The Global Occlusion Map: A New Occlusion Culling Approach*

Wei Hua Hujun Bao Qunsheng Peng A. R. Forrest

State Key Lab of CAD & CG,
Zhejiang University
Hangzhou, P.R. China
86-571-87951045
{huawei|bao|peng}@cad.zju.edu.cn

School of Information System,
University of East Anglia
Norwich， NR4 7TJ.， U.K
44-1603-592605
forrest@sys.uea.ac.uk

ABSTRACT

Occlusion culling is an important technique to speed up the rendering process for walkthroughs in a complex environment. In this paper, we present a new approach for occlusion culling with respect to a view cell. A compact representation, the Global Occlusion Map (GOM), is proposed for storing the global visibility information of general 3D models with respect to the view cell. The GOM provides a collection of Directional Visibility Barriers (DVB), which are virtual occluding planes aligned with the main axes of the world coordinates that act as occluders to reject invisible objects lying behind them in every direction from a view cell. Since the GOM is a two-dimensional array, its size is bounded, depending only on the number of the sampled viewing directions. Furthermore, it is easy to conservatively compress the GOM by treating it as a depth image. Due to the axial orientations of the DVBs, both the computational and storage costs for occlusion culling based on the GOM is minimized. Our implementation shows the Global Occlusion Map is effective and efficient in urban walkthrough applications.

Categories and Subject Descriptors

I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism.

General Terms

Algorithms, Performance, Design, Experimentation.

Keywords

visibility culling, occlusion culling, global visibility, rendering system, potentially visible set.

*This work received support from the National Natural Science Foundation of China for Innovative Research Groups (Grant No. 60021201) and for Distinguished Young Scholars(No. 69925204), National 863 Program (No. 2001AA135040), and The Royal Society China Joint Project No. Q807.

VRST'02, November 11-13, 2002, Hong Kong.

1. INTRODUCTION

Visibility computation is a fundamental topic in computer graphics. The intrinsic problem of determining the visible portions of the scene was recognized as a problem of sorting in three dimensions and was solved by a variety of hidden surface removal (HSR) algorithms. The Z-buffer algorithm, a hardware supported HSR algorithm, has been commonly employed for interactive rendering. In fact it is the only HSR algorithm with linear complexity $O(N)$ where N denotes the input number of faces in the scene. Nevertheless, as the number of faces in scenes is increasing rapidly in current applications, the traditional HSR algorithms fail to process them in real time. *Visibility culling* is a promising technique to reduce the cost of visibility determination. It aims at culling most of the invisible faces with minimal computation remaining during rendering. Visibility culling includes back-face culling, view frustum culling and *occlusion culling*[12]. Occlusion culling tries to set up the appropriate occluders to reject objects behind them without elaborate comparisons. For walkthroughs of urban area or complex architectures, occlusion culling is imperative since most objects are hidden behind the occluders with respect to the current viewpoint. Probably the earliest work in this area is by Jones in 1971 [23].

Depending on whether the occluders are applicable for just one viewpoint or for a view region, there are two kinds of occlusion culling algorithms. The so-called point-based occlusion culling approaches [1,2,13] derive a discrete representation for the occluders in the image space with respect to the current viewpoint during the rendering and the invisibility of objects can be easily determined by organizing this discrete representation into a hierarchical form. Due to the discrete form of occluder representation, point-based occlusion culling approaches are simple to implement and the occluders can be dynamically fused.

Other occlusion culling approaches treating a single viewpoint set up the occluders in object space [14,15,16]. Since all the computations for occlusion culling are accomplished by the CPU, their performances are no longer affected by the time consuming feedbacks from the graphics hardware. Nevertheless these approaches might be less effective for fusing occluders when no larger occluder is available.

Clearly the choice of occluders heavily influences the performance of occlusion culling approaches. Most approaches select occluders based on heuristic rules, but their effect as occluders for invisibility culling is not always satisfactory. Recently, Bernardini et al. [3] proposed an approach that approximates the scene model by an octree and sets the valid boundary faces of some octree's nodes as occluders. This

approach takes advantage of the fact that all occluders are aligned with the world coordinate axes. However, collecting the valid occluders by traversing the octree during walkthrough causes extra costs.

Another kind of occlusion culling approach aims at providing the occlusion information for a view cell along all directions. Some approaches offer answers to *global visibility* queries, but suffer from extremely large complexity both of computation and of storage. The 3D visibility complex [17] is another way of describing and studying the global visibility of 3D space, but its size is still $O(n^4)$. Recently, Durand proposed a new structure, called the visibility skeleton, to compactly encompass accurate global visibility information [18]. However, either its size or its construction cost cannot yet meet the demand for walkthroughs of a complex scene.

Rather than computing global visibility, a variety of approaches attempt to estimate the *potential visible object set* (PVS) for each view cell. Teller et al. [4,19] presented a cell-to-cell visibility determination scheme for architectural models. Wonka et al. designed a distributed system to calculate the PVS online [20]. By shrinking the occluders, the visible faces with respect to a number of points on the boundary of a view cell conservatively form the PVS of the view cell. Subdividing the 5D ray space suggests another way to compute PVS [21,22].

Recent research makes use of fused occluders to compute the *PVS*. Schaufler et al. [7] used the cuboid opaque interior of objects as occluders. Such choice of occluders facilitates the combined occlusion of adjacent opaque regions of space. Koltun et al. [9] introduced the concept of a virtual occluder, which is calculated by merging the umbra of the front polygonal occluder with that of the rear one repeatedly. In [8] the cross sections of occluders in a stack of slices are discretized and aggregated to form the discrete representation, which is used to determine the potential visibility of the occludees.

The most severe problem for PVS-based approaches is their huge storage cost. This is particularly true when the global visibility of the scene varies dramatically from one view cell to another view cell when the scene is complex.

In this paper, we present a new approach for occlusion culling with respect to a view cell. A compact representation, the Global Occlusion Map (*GOM*), is proposed for storing the global visibility information of general 3D models with respect to the view cell. The *GOM* provides a collection of Directional Visibility Barriers (*DVB*) which are virtual occluding planes aligned with the axes of the world coordinates and which act as occluders to reject invisible objects lying behind in every direction from a view cell. Since the *GOM* is a two-dimensional array, its size is bounded, depending only on the number of the sampled viewing directions. Furthermore, it is easy to conservatively compress the *GOM* by treating it as a depth image. Due to the axial orientations of the *DVB*s, both the computation and storage costs for occlusion culling based on the*GOM* are minimized.

In the following section, we will give a straightforward definition for the *GOM*. An equivalent but convenient form of cube-based *GOM* is introduced in section 3. We propose a conservative method to estimate the *DVB* for a view direction in section 4.1. An adaptive algorithm for *GOM* construction is presented in section 4.2. Section 5 outlines our new approach based on the *GOM* while section 6 shows some experimental results. The challenging issues for future research work are addressed in the final section.

2. THE GLOBAL OCCLUSIOIN MAP

The region a viewer can traverse in the environment can be divided into a number of sub-regions, called view cells. We use $ray(\boldsymbol{v},\boldsymbol{q})$ to denote a ray cast from the point $\boldsymbol{q}$ within a view cell Q in the direction $\boldsymbol{v}$. Let $\boldsymbol{p}^{vis}(\boldsymbol{v},\boldsymbol{q})$ be the first intersection point between $ray(\boldsymbol{v},\boldsymbol{q})$ and the surfaces in the scene. Based on the above notations, a Global Occlusion Map of a view cell Q can be defined as

$$GOM(\boldsymbol{v}) = \mathrm{Max}\{\ (\boldsymbol{p}^{vis}(\boldsymbol{v},\boldsymbol{q}) - \boldsymbol{c}) \bullet \boldsymbol{v} \mid \boldsymbol{q} \in Q\ \}, \quad \boldsymbol{v} \in S \qquad (1)$$

where $\boldsymbol{c}$ is the center of the view cell and $\boldsymbol{v}$ can be any direction vector within the direction space of a unit sphere S. Intuitively, $\boldsymbol{p}^{vis}(\boldsymbol{v},\boldsymbol{q})$ is composed of the visible point set from all points $\boldsymbol{q}$ within a view cell along a specific direction $\boldsymbol{v}$, $GOM(\boldsymbol{v})$ records the maximum distance between $\boldsymbol{p}^{vis}(\boldsymbol{v},\boldsymbol{q})$ and the center of the view cell in the direction $\boldsymbol{v}$, as shown by Fig.1. In the case where some rays do not hit any surfaces, $GOM(\boldsymbol{v})$ is set to be infinite (∞). Clearly the $GOM(\boldsymbol{v})$ defines a boundary plane in object space that guarantees all objects lying behind it are invisible from any point in the view cell in the direction $\boldsymbol{v}$ (Fig. 1). We call this virtual plane a *Directional Visibility Barrier* (*DVB*). The *GOM* is in fact a compact representation of the collection of $DVB(\boldsymbol{v})$ along all directions $\boldsymbol{v}$.

It is easy to use a precomputed *GOM* to conservatively determine whether a point $\boldsymbol{p}$ in object space is invisible for any viewpoint $\boldsymbol{q}$ within the view cell Q during rendering stage. If

$$(\boldsymbol{p}-\boldsymbol{c}) \bullet \boldsymbol{v} > GOM(\boldsymbol{v}), \qquad (2)$$

where $\boldsymbol{v}=(\boldsymbol{p}-\boldsymbol{q})/\|\boldsymbol{p}-\boldsymbol{q}\|$, then $\boldsymbol{p}$ must be invisible (Fig.2).

Note that every point (φ,θ) on the surface of a unit sphere specifies a direction. We sample the unit sphere uniformly and represent the continuous direction space by a set of discrete directions. $GOM(\boldsymbol{v})$ can then be represented by a discrete 2D array. The discretisation may incur some aliasing. The general solution to alleviate such aliasing is to super-sample the unit sphere.

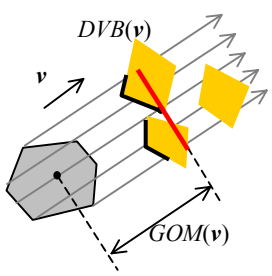


Figure 1: $GOM(\boldsymbol{v})$ is the projected distance between $\boldsymbol{c}$ and the furthest visible point from the view cell in direction $\boldsymbol{v}$.

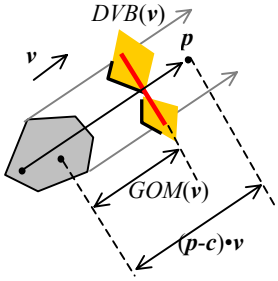


Figure 2: The condition $(\boldsymbol{p}-\boldsymbol{c})\bullet\boldsymbol{v} > GOM(\boldsymbol{v})$ is used to determine the invisibility of $\boldsymbol{p}$ with respect to the viewpoint $\boldsymbol{q}$, where $\boldsymbol{v}=\boldsymbol{p}-\boldsymbol{q}/\|\boldsymbol{p}-\boldsymbol{q}\|$.

Clearly *GOM* has an advantage over PVS by representing the global visibility information of an environment. PVS as the collection of indices of all potential visible faces or objects from each view cell whilst *GOM* is the collection of *DVB* for all directions. For each direction, *GOM* only records a distance value to represent the corresponding *DVB*. Therefore the size of the *GOM* is independent of both the complexity of the scene and the location of the view cell. It is bounded once the resolution of the direction discretization is determined. By contrast, the size of PVS relies heavily on the complexity of the

scene as well as the location of the view cell, and hence is unbounded. Furthermore, the PVS information is difficult to compress because it stores indices. Conservative PVS undoubtedly means more storage. This is to be compared with the efficient compression of *GOM* information. In fact, the *GOM* can be regarded as a depth image and depth coherence between adjacent elements in the *GOM* exists, enabling various image compression methods to be used for compression of *GOMs*. Note that, when a maximum depth value of adjacent elements is used to replace the diverse value of each of these elements, the *GOM* is still conservative in occlusion culling. Finally, to calculate the visibility of moving objects in the scene, current PVS-related approaches estimate the potential visibility of the envelope of these objects during motion rather than calculating directly the visibility of each dynamic object at each specific moment, or they just simply add these dynamic objects into the PVS. Overestimation of visibility is therefore inevitable when the trajectories of these objects are very long or unpredictable during the stage of PVS precomputation. On the other hand, the *DVB* may remain the same regardless of these dynamic objects. Although they may potentially decrease the distance value of each *DVB* from time to time, the original *GOM* can still be used conservatively to cull any objects moving in random routes on-the-fly according to their updated position.

3. CUBE-BASED GOM

Alternatively, we can use a cube to represent the direction space. We call each face of the cube a *principal viewport* denoted by $PVPort(k)$ (k=0…5). A point on $PVPort(k)$ is specified by (s,t,k), where (s,t) are the local coordinates of the point on $PVPort(k)$. The origin of the local coordinates is set at the center of $PVPort(k)$, with the s-axis and t-axis parallel to the two edges of $PVPort(k)$. We use $\boldsymbol{v}(s,t,k)$ to denote a direction vector. $\boldsymbol{v}(0,0,k)$ is called the *principal direction* of $PVPort(k)$, abbreviated to $\boldsymbol{v}(k)$.

Using the convention of a cube system, the definition of *GOM* can be expressed as the following.

$$GOM(s,t,k)=\mathrm{Max}\{(\boldsymbol{p}^{vis}(\boldsymbol{v}(s,t,k),\boldsymbol{q})-\boldsymbol{c})\bullet\boldsymbol{v}(k)|\boldsymbol{q}\in Q\} \quad (3)$$

Note in this definition, $(\boldsymbol{p}^{vis}(\boldsymbol{v}(s,t,k),\boldsymbol{q})-\boldsymbol{c})$ is projected onto the principal direction $\boldsymbol{v}(k)$ instead of $\boldsymbol{v}(s,t,k)$. This kind of *GOM*, called *cube-based GOM*, also generates an appropriate *DVB* for each direction, as shown in Fig. 3. Note the new *DVB* is perpendicular to the principal direction $\boldsymbol{v}(k)$. Using a cube-based *GOM*, the condition for determining the invisibility of point $\boldsymbol{p}$ with respect to viewpoint $\boldsymbol{q}$ is

$$(\boldsymbol{p}-\boldsymbol{c})\bullet\boldsymbol{v}(k) > GOM(s,t,k), \quad (4)$$

where (s,t,k) is specified by $\boldsymbol{v}(s,t,k)=(\boldsymbol{p}-\boldsymbol{q})/\|\boldsymbol{p}-\boldsymbol{q}\|$.

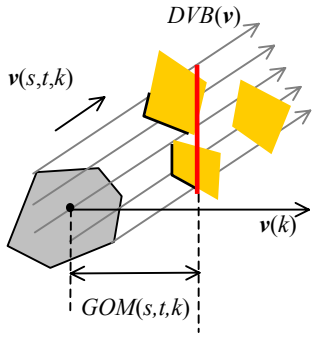


Figure 3: *GOM*(s,t,k) is the distance between the *DVB* and *c*. *DVB* is perpendicular to the principal direction $\boldsymbol{v}(k)$ instead of $\boldsymbol{v}(s,t,k)$.

Note that the invisibility condition (4) can be computed less expensively. If we align the cube with the world coordinates of the scene, then $\boldsymbol{v}(k)$ (k=0…5) will be consistent with the axis of the world coordinates and calculation of $(\boldsymbol{p}-\boldsymbol{c})\bullet\boldsymbol{v}(k)$ will be reduced to evaluating the difference of a corresponding coordinate component of $\boldsymbol{p}$ and $\boldsymbol{c}$.

By discretizing each principal viewport, we can get the discrete form of the cube-based *GOM*. In the remaining sections, we will use the notation $GOM(i,j,k)$ to represent a discrete cube-based *GOM* and $GOM(s,t,k)$ to denote a continuous cube-based *GOM*.

4. GOM CONSTRUCTION

To construct *GOM*, we can sample the view cell densely and use the Z-buffer algorithm to calculate the visible points at each sample viewpoint with the viewing directions aligned with the principal directions. These visible points can then be used to update *GOM*. Unfortunately, this naïve approach is impractical since it incurs massive computation with complexity of $O(N^5)$ where $N\times N$ is the resolution of the Z-buffer.

In this section, we present an adaptive algorithm for *GOM* construction based on conservative estimation of *DVB*. In the following, we assume that the scene is composed of a set of triangles. Such assumption is general enough to handle almost all kinds of 3D scene.

4.1 Conservative estimation of *DVB*

Let us see how to conservatively estimate *DVB* based on the information of a few directional visible points. Fig. 4 shows a case where three directional visible points have been detected and identified by three gray dots. We can easily find the visible triangles at which these directional visible points are located. Then we perform a so-called *directional occlusion test*, to judge whether these triangles are sufficient to occlude all the rays cast from any point within the view cell along the direction $\boldsymbol{v}$. If the result is true, these visible triangles constitute $DVB(\boldsymbol{v})$ and we can use them to estimate $GOM(\boldsymbol{v})$.

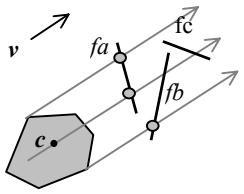


Figure 4: Triangles *fa* and *fb* occlude all the rays cast from view cell along the direction *v*. *fa* and *fb* are obtained from the corresponding directional visible points.

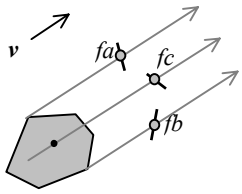


Figure 5: Triangles *fa*, *fb* and *fc* cannot occlude all the rays cast from view cell along the direction *v*.

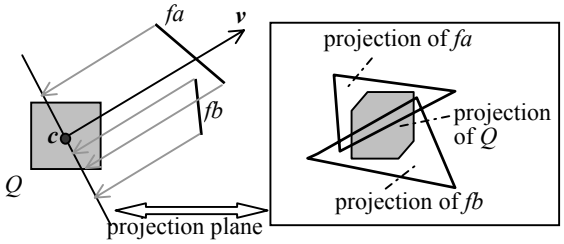


Figure 6: The orthogonal projection of the visible triangles *fa*, *fb* and view cell *Q* along *v*. The right shows the projection results. The union of projections of *fa* and *fb* can cover that of *Q*.

To perform the directional occlusion test, we project all these visible triangles and the surface of the view cell along $\boldsymbol{v}$ orthogonally onto a plane that passes through the center of the view cell and is perpendicular to $\boldsymbol{v}$. In the projection plane, we test whether the union of the projections of the triangles can completely cover the projected region of the view cell (Fig.6). If it can, the test succeeds, otherwise, the test fails and more directional visible points should be sampled (Fig.5). We adopt a progressive approach to get more directional visible points. The directional occlusion test is then repeatedly continued by incorporating more triangles which the newly found directional visible points belong into until either they occlude all rays or the number of directional visible points exceeds a given threshold.

Several methods can be used to evaluate $GOM(\boldsymbol{v})$ once the respective directional occlusion test succeeds. For example we can use a Z-buffer algorithm supported by the graphic accelerator to find all the visible points along the direction $\boldsymbol{v}$ and get the largest depth, which is $GOM(\boldsymbol{v})$. However, this method has to read the Z-buffer pixel by pixel from the graphic accelerator in order to find the largest depth, which inevitably decreases the efficiency of our approach.

Instead, we make use of the vertices of the visible triangles to conservatively estimate $GOM(\boldsymbol{v})$. It is evident that

$$GOM(\boldsymbol{v}) \le GOM^{est}(\boldsymbol{v}) \qquad (5)$$

where $GOM^{est}(\boldsymbol{v}) = \max\{ (\boldsymbol{p}-\boldsymbol{c})\bullet\boldsymbol{v} \mid \boldsymbol{p}$ is any vertex of the visible triangles$\}$, and the estimated $DVB^{est}(\boldsymbol{v})$ can also ensure that all points behind it are definitely invisible from the view cell along $\boldsymbol{v}$. Fig. 7a gives an example to illustrate $DVB^{est}(\boldsymbol{v})$ and $GOM^{est}(\boldsymbol{v})$. Therefore for the cube-based GOM, we can estimate GOM by using (Fig. 7b)

$GOM^{est}(s,t,k) = \max\{ (\boldsymbol{p}-\boldsymbol{c})\bullet\boldsymbol{v}(k) \mid \boldsymbol{p}$ is any vertex of the visible triangles$\}$.

In the remainder of this paper, we do not distinguish $GOM(s,t,k)$ and $GOM^{est}(s,t,k)$ since both of them can describe an effective DVB.

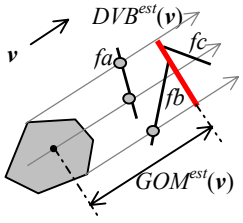


Figure 7a: The $DVB^{est}(\boldsymbol{v})$ defined by $GOM^{est}(\boldsymbol{v})$ can also occlude invisible triangles from the view cell.

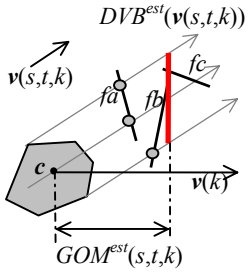


Figure 7b: The estimation of the cube-based *GOM*

4.2 Adaptive algorithm for construction of GOM

In this section, we will present an adaptive algorithm for constructing a cube-based GOM. We first subdivide the entire region a viewer can traverse into a number of uniform cubic view cells according to the complexity of the scene. Each face of a view cell is divided into pixels with the resolution of $N\times N$. The vertices of all cubic cells constitute a 3D array Q. At each vertex $\boldsymbol{q} \in Q$, we set up a unit cube centered at $\boldsymbol{q}$ with the same orientation as the view cell. Each face of the unit cube is also divided into $N\times N$ pixels, so $pixel(i,j)$ on the same oriented faces of all cubes specifies the same direction $\boldsymbol{v}(i,j,k)$, where k refers to the corresponding principal viewport.

We then adopt the hardware-supported Z-buffer algorithm to find the visible points from $\boldsymbol{q}$ along each discrete direction and record the indices of the corresponding visible triangles in an item buffer.

Next, we perform the directional occlusion test view cell by view cell along each direction $\boldsymbol{v}(i,j,k)$ with the information of the visible triangle set stored in the item buffers at its eight vertices. The results are then written into the *Occlusion-buffer* of the current view cell. The results can be one of three forms: "occluded", "not-occluded" and "uncertain". The "occluded" value of $Occlusion\text{-}buffer(i,j,k)$ indicates that the respective $DVB(\boldsymbol{v}(i,j,k))$ is ready to be derived along that direction. On the other hand, the value "not-occluded" means that no visible triangle is found in at least one item buffer along the direction $\boldsymbol{v}(i,j,k)$. In both of the above cases, the result is deterministic. If the value of $Occlusion\text{-}buffer(i,j,k)$ is "uncertain", the directional occlusion test fails.

After the directional occlusion test is accomplished, we count the number of pixels whose value is "uncertain" in the *Occlusion-buffer* of the current view cell. We adopt two slightly different approaches for further process. If the number is larger than a given threshold, we subdivide the current view cell into eight smaller view cells and establish the item buffers at the vertices of all child view cells. The *Occlusion-buffer* of each child view cell initially takes the same values as that of its patent. It is then updated by performing the directional occlusion test in the relevant directions.

Listing 1: The pseudo codes for ConstructGOM

```
ConstructGOM( GOM, Viewcell )
{
ı of GOM and Occlusion-buffer
    Refine(GOM, Viewcell, NULL, Occlusion-buffer )
    For each (i,j) pair while 1≤i≤N, 1≤j≤N
      If( Occlusion-buffer(i,j) is not "occluded" )
      {
          Update GOM(i,j) and Occlusion-buffer(i,j) by
            projecting all triangles orthogonally along v(i,j)
      }
}
```

If the number is smaller, we orthogonally project all scene triangles along the direction $\boldsymbol{v}(i,j,k)$ onto the plane passing through the center of the current view cell and perpendicular to the *principal direction* $\boldsymbol{v}(k)$. All the visible triangles along this direction can be immediately found in this case and no further directional occlusion test is needed. The specific threshold value is chosen based on the balance of the computations involved in both approaches. Note that, the first approach normally detects more directional visible triangles along the direction $\boldsymbol{v}(i,j,k)$ during the subdivision process and these triangles should be merged together to perform the directional occlusion test for the parent view cell. This process is repeatedly carried on until the values of all pixels in the *Occlusion-buffer* of the current view cell have been determined.

Listings 1 and 2 give a pseudo code description of our algorithm for constructing a cube based GOM. For convenience, we describe the algorithm for a *principal viewport* only. Therefore, we use $GOM(i,j)$ to stand for $GOM(i,j,k)$ and $\boldsymbol{v}(i,j)$ for $\boldsymbol{v}(i,j,k)$, omitting the index k that specifies the *principal viewport*.

The variable *Distance*, equal to $\max\{ (\boldsymbol{p}-\boldsymbol{c})\bullet\boldsymbol{v}(k) \mid \boldsymbol{p}$ is any vertex of the visible triangles$\}$ where $\boldsymbol{v}(k)$ is the *principal direction* of

the *principal viewport*, is computed to update *GOM*(*i*,*j*) by the function DVBComp in Listing 2.

Listing 2 : The pseudo codes for Refine

```
Refine(GOM, Viewcell, Parent-O-Buffer , Occlusion-buffer )
{
    Sampling the scene with N×N resolution
        with viewpoint at each corner of ViewCell.
    Store results to Buffers.
    For each (i,j) pair while 1≤i≤N, 1≤j≤N
    {
        If( Parent-O-Buffer(i,j) is "uncertain" )
        {
            Get VisibleFaceSet from Buffers
            Flag = DVBComp( VisibleFaceSet, Distance )
            Occlusion-buffer(i, j) = Flag
            If( Flag is "occluded" )
                UpdateGOM( GOM(i, j), Distance)
        }else{
            Occlusion-buffer (i,j) = Parent-O-buffer (i,j)
        }
    }
    If( NeedSubdiv (Viewcell, Occlusion-buffer )
    {
        Subdivide Viewcell to a set of SubViewcell
        For each SubViewcell,
            Refine(GOM, SubViewcell, Occlusion-buffer,
             Child-O-buffer )
         Update Occlusion-buffer by all Child-O-buffer
    }
}
```

5. OCCLUSION CULLING WITH GOM

The *GOM* is a collection of visibility barriers for any viewpoint along any viewing direction. It thus provides explicit information for quickly rejecting objects hidden behind these barriers. For any viewpoint ***q***, we first find the respective view cell that encloses it. A unit cube is then setup at ***q*** with the same orientation and face resolution as that of the view cell. Any object to be tested is projected onto the faces of the unit cube. For each *pixel*(*i*,*j*,*k*) on the cube which is covered by the projection of the object, its visibility barrier is found from *GOM* (*i*,*j*,*k*). Let *d*(*k*) denote the minimum distance of the object to the bisection plane of the view cell perpendicular to the *principal direction* ***v***(*k*), shown in Fig. 8. If

$$d(k) > GOM(i,j,k),$$

then the object is invisible from *q* along that direction.

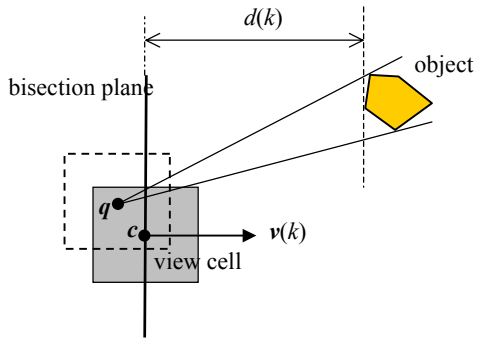


Figure 8: An object is projected onto the *principal viewport PVPort*(*k*) with respect to *q*. The minimum distance of the object to the bisection plane of the view cell perpendicular to the principal direction *v*(*k*) is denoted by *d*(*k*).

Owing to its classic structure, *GOM* is adaptable to many accelerating techniques. For example, *GOM* can be organized into a hierarchical form as a hierarchical Z-buffer. An appropriate level of the *hierarchical GOM* is found so that the projection of the object falls within one pixel, hence avoiding performing the invisibility test at multiple pixels. Also, objects in the scene can be organized into an enclosing box tree, enabling invisible objects to be culled in groups.

One notable feature of *GOM* is that it remains constant for all viewpoints and viewing directions within a view cell. This is in contrast with most of the current Z-buffer based algorithms for which cases the Z-buffer has to be rebuilt and dynamically updated according to the current viewing parameters during the walkthrough of the scene.

6. IMPLEMENTATION AND EXPERIMENTAL RESULTS

We built a virtual city scene as a testing dataset consisting of 29575 objects with a total of 1,556,574 triangles. A birds-eye view is shown in Fig. 12a. A vista is shown in Fig. 12b. At the stage of *GOM* construction, we divided the space along the main streets in the virtual city into view cells, shaded in red as shown in Fig. 12c.

We divided each face of a view cell at 128×128 resolution and set the threshold of the number of pixels in *Occlusion-Buffer* with value of "uncertain" to be 300 to launch the subdivision process of the current view cell. The recursion is terminated when a view cell is subdivided more than three times.

Fig. 9a shows the time cost for constructing the *GOM* of each view cell. The fluctuation indicates that the scene complexity varies over view cells. Average cost for constructing a *GOM* is 237.4 seconds. Fig. 9b shows the average time cost for handling each face of the *GOM*, i.e., a principal viewport during *GOM* construction. The principal viewports *PVPort*(2) and *PVPort*(3) face the sky and the ground respectively. Since the scene along these directions is of rather low complexity, it costs little computation time. The principal directions of *PVPort*(0) and *PVPort*(1) are aligned with the direction of the streets. More visible triangles have to be sampled in this case and the view cell needs to be subdivided into smaller cells. *PVPort*(4) and *PVPort*(5) orient towards the two ends of the streets. Since fewer objects are visible than through *PVPort*(0) or *PVPort*(1), it consequently costs less time.

Fig. 10 shows the depth image on each face of a *GOM*. To compress the information stored in the *GOM*, we adopt a simple approach by storing the images on different faces of a view cell at different resolutions according to the coherence between their adjacent pixels. We build a z-pyramid for each face of the *GOM* and count the number of pixels at a higher level whose four children take different depth values. When the number exceeds a given fraction γ, the resolution at the lower level is adopted for storing the depth image on that face. The parameter γ controls the tradeoff between storage size and conservativeness. From experiment, the depth values in *GOM* were quantitized into 256 steps by using an index table. We took $\gamma = 0.7$, and the average size for a *GOM* is about 17k. The average resolution for *PVPort*(0), *PVPort*(1), *PVPort*(4) and *PVPort*(5) is around 64×64 and the average resolution for *PVPort*(2) and *PVPort*(3) is close to 1×1.

In order to cull objects efficiently, we reorganized the scene into a hierarchical enclosing box tree and took the enclosing box of objects as the culling primitives in both view frustum culling and occlusion culling. Fig. 11a shows the effectiveness of occlusion culling based on *GOM* against that of the frustum

culling during a 700 frames walkthrough of a street in the virtual city. While the number of objects accepted after view frustum culling varies widely, the number of objects accepted after GOM based occlusion culling almost remains constant. The corresponding frame rates are shown in Fig. 11b. The PVS curve in Fig. 11b refers to the frame rate for displaying a pre-computed potential visible set generated offline by *GOM*. It demonstrates on the other hand the time efficiency of our online occlusion culling, only dropping about 1 frame per second.

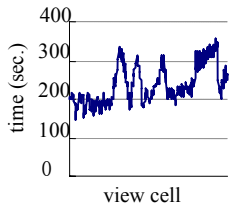


Figure 9a: The time cost for *GOM* construction

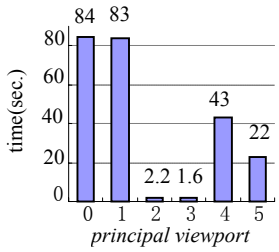


Figure 9b: Average time for handling each *principal viewport*

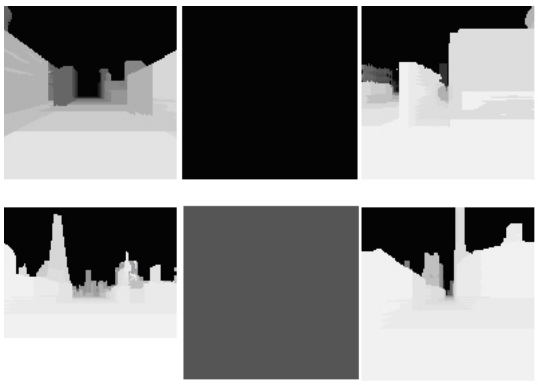
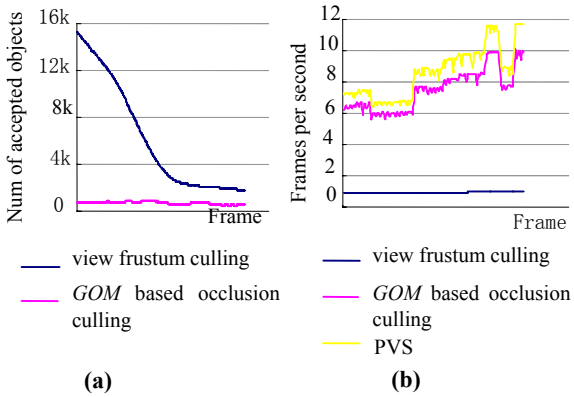


Figure 10: The depth images on each *principal viewport* of a constructed *GOM*.



(a) **(b)**

Figure 11 (a): Effectiveness comparison between view frustum culling and *GOM* based occlusion culling. (b): The rendering rate using respectively view frustum culling, *GOM* based occlusion culling and PVS.

7. CONCLUSION AND FUTURE WORK

We have proposed an innovative representation, the Global Occlusion Map, for invisibility culling. It compactly records the Directional Visibility Barriers in all directions for a view cell. Each Directional Visibility Barrier is a boundary plane in object space behind which all objects are directionally invisible with respect to the view cell. We have proposed an adaptive approach to construct the *GOM*. The key idea is to perform the *directional occlusion test* for each direction, which exploits the combined occlusion to find the Directional Visibility Barrier. By using the coherence between the adjacent pixels in the *GOM*, it is easy to compress the *GOM* into a bounded image of small size despite the complexity of the scene. Since our assumption for the scene's geometry is general, our approach can be used in many applications.

Compared with PVS based visibility approaches, our new approach has the following advantages:

1. It costs much less in storage whilst maintaining a good rendering rate comparable with PVS approaches when walking through a complex environment
2. *GOM* remains conservative when some dynamic objects intrude into the environment.
3. The information of the *GOM* can be conservatively compressed.

Many issues need to be investigated in future work. The most challenging one is to accelerate the construction process of *GOM* especially for scenes including very tiny triangles relative to the size of view cells. Exploiting the coherence between adjacent directions when estimating the Directional Visibility Barrier is another promising research topic to further increase the efficiency of *GOM* construction. More elaborate approaches for compressing *GOM* should also be studied.

8. REFERENCE

[1] Ned Greene, M. Kass, and Gavin Miller. Hierarchical Z-buffer visibility. Proceedings of SIGGRAPH'93, pages 231-240,1993.

[2] Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff III. Visibility culling using hierarchical occlusion maps. Proceedings of SIGGRAPH 97, pages 77-88, August 1997.

[3] F. Bernardini, J.T. Klosowski, and J. El-Sana. Directional discretized occluders for accelerated occlusion culling. Computer Graphics Forum, 19(3), 2000.

[4] Seth J. Teller and Carlo H. Sequin. Visibility preprocessing for interactive walkthroughs. Proceedings of SIGGRAPH'91, pages 61-69, 1991.

[5] John M. Airey, John H. Rohlf, and Frederick P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. Computer Graphics (1990 Symposium on Interactive 3D Graphics), 24(2), pages 41-50, March 1990.

[6] Graig Gotsman, Oded Sudarsky, and Jeffry Fayman. Optimized occlusion culling using five-dimensional subdivision. Computer & Graphics, 23(5), pages 645-654, 1999.

[7] Gernot Schaufler, Julie Dorsey, Xavier Decoret, and François X. Sillion. Conservative volumetric visibility with occluder fusion. Proceedings of SGGRAPH 2000, pages 229-238, July 2000.

[8] F. Durand, George Drettakis, Joelle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. Proceedings of SIGGRAPH 2000, pages 239-248, July 2000.

[9] Vladlen Koltun, Yiorgos Chrysanthou, and Daniel Cohen-Or. Virtual occluders: An efficient intermediate pvs representation. Rendering Techniques 2000: 11th Eurographics Workshop on Rendering, pages 59-70, June 2000.

[10] Michiel van de Panne and James Stewart. Efficient compression techniques for precomputed visibility. Proceedings of Eurographics Workshop on Rendering 1999, 1999.

[11] Hansong Zhang. Effective Occlusion Culling for the Interactive Display of Arbitrary Models. Ph.D. thesis, Department of Computer Science, UNC-Chapel Hill, 1998.

[12] Daniel Cohen-Or, Yiorgos Chrysanthou et. al. A Survey of Visibility for Walkthrough Applications. SIGGRAPH '00 Course Notes, Course 4, 2000.

[13] Dirk Bartz, Michael Messner, et. al. OpenGL-assisted occlusion culling for large polygonal models. Computers & Graphics, 23(5), pages 667-679, 1999.

[14] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. 1997 Symposium on Interactive 3D Graphics, pages 83-90, April 1997.

[15] T. Hudson, D. Manocha, et. al. Accelerated occlusion culling using shadow frusta. Proc. 13th Annu. ACM Sympos. Gomput. Geom., pages 1-10, 1997.

[16] J. Bittner, V. Havran, et. al. Hierarchical visibility culling with occlusion culling. Proceedings of Computer Graphics International '98, pages 207-219.

[17] F. Durand. 3D Visibility: Analytical study and Applications. PhD thesis, Université Joseph Fourier, Grenoble, France, July 1999.

[18] F. Durand, George Drettakis, et. al. The Visibility Skeleton: A Powerful and Efficient Multi-Purpose Global Visibility Tool. SIGGRAPH'96.

[19] Seth Teller. Visibility Computations in Densely Occluded Environments. PhD thesis, University of California, Berkeley, 1992.

[20] Peter Wonka, Michael Wimmer, et. al. Instant Visibility. EUROGRAPHICS'2001, pages C411-21, September 2001.

[21] Graig Gotsman, Oded Sudarsky, and Jeffry Fayman. Optimized occlusion culling using five-dimensional subdivision. Computers & Graphics, 23(5), pages 645-654, 1999.

[22] Yigang Wang, Hujun Bao and Qunsheng Peng. Accelerated Walkthroughs of Virtual Environments Based on Visibility Preprocessing and Simplification. EUROGRAPHICS'98, 1998, 187-194.

[23] C.B. Jones. A new approach to the 'hidden line' problem. The Computer Journal, 14(3), pages 232-7, August 1971.

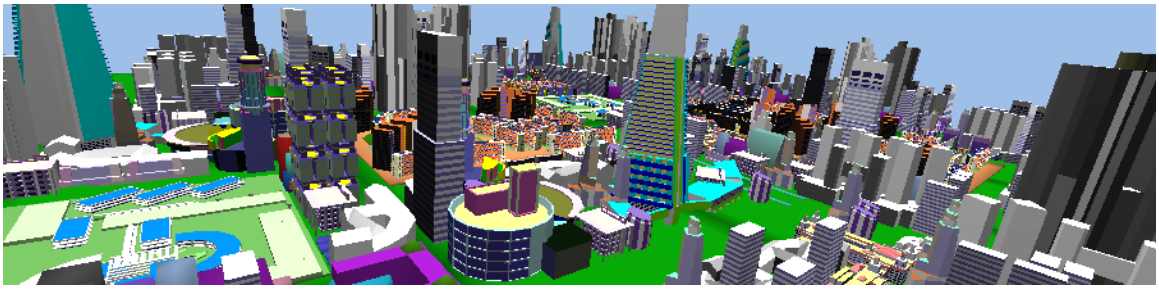


Figure 12a: A bird-eye's view of the virtual city

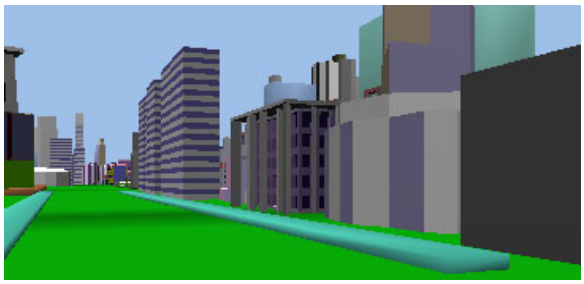


Figure 12b: A vista in the virtual city

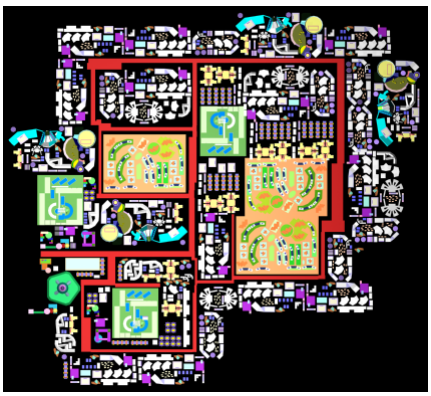


Figure12c: View cells in the virtual city