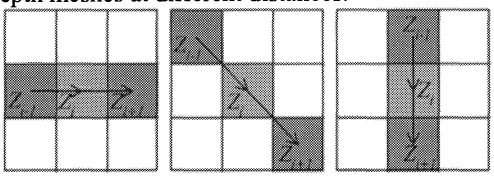
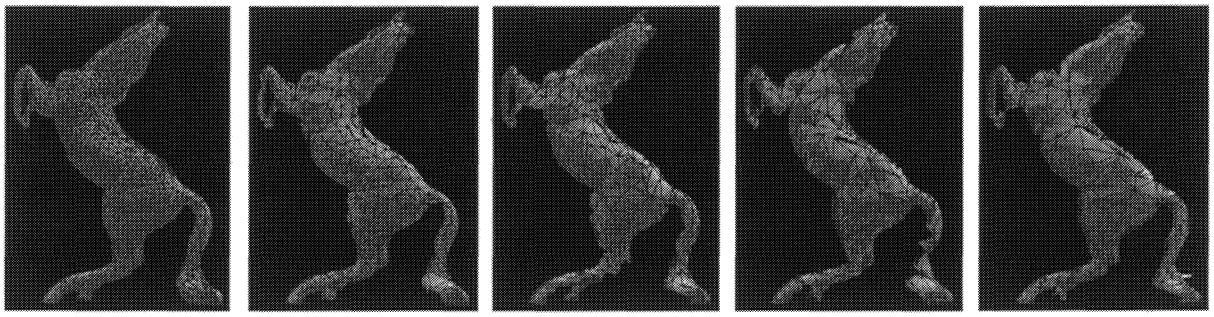


Figure 4. *Gap* and *sharp edge* extraction.



(a) 2 cells away. (b) 6. (c) 7. (d) 9. (e) 10.

Figure 5. (a) is the original mesh of a horse captured at two cells away (at distance 173), and (b-e) are depth meshes generated at 6, 7, 9 and 10 cells away.

3.1.5. Regional conservative occlusion culling

It is very common that objects or part of objects are occluded by other objects in a complex scene, especially a densely occluded environment. In order to utilize the spatial coherence of occlusion, we perform a regional conservative occlusion culling in the pre-processing stage for both LOD meshes and depth meshes. The extended projection[5] provides us a useful tool with a little modification for our hexagonal subdivision scheme. 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 size. Only those meshes whose projected sizes are larger than a user-specified threshold are selected to be occluders.

3.2. Run-Time Stage

In the run-time stage, we do the following steps:

1. At program start-up time, we setup a lowest priority thread for pre-fetching the geometry and image data of neighboring cells.
2. Ensure that the geometry and image data for the current navigation cell is loaded into memory.
3. Perform a run-time normal-cluster-based back-face culling for the objects inside the current navigation cell.
4. Perform a run-time occlusion culling for all meshes.
5. Render the remaining polygons. Depth meshes are rendered using the projective-alike texture mapping.
6. Pre-fetch data of neighboring cells when CPU load is relative low.

3.2.1. Rendering

Objects inside the navigation cell can be seen from any direction, it is impossible to determine the visibility during the pre-processing stage. Those polygons inside the navigation cell can be grouped into clusters according to its normal[10] in the pre-processing stage. During the run-time stage, we can quickly cull out the whole back-facing cluster of polygons according to the viewing direction and the FOV.

Although there are considerable overheads, it is a beneficial approach to reduce the polygons sent into graphic pipeline by applying a run-time occlusion culling for a densely occluded environment. To perform the culling, we generate an occlusion map similar to the idea proposed in Ref. 18. Only LOD meshes, depth meshes, and original meshes inside the navigation cell whose projected area larger than a pre-specified threshold are selected to be occluders.

Though this approach of occluder selection is not optimized, it is advantageous not to spend too much time on selecting occluders.

3.2.2. Projective-alike texture mapping

A *projective-alike-texture-mapping* method is developed to map the cached image onto depth mesh in such a way that the texture coordinate of each vertex can be generated automatically by the standard OpenGL (glTexGen().)

In most cases, an object-based depth mesh located in part of a source image. To minimize the storage requirement and to reduce the loading time of cached images. Only necessary rectangular part of source image is stored as cached image, and, in addition, S3's S3TC is applied to compress the cached images. Such that, more textures can put into the limited texture memory, bandwidth between host and graphics hardware can be greatly reduced, and a decompression is not required at run-time anymore.

Texture coordinates (s, t) mapped from image coordinates (x, y) can be derived by $(s, t) = ((x - x_{offset})/w_{stored}, (y - y_{offset})/h_{stored})$, where x_{offset} and y_{offset} are the offset of the cached image relative to source image, and w_{stored} and h_{stored} are width and height of the cached image, respectively. The GPU of graphics hardware can also do the transformation of re-projection from the source image coordinates to the destination image coordinates by multiply the inverse of the camera matrix of the source image, allowing CPU leisurely does the visibility culling, level-of-detail selection, pre-fetching, and so on.

This method does not need to specify texture coordinates, and in consequence, reduces the bandwidth needed between CPU and graphics accelerator, and does not require additional memory for storing texture coordinate at each vertex. Moreover, vertices of a depth meshes are allowed to be stored in source image coordinate system, which requires 16-bit unsigned integer for x and y, and 32-bit floating point for z (provides a tradeoff between storage requirement and precision). As a result, only 8 bytes is necessarily sufficient for each vertex, compared to 20 bytes per vertex if all x, y, and z are stored as world coordinates as well as texture coordinates s and t.

3.2.3. Cell transition

A major problem arises in spatial-subdivision approach is how to achieve smooth and unnoticeable transition between cells. When the view point moves across the cell boundary and makes a transition from cell A to cell B, we will switch the geometry set from G_A to G_B, depth mesh set from D_A to D_B, and cached image set from I_A to I_B. Accordingly, it may spend a lot of time on loading data from disk.

Here, we develop a pre-fetch mechanism which preloads the geometry and image data of neighboring cells when CPU load is relative low. It will amortize the loading time to several inside-cell frames and reduce the difference of rendering time between an inside-cell frame and a cross-cell-boundary frame. We can easily attain this by setting the priority of a pre-fetch thread to be lower then others. As a result, we will not be interrupted by the loading of newly navigated data during the cell transition and obtain a more smooth frame rates and an unnoticeable transition. Note that the smaller cell size is the more frequent the transition is.

4. EXPERIMENTS

Our test scenes are two statuary parks, one is consists of 358 objects with 970,254 polygons (sparse scene), and another is consists of 956 objects with 2,265,978 polygons (dense scene, see Figure 1(a)) on an area of 2400×2000. Figure 6 represents images of the same view rendered by different rendering schemes.

The test platform is a PC with an AMD ThunderBird 1200Mhz CPU, 512MB main memory, and an nVidia GeForce3 with 64MB DDR RAM graphics accelerator.

4.1. Image Quality

To identify how much quality-loss of our proposed method, we use the signal-to-noise ratio SNR(dB), defined as follows:

$$\text{SNR} = 10 * \log \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x,y) - f(x,y)\right]^2}$$

where $\hat{f}(x,y)$ is the pixel color of the approximated image at position (x, y), M and N are the dimensions of the image. Before applying SNR, the RGB color is mapped to a single luminance value Y since our human eyes are more sensitive to the changes in luminance than to the changes in chrominance. Such mapping[6] is $Y = 0.299*R + 0.587*G + 0.114*B$. Note that, human eyes in general would not able to distinguish between two images that have an SNR greater than 25dB.

4.2. Cell Size and Self-Occluding Error Consideration

Several settings of the combination of different cell size (50 and 100) and self-occluding error tolerance (1-, 3-, and 5-pixel) are used in our experimental test.

The distributions of LOD meshes and depth meshes under two different self-occluding error tolerances for the dense scene are shown in Figure 1(b) and Figure 1(c), respectively. A LOD mesh is marked as a blue rectangle while a depth mesh as a red hollow diamond. As we can expect, the higher self-occluding error we can tolerate, the more objects are represented by depth meshes. In the case of cell size 50 and 1-pixel error tolerance, 54.0% objects are represented by depth meshes, while it is 63.9% for 3-pixel error tolerance. However, with no doubt, image quality for 3-pixel tolerance is lower than that of 1-pixel tolerance.

Since we want to identify how much benefit and image quality-loss comes from the depth mesh representation alone, we use original meshes instead of LOD meshes to represent those objects with the self-occluding error exceeding the tolerance. Such that, when cooperating with LOD techniques the performance would be even better.

Table 2 lists the polygon number of regional back-face culled LOD meshes, depth meshes, polygons inside a cell, the number of objects represented by depth mesh, the average frame rate, and average image quality under different settings. Under 60° FOV, three view windows should be rendered, such that, half of the LOD meshes, half of the depth meshes and all meshes inside the cell are the potential visible polygons. All of them show that, more polygons are simplified for the higher error tolerance setting with the cost of higher quality-loss. That is, our proposed scheme provides a tradeoff between the performance and image quality. Note that, under the same self-occluding tolerance, a larger cell size would not produce much poorer image quality, mainly due to the fact that the selection of object's representation is designed to ensure the bounded self-occluding error. Hence, the cell size has little impact on the image quality under our proposed scheme. However, as the cell size increases, more objects are represented by LOD meshes, more objects are put inside the navigation cell, and the number of potential visible polygons increases also, so the number of polygons increases dramatically. As a result, worse performance will be found.

The depth mesh simplification can simplify about 94% polygons in average, and, the farther object is, the greater simplification ratio is. On the other hand, the conservative back-facing culling could cull out about 43.7% polygons of LOD meshes for the case of cell size 50, and 41.3% for cell size 100. In general, the larger size a cell has or the nearer object is, the fewer back-facing rate is. At the time of writing this paper, we are still working on several minor bugs of this pre-processing occlusion culling program; therefore, we have no information about how many percentage of polygons could be culled.

4.3. Performance

We use the setting of cell size 50 and 3-pixel self-occluding error tolerance for the further testing for the dense scene. Three rendering configurations are used to do the performance comparison:

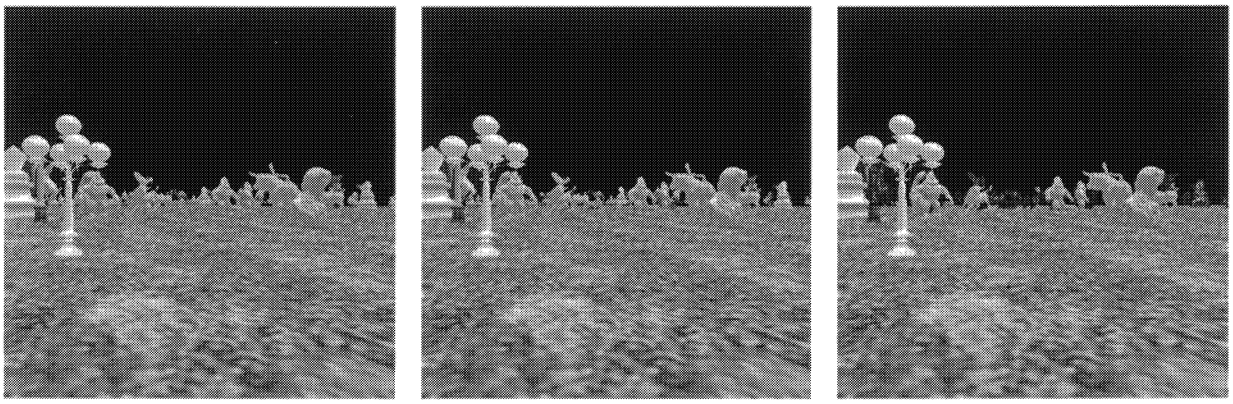
- **A: (Pure geometry)** The original meshes of the scene are rendered using the traditional graphics pipeline.
- **B: (Pure geometry w/ view frustum culling)** Same as **A**, but with software view frustum culling.
- **C: (Proposed scheme w/o run-time occlusion culling)** The scene is rendered by proposed scheme with view frustum culling but *without* run-time occlusion culling.

Table 3 lists the performance of each configuration. Configuration **C** spends additional time on loading neighboring cell data at the first frame, such that, there is a low peak at the first frame. Our proposed scheme w/o run-time occlusion culling has about 20.4 speedup factor compare to the pure geometric rendering while yields an average SNR about 22.2dB (13.6× under 1-pixel error tolerance and yields an SNR about 26.5dB.) It shows that our proposed method provides a faster rendering with an acceptable quality-loss while configuration **B** is hardly to achieve interactive frame rate for such complex scene. Figure 7 depicts the frame rate of every rendering frame on the same navigation path under these three rendering configurations.

As mentioned before, the overhead of run-time occlusion culling is un-negligible, and our test scene is not a densely occluded environment, our experiment is, w/ runtime occlusion culling, it is slower than **C** but still outperforms **B**.

5. DISCUSSION & FUTURE WORK

In this paper, we have proposed a hexagonal spatial subdivision and a hybrid rendering scheme for navigating complex scenes. Such scheme can achieve a smooth, navigation with no apparent popping effects at an almost constant and interactive frame rate for a very complex scene. By cooperating with LOD meshes and object-based depth meshes,



(a) Rendered by pure geometries.

(b) Rendered by the proposed scheme.

(c) Depth meshes are rendered with its wire-frame.

Figure 6. The same view rendered by different methods of our test scene.

Table 2. The average potential visible polygon no. for a cell, the average frame rate, and image quality under the different settings.

	Polygon no. of regional back-face culled LOD mesh	Polygon no. of depth mesh	Polygon no. of inside mesh	Polygon no. of potentially visible polygons	No. of objects represented by depth mesh	Average frame rate (fps)	Image quality (SNR(dB))
sparse scene (358 objects, 970,254 polygons)							
size 50, 1-pixel	198,830	27,579	2,278	115,483	183.5	30.0	26.4
size 50, 3-pixel	80,875	43,086	2,278	64,258	221.5	42.8	23.9
size 50, 5-pixel	50,921	48,843	2,278	52,160	231.5	48.9	23.2
size 100, 3-pixel	152,796	35,745	8,452	102,723	207.4	36.0	24.9
dense scene (956 objects, 2,265,978 polygons)							
size 50, 1-pixel	433,054	69,293	5,281	256,455	516.2	20.0	26.5
size 50, 3-pixel	179,805	102,501	5,281	146,434	610.7	29.9	22.2

Table 3. Performance under the three configurations.

	A	B	C
Frame time(ms)	680.3	268.1	33.4
Frame rate(fps)	1.47	3.73	29.9
Speed up	1.0×(baseline)	2.54×	20.4×

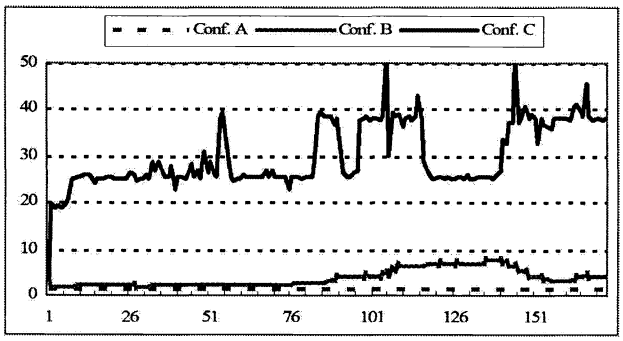


Figure 7. The frame rate (fps) under the three configurations on the same navigation path.

parallax error, crack, hole, rubber effect, and popping effect can be minimized. Further-more, with visibility preprocessing, polygons invisible from a region will never be sent into graphics pipeline. A tradeoff between performance and quality requirement can be easily made by specifying the self-occluding error tolerance.

When constructing the depth mesh in both object and image space, a special affair should handle with care. The rasterization engine of graphics hardware may render small polygon specially, such that a vertex may not projected into the same image pixel, in a result, a wrong weight may retrieved.

As the future works, we will firstly finish the preprocessing occlusion culling program to effectively cull out the polygons that are invisible from a cell region. We will also improve the depth mesh construction and simplification to have a better simplified depth mesh. For handling more complicated and extremely large scale scene, distant objects that are close to each other can clustered together to generate a single depth mesh. Such that, the amount of textures could be reduced and an approximated occlusion culling is done on the fly. Moreover, we will try to exploit the data coherence between neighboring cells to improve our pre-loading scheme.

REFERENCES

1. S. E. Chen and L. Williams. "View Interpolation for Image Synthesis," in *Computer Graphics (SIGGRAPH 93 Proceedings)*, pp. 279–288, 1993.
2. D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. "Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes," *Computer Graphics Forum*, 17(3):243–253, 1998.
3. L. Darsa, B. Costa, and A. Varshney. "Walkthroughs of Complex Environments using Image-based Simplification," in *Computers & Graphics*, volume 22, pp. 55–69, 1998.
4. X. Decoret, G. Schaufler, F. X. Sillion, and J. Dorsey. "Multi-Layered Impostors for Accelerated Rendering," *Computer Graphics Forum*, 18(3):61–73, 1999.
5. F. Durand, G. Drettakis, J. Thollot, and C. Puech. "Conservative Visibility Preprocessing using Extended Projections," in *Computer Graphics (SIGGRAPH 2000 Proceedings)*, pp. 239–248, 2000.
6. R. C. Gonzalez and R. E. Woods. *Digital Image* Processing, chapter 4, page 228, 1993.
7. S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. "The Lumigraph," In *Proceedings of SIGGRAPH 96*, pp. 43–54, 1996.
8. N. Greene, M. Kass, and G. Miller. "Hierarchical Z-Buffer Visibility," *In Computer Graphics (SIGGRAPH 93 Proceedings)*, pp. 231–238, 1993.
9. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. "Mesh Optimization," in *Computer Graphics (SIGGRAPH 93 Proceedings)*, pp. 19–26, 1993.
10. S. Kumar, D. Manocha, B. Garrett, and M. Lin. "Hierarchical Back-face Culling," In *7th Eurogra-phics Workshop on Rendering*, pp. 231–240, 1996.
11. M. Levoy and P. Hanrahan. "Light Field Rendering," in Proceedings *of SIGGRAPH 96*, pp. 31–42, 1996.
12. W. R. Mark, L. McMillan, and G. Bishop. "Post-Rendering 3D Warping," in *Symposium on Interactive 3D Graphics*, pp. 7–16, 1997.
13. L. McMillan and G. Bishop. "Plenoptic Modeling: An Image-Based Rendering System," in *Proceedings of SIGGRAPH 95*, pp. 39–46, 1995.
14. J. Shade, D. Lischinski, D. H. Salesin, T. DeRose, and J. Snyder. "Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments," in *Proceedings of SIGGRAPH 96*, pp. 75–82, 1996.
15. J. W. Shade, S. J. Gortler, L.-W. He, and R. Szeliski. "Layered Depth Images," in *Proceedings of SIGGRAPH 98*, pp. 231–242, 1998.
16. F. Sillion, G. Drettakis, and B. Bodelet. "Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery," in *Proceedings of Eurographics'97*, pp. 207–218, 1997.
17. H. Zhang and K. E. Hoff III. "Fast Backface Culling Using Normal Masks," in *Symposium on Interactive 3D Graphics*, pp. 103–106, 1997.
18. H. Zhang, D. Manocha, T. Hudson, and K. Hoff. "Visibility Culling Using Hierarchical Occlusion Maps," in *Computer Graphics*, volume 31, pp. 77–88, 1997.