An On-line Occlusio-Culling Algorithm for Fast Walkthrough in Urban Areas†

Yusu Wang‡ Pankaj K. Agarwal‡ and Sariel Har-Peled§

Abstract

We describe a fast algorithm to speed up rendering of scenes for walkthroughs in urban environments. Our occlusion culling algorithm takes advantage of temporal coherence in image space. As such, occlusion calculation is performed online only when needed. This enables us to employ intelligent occluder-selection and culling algorithms. We do not preprocess visibility information or pre-select occluders. Therefore, we can update scenes dynamically at a little cost. The algorithm features a tradeoff between accuracy and efficiency. While it approximates visibility testing, our experiments show that errors occur rarely.

1. Introduction

In urban walkthroughs, a user virtually navigates through a 3D city model as a pedestrian or as an auto driver. Optimally, we would like interactive rendering of 30 to 60 frames a second. Unfortunately, data gathering techniques have outstripped advances in rendering hardware, making interactive rendering of massive data sets impossible without reducing the number of primitives rendered at each frame. Occlusion culling is one popular technique for this reduction.

Occlusion culling is especially suitable for urban environments since the scenes are usually densely occluded. However, the characteristics of urban environments also raise several challenging issues for occlusion culling algorithms. First, large amounts of objects in urban environments are hidden by the combination of several, not necessarily connected occluders, therefore, the effect of multiple occluders — occluders fusion — has to be considered for a city model. Second, most buildings in city models are of similar sizes, so a few occluders seldom suffice and occluder-selection methods only relying on heuristics such as size and distance may fail to capture significant occlusion in a city model (see Fig-

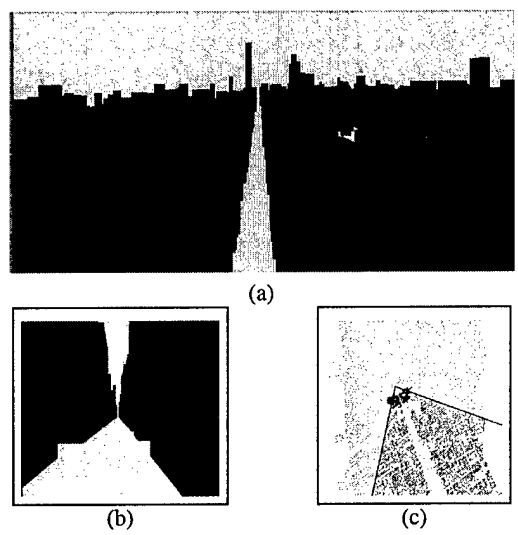


Figure 1: *Visualization of our algorithm on a Manhattan city model with 27,400 polygons. (a) is a bird-eye view. (b) is a view along a street. (c) shows light grey city map overlayed with black view frustum, dark grey culled objects, and black occluders. In this view, our algorithm culls 88% of the polygons.*

† Work is supported by Army Research Office MURI grant DAAH04-96-1-0013 and an NSF grant CCR–9732787. P.A. is supported by Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by NSF grants ITR–333–1050, EIA–9870724, EIA–997287, and CCR–9732787, and by a grant from the U.S.-Israeli Binational Science Foundation.

‡ Department of Computer Science, Duke University; Durham, NC 27708; USA. E-mail: wys,pankaj@cs.duke.edu

§ Department of Computer Science, University of Illinois; Urbana, IL 61801; USA. E-mail: sariel@cs.uiuc.edu. This work was done while the author was in Duke University.

© The Eurographics Association 2001.

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

20030612 053

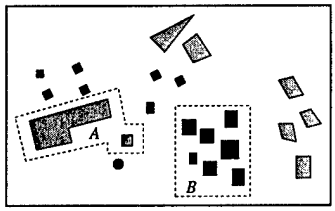


Figure 2: *None of the buildings inside region B has large size or is close to the viewpoint, but they together form good occlusion.*

ure 2). Besides, urban models are always deep; namely at most viewpoints, a significant number of buildings are far away from the viewpoint. As the viewpoint moves continuously, faraway buildings that are occluded remain so for a "long" period of time. Thus strong temporal coherence — the scene does not change much within two consecutive frames — exists in urban walkthrough applications.

In this paper, we present a simple and fast occlusion-culling algorithm for urban environments. The algorithm selects the occluders based on a novel measure of importance. The key features of our algorithm are its selection of effective occluder and its exploitation of temporal coherence by means of occluder set shrinkage. The algorithm performs occlusion culling in only a small subset of all the frames due to the utilization of temporal coherence. The shrinking allows tradeoff between accuracy and efficiency. For the purposes of this paper, we assume the input model to be 2.5D, although our algorithm can be extended to the 3D case. Given a hierarchical representation of the scene, the proposed algorithm does not require any pre-processing or prior knowledge about the walkthrough path. It computes and maintains the occluder set and the necessary visibility information in an on-line fashion, and can update the scene dynamically. The algorithm is simple and can be integrated with most existing occlusion-culling algorithms to improve their culling rate at little extra cost.

The resulting algorithm has been implemented on a SGI Octane Mips R10000 platform, and tested on both static and simulated dynamic environments. Considerable speedup in both culling rate and overall frame rate has been achieved, as demonstrated by the experimental results.

The remainder of the paper is organized as follows. Section 2 gives a review of previous work. Section 3 describes the outline of the algorithm, while Sections 4 – 6 elaborate on the key stages of our algorithm. Section 7 presents the results and performance analysis, and finally, Section 8 concludes by discussing future work and open problems.

2. Previous Work

Cohen-Or *et al.* [4] survey recent results on occlusion culling and visibility. In what follows, we distinguish between two classes of occlusion-culling algorithms: preprocessing approaches and online approaches.

Preprocessing methods typically partition the view space into cells, then pre-compute and store visibility information for each region [6, 12, 15, 17, 18, 19, 20]. Occluders fusion is inherently difficult to be computed for a view cell, and some approaches exploit the idea of "virtual" occluders [2, 9, 16]. For the special case of urban walkthroughs, Cohen-Or *et al.*[5] provide a modeling method for densely occluded city data sets and pre-compute hidden buildings for each view cell. Wonka *et al.* [22] apply a shrinking idea in the object space and cull maps in the image space to perform visibility preprocessing. The above approaches are fast during real-time applications, but are considerably costly with respect to time and memory during the preprocessing, and may not be generalized to dynamic scenes well.

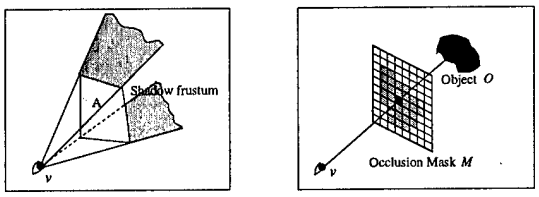
Unlike the above preprocessing approaches, most on-line methods perform little preprocessing and apply occlusion tests and culling with respect to current viewpoint at each frame. Many of them [3, 7, 8, 11] preselect a few large occluders and do on-line computation in object space. Zhang *et al.* [23] use the image-space idea and store the "opacity" information into a hierarchical occlusion map, which are generated with the help of texture-mapping graphics hardware, though they still need to preselect a set of occluders (see also [10]). Wonka *et al.* [21] apply a similar cull map idea using z-buffer to urban environments and allow dynamic occluders selection. One different approach proposed by Klosowski *et al.* [13, 14] is to render on a budget (on demand) using a novel prioritized-layered projection technique. Our algorithm takes a similar idea for occluder selection.

In some ways, our algorithm is similar to the approach of Wonka *et al.* [22] — we also use the shrinking idea and focus on urban walkthroughs. But as illustrated later in the paper, we provide a much faster on-line shrinking process in the image space and solve the problem caused by shrinking objects separately. As in some earlier approaches [9, 10, 21, 23], our algorithm utilizes the graphics hardware to fuse an occlusion mask to do culling in the image space. But we take further advantage of the strong temporal coherence existed in urban environments and perform an image-space culling only once per several frames.

3. Algorithm Outline

We propose an on-line occlusion-culling algorithm for walkthrough applications in dynamic urban environments. Although our occlusion culling algorithm might generate wrong pixels in the resulting image, thus not being conservative, we can obtain a tradeoff between accuracy and culling rate. To the best of our knowledge, our algorithm is the first on-line algorithm to utilize the temporal coherence in the image space. The algorithm computes a hierarchical occlusion mask with the help of the graphics hardware, and

© The Eurographics Association 2001.



(a) (b)

Figure 3: *Occlusion test: (a) shadow frustum in object space; (b) mask in image space.*

marks a subset of objects that are occluded by the mask. It also computes a conservative estimate of the time, called *time stamp*, until when all of these objects would remain occluded. Therefore all objects whose time stamps are greater than the current time are currently occluded. Since we use "time" instead of one bit to mark the occluded objects, we do not have to unmark them when they are no longer occluded — we simply compare their time stamps with the current time. Our algorithm also supports dynamic insertions and deletions of new objects during the walkthrough.

Most of the early culling algorithms perform occlusion culling in the object space, where all objects inside the shadow frustum formed by the occluder and the viewpoint are culled. This approach becomes impractical when there are relatively large number of occluders, and occluders fusion is required. We extend the image-space approach proposed by Green [10] to do visibility tests using occlusion masks.

Occlusion masks: For a single viewpoint v, an occlusion mask is a regular 2D grid on the image plane. Each cell stores the maximum depth value of all the objects visible inside this cell, where depth refers to the distance of the object from the current viewpoint. An object O is occluded by a mask M if for any point p on O, the depth of p is greater than the value stored in the cell on the mask intersected by the segment vp. See Figure 3 (b). In our implementation, an occlusion test is performed by an overlap test followed by a depth test.

Critical frames: As we mentioned earlier, at some of the frames our algorithm recomputes the visibility information and masks a subset of the objects that are not visible for several consecutive frames. We call these frames *critical*. In current implementation, the critical frame happens when (i) real time reaches the value of the time stamp, (ii) a dramatic change happens in the view-direction, or (iii) an occluder is deleted from the scene.

We preprocess the set of input polygons and store them into a kd-tree T, which induces a hierarchical subdivision of the object space. The algorithm then does the following at each frame:

I. If it is a critical frame, it performs the occlusion marking operation as follows:

(I.1) Choose a set of occluders.
(I.2) Apply the image-space shrinking algorithm.
(I.3) Generate a hierarchical occlusion mask.
(I.4) Compute a time stamp and mark all objects lying behind the mask with this value of the time stamp.

II. For any frame:

(II.1) If the environment has changed, then perform the dynamic update algorithm.
(II.2) Pass all objects that are inside the view frustum and whose time stamps are less than the current time to the graphics hardware.

4. Preprocessing of the Input

Let S be the set of input polygons. Since a city model is 2.5-dimensional, namely all objects are placed on top of a ground plane, we use a invariant of 2D kd-tree to store the xy-projections of the polygons in S, along with the height information. The data structure is constructed as follows.

For each polygon $\triangle \in S$, let $B_\triangle$ be the smallest orthogonal box containing $\triangle$, and let $p_\triangle$ be the xy-projection of the bottom-left corner of $B_\triangle$. We construct a 2D kd-tree T on the point set $P = \{p_\triangle \mid \triangle \in S\}$. Each node τ of T is associated with a subset $P_\tau \subseteq P$ and a rectangle R_τ, which is the smallest enclosing rectangle of P_τ. We also associate a 3D box B_τ, which is the smallest orthogonal box containing all polygons $\triangle$ such that $p_\triangle \in P_\tau$. For the root u of T, $P_u = P$ and $R_u = \mathbb{R}^2$. If $|P_\tau|$ is less than a certain parameter ρ, then τ is a leaf. At each leaf τ, we store the set of polygons $\{\triangle \mid p_\triangle \in P_\tau\}$. Otherwise, we choose an axis-parallel line l_τ, called *splitter*, and partition R_τ into two rectangles, each of which is associated with a child of τ. Points of P_τ lying in each of the rectangles are associated with the corresponding child of τ.

There are many possibilities of choosing a splitter l_τ. We could simply bisect the points in P_τ and alternate between horizontal and vertical splitters, or we could use a more sophisticated method.

Note that the rectangles associated with the leaves of T are disjoint, but the xy-projections of the bounding boxes associated with the leaves could intersect because they bound the polygons. Each polygon is stored at only one leaf. The interior nodes do not store the polygons. The total size of the data structure is $O(n)$, where n is the number of input polygons.

5. Occlusion Marking Operation

At each critical frame, the algorithm first performs an occlusion-marking operation. The goals of this step are to take advantage of the temporal coherence, to choose a set of

© The Eurographics Association 2001.

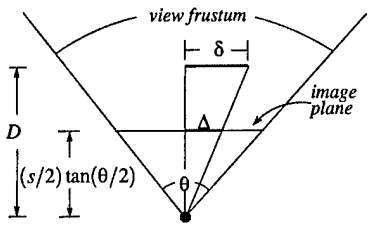


Figure 4: *The relationship between shrinkage in image-space and in object-space.*

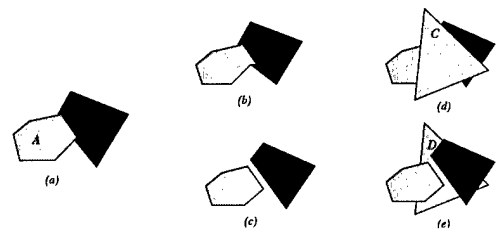


Figure 5: *Shrinking the union of projections may cause a leak as shown in (b). But this leak will not cause error on the resulting image if other objects (as C in (d)) or other occluders (as D in (e)) still cover it.*

occluders, and to generate the hierarchical occlusion masks such that the marked objects would remain occluded for several subsequent frames.

5.1. Temporal coherence

Let v be the current viewpoint, and let $B_\delta(v)$ be a ball of radius δ centered at v. For an object σ, the *shrinking* of O by δ is the Minkowski difference $\sigma \ominus B_\delta = \{x \mid B_\delta(x) \subset \sigma\}$. The following lemma is straightforward.

Conservative Visibility Lemma: *Let Σ be an occluders set, and let v be the viewpoint. $\Sigma' = \{\sigma \ominus B_\delta \mid \sigma \in \Sigma\}$. If a point p is occluded from v by Σ', then p is occluded by Σ from any $v' \in B_\delta(v)$.*

The lemma suggests that if we use Σ' instead of Σ to do occlusion culling at the viewpoint v, the culling would remain valid as long as the viewpoint remains inside $B_\delta(v)$. Shrink occluder P by δ, and let P' be the new polygon. If a polygon Q is occluded by P' and the viewpoint is moving with a maximum velocity V at the moment, then Q will remain occluded by P for the next δ/V time units. In such a case, one can set the time stamp to $t_c + \delta/V$, where t_c is the current time.

In the image space, how much the projection of a polygon P shrinks as we shrink P by δ in the object space depends on the distance between P and the viewpoint. More precisely, let the image resolution be $s \times s$, the minimum distance between P and v be D, the angle formed by the view frustum be θ, and let Δ be the amount by which we shrink the projection of P (see Figure 4). Then

$$\delta \geq 2\Delta \cdot \frac{D}{s\tan(\theta/2)}. \qquad (1)$$

The analysis indicates that by shrinking the projection of each occluder P in the image space by Δ, the resulted image, i.e., fusing the shrunk projection into one occlusion mask, is conservative in visibility for all viewpoints inside $B_\delta(v)$.

5.2. Image-space shrinking and fusing

Shrinking each occluder in object space separately has two major disadvantages: (i) it is expensive and complicated [22]; and (ii) unnecessary leaks may appear if two connected polygons are shrunk independently. The problems are further exacerbated if the model is finely tessellated or if the input is given as a set of polygons without any connectivity information between them. Our algorithm instead shrinks the union of the projections of occluders on the image plane. Thus the algorithm puts no restrictions on the input data and preserves the connectivity between polygons during shrinking.

Shrinking the union may cause error sometimes, such as illustrated in Figure 5. Object A is in front of Object B, but their projections overlay each other. The result of shrinking them separately is depicted in Figure 5 (c), which contains a small leak where a third object could be visible if no other objects were covering this slit and thus hiding the object from the viewpoint. However, if one shrinks the union of the two objects together, as depicted in Figure 5 (b), the leak does not appear and the resulting visibility is thus approximate. The maximum size of each error on the resulting image is bounded by the shrinkage Δ.

However, if we consider the visual error, i.e., the misdrawn pixels on the resulting image, the slit as illustrated in Figure 5 (c) would produce little or no error since: (i) as will explain later, the occluders chosen by the algorithm are relatively far from the viewpoint, there is high probability that closer objects would occlude the slit (see Figure 5 (d)); (ii) we choose a "thick" layer of occluders (we will address this issue in Section 5.3), thus other occluders may cover this slit (see Figure 5 (e)); (iii) Δ is small and thus the visible portion through the slit is tiny; and (iv) the user is generally walking along the streets, and buildings along the streets and objects close to him are the focus of the view, while the errors occur in places of less visual importance. So, although larger Δ potentially allow larger slits, as we will see in Section 7, errors rarely occur.

Though larger Δ means more possible errors and less coverage on the occlusion mask, it can increase the frame rate

© The Eurographics Association 2001.

(the speed), until it reaches the point that it damages the culling rate more severely. So Δ is an important parameter in the tradeoff between accuracy and speed.

5.3. Occluder selection

We use the following criteria to design the occluder selection algorithm: (i) the selection process is fast, (ii) R, the number of occluders in $\mathcal{O}$, is not too big, (iii) the mask generated is well-covered, and (iv) the distance between an occluder and the viewpoint v is at least some parameter D in order to take advantage to temporal coherence (refer to Equation (1)).

Since the occlusion mask is generated in the image space with the help of graphics hardware, our algorithm can afford to choose a relatively large set of occluders than allowed by the object-space approaches. The influence of R will be addressed in Section 7.1. Given a fixed number R, the goal of the occluder-selection algorithm is to find R most "important" objects among all the objects that lie inside the view-frustum and whose minimum distance from v is at least D. The "importance" of an object or a node in the kd-tree refers to their contribution to the occlusion mask, namely, how well they occlude objects that are farther away from the viewpoint. For a node ξ of T, let $\phi(\xi)$ represent the inverse of "importance" value of a kd-tree node ξ, thus a smaller $\phi(\xi)$ means node ξ is more important. The occluder-selection algorithm is depicted in Figure 6. The algorithm works by maintaining a priority queue of the kd-tree nodes to be visited, based on the value of ϕ.

```
ALGORITHM Occluder-selection(T, v, R)
  Input: kd-tree T, viewpoint v, #occluders R
  Output: R polygons chosen as occluders.
    Insert root(T) into Q;
    while ((Q ≠ ∅) and (count < R))
      ξ = get-min(Q);
      if (size(ξ) is small) and (mindist(ξ, v) ≥ D)
        Output polygons in ξ as occluders;
        update count;
      else
        Insert two children of ξ into Q;
      end if
    end while
  end Occluder-selection
```

Figure 6: *Occluder selection algorithm; get-min(Q) returns the node in Q with the minimum ϕ value, and mindist(ξ, v) returns the minimum distance between ξ and v.*

We can simply choose $\phi(\xi)$ to be the minimum distance between ξ and v. This is equivalent to choosing all objects in an annulus with an inner radius D; the size of the annulus depends on the value of R. See Figure 7 (a) for illustration. This works well for dense models with almost uniform distribution of buildings.

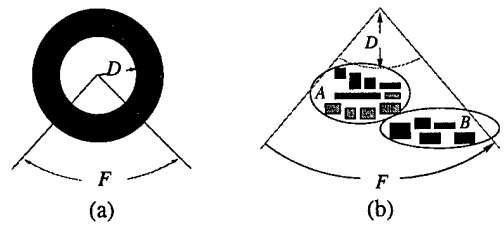


Figure 7: *F is the view frustum. D is the minimum distance. In (a), objects in the dark region will be chosen as occluders based on distances. In (b), buildings in region A will be chosen as occluders based on distances. Buildings in B are missed, although they contribute a lot to the occlusion mask too.*

However, it ignores the objects that are far away but nevertheless contribute significantly to the occlusion mask. Figure 7 (b) illustrates one such scenario, where the above distance criteria only choose the building in region A even though the buildings in region B would be good occluders.

In the following, we consider only the objects that are inside the view frustum and are at least D distance away from v. Intuitively, the definition of $\phi(\xi)$, for a node ξ in the kd-tree, should depend on how many polygons closer than ξ occlude the objects stored at ξ. In other words, if ξ covers a spot s on the image plane, and several closer polygons (partially) have already covered s, then ξ is not a good candidate to be an occluder, as it potential contribution to the occlusion of s is small (i.e., see the buildings with light color in Figure 7 (b)). We therefore use the following approach.

We divide the image plane into cells, and assign a "coverage" value $C(c)$ for each cell c. At any moment during the occluder-selection algorithm, $C(c)$ is defined as the number of polygons encountered so far whose projections overlap c. For a kd-tree node ξ, we define

$$\phi(\xi) = \min_{i=1..k}\{C(c_i)\} * \text{mindist}(\xi, v),$$

where c_i, for $i = 1, \ldots, k$, are sampled cells that the projection of ξ covers, and mindist(ξ, v) returns the minimum distance between node ξ and viewpoint v. In the current implementation, k is set to be 3. More precisely, for a node ξ with a bounding box B_ξ, we choose 3 points, namely, the center point(p_1) of B_ξ, and the two endpoints (p_2 and p_3) of one diagonal of B_ξ, and c_i is the cell that ray vp_i passes through. Whenever a node ξ is added to the occluders set, each cell it covers updates its coverage to

$$C(c_i)_{new} = C(c_i)_{old} + (\#\text{ polys in } \xi)/(\#\text{cells } \xi \text{ covers}).$$

Figure 8 illustrates the idea in 2D.

We have implemented both methods described above for computing $\phi(\xi)$. Figure 9 shows the different outputs of the occluder-selection algorithm under the same viewpoint and

© The Eurographics Association 2001.

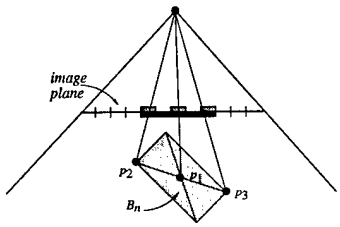


Figure 8: *The 3 light cells are chosen to decide $\phi(\xi)$ for node ξ, while the dark cells will be updated if polygons in ξ are outputed as occluders.*

(a) (b)

Figure 9: *In both city maps, dark polygons are occluders selected by the algorithm. The value of ϕ is defined by distance only in (a), and by the more complicated method in (b).*

view direction, using those two different versions of ϕ. The occluder selection based on the combined criteria is slightly slower than the distance criteria. However, since it results in better culling rate and the time spent in this step is relatively insignificant (refer to Figure 10), it overall results in faster frame rates.

5.4. Hierarchical occlusion mask

Our algorithm sends all the occluders to the graphics hardware and reads the contents of the z-buffer. The background is assumed to be at infinity with the maximum depth value. We call the non-background regions in the z-buffer *occluder regions*, which is the union of the projections of occluders on the image plane. The algorithm shrinks the union by Δ and then computes the time stamp using (1). However, since we exploit the standard OpenGL rasterization, the z-buffer is not a conservative mask for current viewpoint due to the partially covered but drawn pixels on the silhouette edges. Similar to the technique proposed by Wonka *et al.* [22], our algorithm shrinks one extra pixel to guarantee that only fully covered pixels would be counted. The shrinking operation mentioned above is the same as the erosion operation in image processing community.

After the algorithm shrinks the occluder regions in the image acquired from the z-buffer by $\Delta + 1$ pixels, the resulting image serves as a *primary occlusion mask* M_0. In order to accelerate the occlusion test operation against the mask, in the implementation, as in Zhang [23], our algorithm uses a set of hierarchical masks $M_0, M_1, \ldots, M_m$. Starting from the primary mask M_0, the hierarchy is built up by creating lower resolution versions of M_0. We fix a parameter b, each pixel in M_{i+1} is obtained by combining a block B of $b \times b$ pixels of M_i. The value for this pixel in M_{i+1} is the maximum z-value in B, which guarantees that occlusion tests involving M_{i+1} are conservative.

Zhang *et al.* [5] accelerate the construction of their hierarchical occlusion maps by graphics hardware that supports bilinear interpolation of texture maps. The step is made faster, but can introduce artifacts. Our conservative mask generation algorithm is slower but is performed at only a subset of all the frames.

5.5. Marking algorithm

We traverse the T in a top-down manner to mark the occluded nodes of T with the computed time stamp . At each node τ, we use the hierarchy of masks and the 3D bounding box B_τ to speedup the occlusion test. The algorithm traverses T as follows: For the current node τ, let B^*_τ denote the smallest orthogonal rectangle enclosing the projection of the box B_τ on the image plane. If B^*_τ is occluded by the masks, the algorithm marks τ. Otherwise, it recursively visits the children of τ. In order to determine whether B^*_τ is occluded, depending on the size of B^*_τ, the algorithm first selects an appropriate level of mask M_i. It then searches the hierarchy of masks, starting from M_i, with B^*_τ in a standard manner. We check all cells $b \in M_i$ that intersect B^*_τ. If $b \subset B^*_\tau$ is not covered, we conclude that B^*_τ is not occluded. If b intersects B^*_τ partially, we recursively check the cells of M_{i-1} lying inside b to determine whether $B^*_\tau \cap b$ is occluded by the occluders.

At each node τ of T, we also store the number of objects stored in the subtree rooted at τ. During the marking step, instead of going all the way to a leaf, we stop visiting the descendants of a partially visible node if only few objects — below a chosen threshold ρ — are stored in the subtree because the time spent in determining the visibility of these nodes will offset the time saved in not sending the occluded objects. We refer to the threshold ρ, which determines where to terminate the visibility test, as *leaf size*.

6. Dynamic Update Algorithm

We simulate the dynamic scenes by inserting or deleting some random buildings at some randomly chosen frames. A "lazy" approach is employed to perform the update. We omit the details here and related experimental results later because of lack of space from current short abstract version.

© The Eurographics Association 2001.

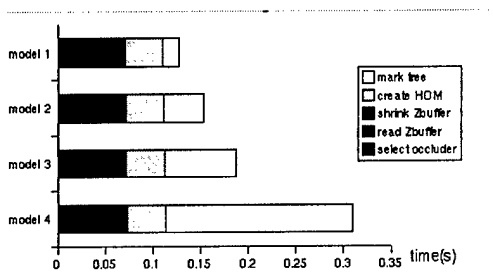


Figure 10: *The time consumed by each sub-step of an occlusion marking operation for the 4 models.*

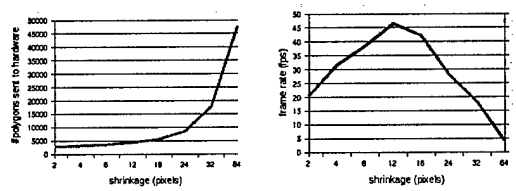
7. Implementation and Performance

We tested the performance of the above algorithm on an SGI Octane Mips R10000 with a 196MHZ CPU and 128MB main memory. The program provides a GUI for the user to select a walkthrough path. All of our testing paths are along the streets since this is the most realistic case. We demonstrate the performance of our algorithm on four sets of city models of Manhattan suburb and middle Manhattan. The sizes of the models are model 1 with 3,657 polygons, model 2 with 27,437 polygons, model 3 with 109,748 polygons, and model 4 with 438,992 polygons. All of them consist only of buildings, and each building is composed of a few polygons. Most polygons are quadrangles, though there are also a small number of polygons with more than 4 vertices.

7.1. Analysis of parameters

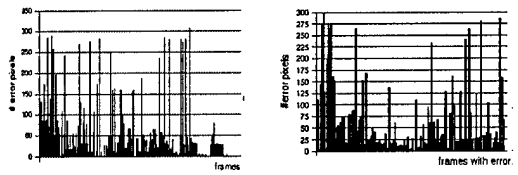
We split one marking operation into five steps: (i) selecting occluders, (ii) drawing occluders and reading z-buffer , (iii) shrinking to get the primary mask M_0, (iv) generating the hierarchical occlusion masks, and (v) marking the kd-tree T. Figure 10 shows the average time taken by each sub-step for four different data sets. Note that time spent by sub-steps (i) –(iv) does not vary much as the size of data sets grows, since it is mainly determined by the graphics hardware configurations. The time of the last step dominates the overall time. It increases as the dataset becomes larger because the algorithm has to do occlusion test on more kd-tree nodes against the mask for larger data sets.

Several crucial parameters are involved in each step, such as the minimum/maximum distance of the occluders, the size and the number of levels of the hierarchical occlusion masks, the size of the leaves ρ in T, and the shrinkage Δ in image space. The final frame rates are determined by the overhead of the occlusion marking operation occurred at each critical frame, by the frequency of critical frames, and by the culling rate. All these parameters have a compound effect on the frame rate. We performed many experiments with different



(a) (b)

Figure 11: *Shrinking size of the mask can decrease the culling rate as depicted in (a), but (b) shows that it still enhances the frame rate until the shrinkage is too large.*



(a) (b)

Figure 12: *(a) Number of error pixels for each frame tested on Sun Ultra 5. (b) Number of misdrawn pixels for frames in which error occurs , tested on a SGI Octane. $\Delta = 6$ pixels in both tests.*

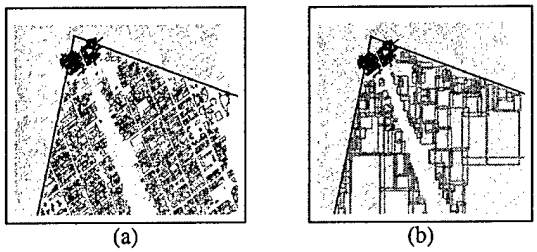
configurations to inspect their influence, but omit the results and analyses from current short paper. Interested reader are referred to the full version [1].

7.2. Tradeoff between accuracy and speed

Figure 11 (a) depicts how Δ affects the number of polygons sent to the graphics hardware (culling rate) for model 3 consisting of around 100,000 polygons. As the value of Δ increases, it has little affect on the culling rate in the beginning, but after a while, it starts reducing the culling rate since the coverage on the occlusion mask is too sparse. Eventually, the number of polygons sent to the hardware converges to the number of polygons inside the view frustum. The curve in (b) reflects the relationship between Δ and the overall frame rates. As the figures show, before Δ reaches the point that it starts to reduce the culling rate significantly, it enhances the frame rates. On the other hand, larger Δ potentially allow more error. Thus before Δ meets the threshold, the shrinking size provides a way to achieve tradeoff between accuracy and speed.

We tested the size of error, i.e., the number of pixels misdrawn on the resulting image compared with displaying the scene using the view-frustum culling algorithm. Namely, we read the z-buffer and frame buffer from the hardware after applying our algorithm and the view-frustum culling algorithm respectively, and then compare these buffer. We performed the same algorithm on two platforms, a SUN Ultra 5

© The Eurographics Association 2001.



(a) (b)

Figure 13: *Light grey city maps are overlapped with black view frustum and black occluders in both figures. Dark grey building in (a) are marked occluded by our algorithm, while dark grey boxes in (b) are the bounding boxes of the marked nodes.*

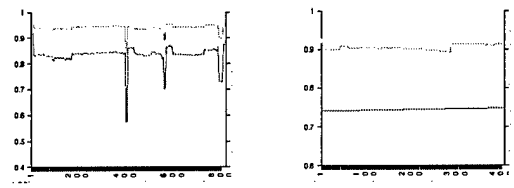
and a SGI Octane, and observe very different number of misdrawn pixels. Figure 12 (a) shows the number of error pixels at each frame for a path consisting of 800 frames. However, very few error pixels appear for the same path when tested on the SGI platform. So we instead apply the algorithm to a long path (11,558 frames) throughout the city model on the SGI machine. Errors only appear in 158 frames. See Figure 12 (b). Experiments for other models show similar patterns. Furthermore, we did not observe any substantial increase in the visual error as Δ becomes larger. This somewhat counterintuitive behavior, follows from the fact that closer objects and other occluders cover the leaks produced. Also, note that our analysis is rather conservative, and as such, in practice, it is rather "pessimistic" in estimating the visual errors.

7.3. Performance

We demonstrate the performance of the algorithm by comparing the culling and frame rates with the view frustum-culling algorithm. Our main purpose is to prove the effectiveness of our algorithm, so the algorithm has not focused on accelerating the frame rates using graphics hardware such as using display lists or triangle strips. Instead, we send a set of polygons in each frame.

In Figure 13, the viewpoint is on a very long street, with the view direction along that street. Figure 13 (a) shows that the culling is effective since most unmarked objects are those buildings along the street sides. In that case, other culling algorithm would only do worse if they are not careful about the occluders set. The efficiency of the culling is shown in Figure 13 (b) where the dark boxes are the bounding boxes of the nodes marked occluded in the kd-tree. Most marked nodes are close to the root, except for those whose projections are on the boundary of the occluded region. In this specific case, nodes close to the streets and near the view frustum have to be broken down to lower level during the marking.

In our algorithm, a polygon is culled either because it is



(a) (b)

Figure 14: *Culling rates achieved by our algorithm for (a) model 2 (27,437 polygons) and (b) model 4 (438,992 polygons). In both plots, the upper curve shows CR_o while the lower curve shows CR_v.*

Data size	Hardware only	Viewfrustum culling	Our algorithm
3,657	328.57	58.93	56.63
27,437	9.26	14.78	41.27
109,748	1.86	3.26	38.53
438,992	0.43	1.17	20.13

Table 1: *Frame rates using (i) z-buffer directly, (ii) viewfrustum culling, and (iii) our algorithm. The unit for the frame rate is frames/sec.*

outside the view frustum, or alternatively, an ancestor of the leaf storing the polygon is marked occluded. Let

$$CR_v = \frac{\text{\#polygons culled by the algorithm}}{\text{\#polygons inside view frustum}},$$

and

$$CR_o = \frac{\text{\#polygons culled by the algorithm}}{\text{\#polygons in the data set}}.$$

The value of CR_v shows the improvement in the culling rate performance achieved by our algorithm over the view frustum culling approach, while CR_v refers to the culling rate compared to the original data set. The graphs depicted in Figure 14 show the changes of these two culling rates with respect to time. The high culling rate is achieved because the algorithm includes most of the "useful" objects as occluders.

Table 1 shows the frame rate (averaged over 800 frames) obtained by our algorithm. There is no benefit in using our new algorithm for small data sets due to the overhead at critical frames. The big gains appear when our algorithm is applied to medium to large inputs, where our more aggressive culling pays off. In the current implementation, the time required at each frame is not balanced since a critical frame needs to perform the expensive occlusion-marking operation. As the rendering and occlusion-marking steps can be performed independently, we can use multiple threads to perform these two steps, which would amortize the cost of occlusion marking over several frames.

© The Eurographics Association 2001.

Acknowledgments

We would like to thank Claudio Silva for helpful comments.

References

1. P. K. Agarwal, S. Har-Peled, and Y. Wang. An on-line occlusion culling algorithm for fast walkthrough in urban areas. http://www.cs.duke.edu/~wys/research. 7
2. F. Bernardini, J. El-Sana, and J. T. Klosowski. Directional discretized occluders for accelerated occlusion culling. In *Proc. Eurographics Conf.*, volume 19, 2000. 2
3. J. Bittner, V. Havran, and P. Slavik. Hierarchical visibility culling with occlusion trees. In *Proc. of Comp. Graphics Internat.'98*, pages 207–219, 1998. 2
4. D. Cohen-Or, Y. Chrysanthou, and C. T. Silva. A survey of visibility for walkthrough applications, 2000. Course notes of EUROGRAPHICS '00. 2
5. 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. 2, 6
6. D. Cohen-Or and E. Zadicario. Visibility streaming for network-based walkthroughs. *Graphics Interface*, pages 1–7, 1998. 2
7. S. Coorg and S. Teller. Temporally coherent conservative visibility. In *Proc. 12th Annu. ACM Sympos. on Comput. Geom.*, pages 78–87, 1996. 2
8. S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. In *Proc. of the ACM SIGGRAPH Sympos. on Interactive 3D Graphics*, pages 83–90, 1997. 2
9. F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. In *Proc. of SIGGRAPH '00*, pages 239–248, 2000. 2
10. N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. In *Proc. of SIGGRAPH '93*, pages 231–240, 1993. 2, 3
11. T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frusta. In *Proc. of the 13th ACM Sympos. on Comput. Geom.*, pages 1–10, 1997. 2
12. W. F. H. Jimenez, C. Esperanca, and A. A. F. Oliveira. Efficient algorithms for computing conservative portal visibility information. In *Proc. Eurographics Conf.*, volume 19, 2000. 2
13. J. T. Klosowski and C. T. Silva. Rendering on a budget: A framework for time-critical rendering. In *IEEE Visualization'99*, pages 115–122, 1999. 2
14. J. T. Klosowski and C. T. Silva. Efficient conservative visibility culling using the prioritized-layered projection algorithm. 2000. manuscript. 2
15. V. Koltun and D. Cohen-Or. Selecting effective occluders for visibility culling. In *Proc. Eurographics Conf.*, 2000. 2
16. G. Schaufler, J. Dorsey, X. Decoret, and F. Sillion. Conservative volumetric visibility with occluder fusion. In *Proc. of SIGGRAPH '00*, pages 229–238, 2000. 2
17. J. Stewart. Hierarchical visibility in terrains. In *Eurographics Rendering Workshop*, pages 217–228, 1997. 2
18. S. Teller and P. Hanrahan. Global visibility algorithms for illumination computation. In *Proc. of SIGGRAPH '93*, pages 239–246, 1993. 2
19. S. Teller and C. Sequin. Visibility preprocessing for interactive walkthroughs. In *Proc. of SIGGRAPH '91*, pages 61–69, 1991. 2
20. Y. Wang, H. Bao, and Q. Peng. Accelerated walkthroughs of virtual environments based on visibility processing and simplification. In *Proc. Eurographics Conf.*, volume 17, pages 188–194, 1998. 2
21. P. Wonka and D. Schmalstieg. Occluder shadows for fast walkthrough of urban environments. *Computers and graphics*, 23(6):831–838, 1999. 2
22. P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. Technical report, Technical Report TR-186-2-00-06, Institute of Computer Graphics, Vienna University of Technology, March 2000. 2, 4, 6
23. H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility culling using hierarchical occlusion maps. In *Proc. of SIGGRAPH '97*, pages 77–88, 1997. 2, 6

© The Eurographics Association 2001.