A Visibility Pipeline toward Real-time Rendering Based on the Hardware-accelerated Occlusion Query

Wen-Kai Tai and Chih-Kang Hsu

Department of Computer Science and Information Engineering
National Dong Hwa University
No. 1, Section 2, Da Hsueh Road, Shou-Feng,
Hualien, 974, Taiwan, R.O.C.
TEL: +886-3-8634023 FAX: +886-38634010
EMail: wktai@mail.ndhu.edu.tw

Abstract

In this paper, we propose a visibility pipeline based on hardware-accelerated occlusion queries to render complex and dynamic walkthrough environments in real-time. The input scene is represented by a regular grid and organized as an octree-like hierarchy. A 2-tier view frustum culling taking advantages of intersection test and occlusion query is proposed to efficiently cull away nodes invisible from a given viewpoint. With the novel encoding mechanism, eye-siding number, nodes in the hierarchy are encoded by its eye-siding. While traversing the hierarchy for occlusion evaluation, one can efficiently enumerate an occlusion front-to-back order for all nodes by the eye-siding number, and this number can be further used to effectively maximize the number of parallelizable occlusion queries. As rendering performance trades for image quality, we propose the importance and contribution culling techniques based on exploiting the returned pixel count of occlusion query coupled with the stencil test to the prespecified importance mask.

As the experimental results show, the proposed visibility pipeline improves the rendering performance; 2-tier view frustum culling approach is more efficient, maximizing the number of parallelizable occlusion queries makes the utilization of hardware-accelerated mechanism even better, faster frame rates, and occlusion query together with the importance and contribution culling methods produces acceptable image quality but gains more efficiency.

Key words: visibility, Visibility Pipeline, Occlusion Query, 2-tier View Frustum Culling, Occlusion Culling, Importance Culling, Contribution Culling, Eye-siding Number, Occlusion Front-to-Back Order, Multiple Hardware-accelerated Occlusion Query.

Preprint submitted to SIMPRA SPECIAL ISSUE on Programmable Graphics Hardware

1 Introduction

The visibility techniques have been studied extensively in interactive walkthrough applications. Generally, there are large amount of objects in most of walkthrough environments. Though the number of polygons that can be processed by GPU in a second is rapidly rising, the bottleneck is tending toward the bandwidth between main memory and graphic card. To reduce the bandwidth, the view frustum culling (VFC) is used to cull away objects outside the view frustum from a viewpoint, the occlusion culling (OC) is exploited to prune the occluded objects, and the importance culling (IC) and contribution culling (CC) can be used to avoid rendering objects which contribute to a worthless portion of the resultant image or too few pixels to be noticed respectively. In this paper, we propose a visibility pipeline which provides a real-time rendering for interactive walkthrough environments with large amount of objects by exploiting the hardware-accelerated occlusion query (OQ) for efficiently performing VFC, OC, and IC and CC.

A regular grid is used to represent the input scene, and it is organized as an octree-like hierarchy. In each frame, the actual position of dynamic objects is update for each cell, voxel, of the grid. Conceptually, each voxel is extended to overlap each other such that every object belongs to exactly one voxel (overlapping voxel). With the grid representation, the corresponding octree-like hierarchy is effective to construct and the actual position update for dynamical objects in each frame can be done efficiently. While traversing the hierarchy, the visiting nodes are enumerated in an occlusion front-to-back order using the concept of eye-siding number. Instead of using intersection test only for performing VFC, a 2-tier mechanism that takes advantages of the efficiency of intersection test and the occlusion effectiveness of OQ is employed to cull away nodes outside the view frustum. As for nodes inside the view frustum, the OC is then performed to prune them away if they are occludees. The hardware-accelerated OQs are invoked for evaluating occlusions. To maximize the performance of OQs, nodes in an occlusion front-to-back order are further grouped into parallelizable units using their eye-siding number again. All nodes in one unit invoke OQs at a time. The OQ returns the number of visible pixels. This pixel count is not only used to determine the occlusion but also can be used to judge the pixel contribution at the rendered image. Given a contribution threshold, nodes passing the OC is still culled away if their returned pixel count is less than that threshold. Moreover, we can manually or procedurally specify an importance region of a rendering image in the stencil buffer. Whenever evaluating occlusion using OQ, the stencil test is enabled and a node is visible if objects inside the node project to the importance region and the number of rendered pixels in the region is over the contribution threshold.

We summarize our contributions as follows:

[**Visibility Pipeline**] To our knowledge, it is the first time that a visibility pipeline based on hardware-accelerated OQ is proposed to provide real-time rendering for complex walkthrough environments. Before start rendering, all objects go through the visibility pipeline, namely performing VFC, OC, IC and CC for them. All culling mechanism exploits the hardware-accelerated OQs to provide as maximal efficiency as possible. As experimental results show, the overall performance of our approach is more efficient than methods proposed. Also, together with the importance and contribution culling techniques the rendering can be even faster as the approximate rendering result is still acceptable.

[**Importance Culling**] Sometimes we focus on a portion of the display rather than keep watching the whole display area. User can flexibly define the important regions on a screen as a mask. We take advantage of hardware acceleration by preloading the mask to the stencil buffer and enabling the stencil test to perform importance culling while evaluating the occlusion using OQ. An Object is visible if the returned pixels of OQ pass the stencil test. This culling approach produces acceptable rendering result but significantly improves the frame rates.

[**Efficient Hierachy Maintenace**] The actual position of dynamic object needs to be updated for each frame so the object list in a node keeps consistent. Conceptually extending the voxels of a grid as overlapping voxels, each object can be uniquely assigned. Plus, the scene is scaled such that each voxel becomes a unit cube. Hence, the voxel to which a dynamic object belongs can be efficiently determined by eliminating the decimal parts of coordinates of its bounding box's minimum vertex.

[**2-tier View Frustum Culling**] Given a viewpoint, VFC is performed first and then OC. We exploit OQ in the VFC together with intersection test. Invoking OQs for large nodes costs high and the occlusion effectiveness is low, instead the intersection test is used. While the node size is smaller at a deeper level and the occlusion possibility is high, the OQs are invoked for VFC. The proposed 2-tier mechanism is efficient for culling invisible object against the view frustum. Besides, we introduce a near face intersection test method to make OQ robust.

[**Maximum the Number of Parallelizable Node Occlusion Queries**] Hardware-accelerated OQ is efficient. However, waiting for the result of an OQ stalls the rendering pipeline. The more multiple queries sent for occlusion evaluation at a time, the better performance gained. The concept of eye-siding number is not only used to enumerate an occlusion front-to-back order for all nodes, but also nodes with the same eye-siding number in an occlusion

front-to-back order sequence can be grouped into a set of parallel units in our approach. Nodes in a parallel unit are sent for occlusion evaluation at a time, and these query results are collected later on. Maximizing the number of parallelizable occlusion queries for nodes in the hierarchy makes the utilization of hardware-accelerated mechanism even better.

The rest of this paper is organized as the follows. We first briefly dscribe related works in section 2. The proposed visibility pipeline is specified in section 3. In section 4 we show and demonstrate the experimental results. Finally, the conclusion and future works are given in section 5.

2 Related Works

A recent survey of different algorithms is given in [1]. Most OC algorithms, [2], [3], [4], [5], [6], [7], [8], [9] and [10], are conservative. Namely, the occlusion evaluation is not aimed to cull away the exactly invisible objects but usually overestimate the visible objects with respect to the viewpoint such that the rendered images are correct. However, a few approaches, [11], [12] and [13], were proposed to approximate the final image by restricting the visible set in a frame to fit some predefined requirements. These approximation techniques sacrifice visibility conservativeness for the overall performance and simplicity of implementation.

Region-based visibility techniques, [4], [10], [14], [9], [7], and [15], ordinarily computed the visibility from given regions. The regions constrain the movement of viewers. The visible objects from the region, the potentially visible sets, are computed and recorded in preprocessing stage. While rendering, the viewcell where the viewpoint locates is found and its visible objects are sent to the graphic card. In contrast, point-based visibility techniques, [3], [16], [17], [18], [19], [20], [2], [21], [12], and [22], relying on the identification of large occluders, computed the visibility on the fly with respect to the viewpoint only. Most region-based techniques work well for scenes with large convex objects as occluders and have the advantage of viewpoint coherence, i.e., without reevaluate the visibility while the viewpoint stays in the same region. However, these techniques may take long preprocessing time, require large storage space, and result in low culling effectiveness. Point-based algorithms are suitable for handling moving objects but with less effective in dealing occlusion fusion.

The occlusion evaluation for objects in a scene is eventually performed by comparing the occluded regions formed by occluders with a representation of the object. The object is an occludee if the occluded regions completely overlap it. The object-based approaches, [3], [16], [17], and [18], evaluated the occlusion

by comparing the occlusion volumes formed with raw 3D objects. The object-based approaches take advantage of spatial hierarchies, but they suffer from performing occlusion fusion for small occluders in a scene. The projection-based schemes, [23], [9], [19], [20], [2], [21], [12], and [22], performed occlusion evaluation by testing the projected region of objects to the maintained occlusion information. If the projected object is in a discrete representation, the hardware rasterization can be used to accelerate. The approaches of analytic visibility, [5], [6] and [7], exploited the geometry information of special domains and determine the visibility in the domain. The projection-based and analytic approaches can fuse the occlusion in their space of the overlap tests.

A few of approaches in OC are capable of handling dynamic objects in a scene. Sudarsky [24] adapted existing visibility algorithms to dynamic environments for minimizing the updates of dynamic objects using temporal coherence, temporal bounding volumes (TBVs). Each dynamic but invisible object is associated with a TBV and used for visibility evaluation. The complexity of this algorithm does not depend on the size of input scenes. Only potentially visible objects and expired TBVs need updating. Hence, the output-sensitivity is provided. Sudarsky's work was based on the assumption that the motion of objects is predictable. Though this is quite reasonable, it is still a limitation. Also, the hierarchy update may be still too expensive, so some successive researches turn to non-hierarchical structures. Based on Schaufler's work [9], Batagelo [23] adapted it for dynamic scenes using Sudarsky's approach [24]. Instead of spatial hierarchies, they used a regular grid to discretize the scene into voxels. Each voxel maintains a set of volumetric characteristics in its region: occluder, occlusion, identifiers and TBVs matrix. To reduce the computation of the spanning voxels for the dynamic objects, TBVs is used for hidden objects. Batagelo can take care of truly dynamic environments and the output-sensitivity is provided. Although the voxel traversal of this approach approximates the front-to-back order but cannot exploit the advantages of hierarchy schemes like other methods, [3], [25], [7] and [24]. In a densely occluded scene, this may result in more traversals with respect to the hierarchical approaches.

Recently hardware vendors, including HP, ATI, and NVIDIA, have implemented a projection based OC, OQ, in graphic card. Users can query the hardware to see if any change is made to the z-buffer while rasterizing an object. Algorithms proposed by [11], [26], [27], [28], [29], [30] and [31] evaluated the occlusion by performing OQs. If none of pixels are visible, i.e., no pixels returned from the query, then the object is occluded and culled away. These techniques really have faster performance if the scan-converted bounding boxes contain a large number of objects, and the effectiveness of OQ depends on the underlying hardware and input models. There are three approaches similar to our method on performing OC. Govindaraju [30] switched roles of two GPUs for performing OC in parallel between successive frames. The parallelism of

OQs is exploited by sending all possible OQs for the nodes at a given level in a hierarchy at a time. Nodes in a level, however, are not guaranteed in an occlusion front-to-back order. Also, the occlusion representation from the previous frame may not be a good occlusion approximation for the current frame. Hence, the culling effectiveness and image quality are somewhat not high and accurate respectively. Hillesland [29] decomposed the static scene using uniform grid and nested grid and made use of OQ to evaluate the visibility in front-to-back order determined by a variant of the axis aligned slabs. To reduce the setup cost, the pipeline is keeping busy by submitting n cells in a slab at a time, and recursively traverse the contained subgrids of a visible cell. This method is simple and fast. But, there are too many OQs sent for visibility evaluation in the scene represented by the uniform grid. Also, it is less effective on reducing the pipeline stalls that multiple OQs are only selected from a single subgrid of a visible cell for the nested grid traversal. Staneker [31] proposed the software-based occupancy map to significantly reduce the overhead of OQs and to arrange multiple OQs in a static scene browsing. The proposed method is useful for scenes with low occlusion. However, the screen space bounding rectangle is too conservative such that it tends to low occlusion effectiveness, especially in a dense environment.

3 The Visibility Pipeline

Before all objects in a scene are sent for rendering, they go through the visibility pipeline first for pruning away invisible objects and/or objects with worthless contributions to the rendered image. In the following, we first specify how the input scene is organized and maintained. Second, the 2-tier VFC is detailly described. Third, the concept of eye-siding number and how the number is employed to enumerate an occlusion front-to-back order and maximize the parallelizable node OQs are explained specifically. Finally, we conducts the importance and contribution culling based on the returned pixel count.

3.1 Input Scene Organization and Maintenance

Hierarchically organizing the input scene makes traversal efficient. A regular grid is used to represent the scene, and each cell in the grid, called voxel, is an axis-aligned box. In some special case, the grid can be treated as an octree [32], and for some other cases, because its dimension is arbitrary, an octree-like hierarchy can be set up. While constructing the hierarchy, efficiently determining the object list for each voxel is a challenging work, especially for a scene with dynamical objects. To reduce the construction time, the size of voxel is set as a multiple of the average size of object's bounding box which

are the majority in a scene so that the voxel can contain several objects. If most of objects are small but a few are large, we divide large objects into a set of small objects because small ones increase the probability of considering them as hidden.

Three solutions [33], splitting objects, distributing objects to the spanning subspace, and finding the smallest node accommodating objects, have been proposed to handle cross-node objects. But, they might not be feasible for dynamic objects, require large memory space when the cross-node object increases, or suffer from low OC effectiveness. To address cross-node objects, we conceptually extend every voxel up to a given constrained size in each axis's positive direction such that each object being located in a node can be fully contained in the extended voxel, called overlapping voxel. The constrained size is the maximum dimension size of bounding box of dynamic objects in majority. Of course, it must be smaller than the size of voxel which is set to contain several objects.

In the dynamic environment, the object list for each voxel must be maintained up-to-date for each frame. To minimize the update time, the scene is initially scaled in a way that each voxel is a unit cube so that the minimum vertex of the bounding box of every object can be used to represent its actual position. The voxel to which an object belongs is efficiently determined by eliminating the decimal parts of the minimum vertex. For instance, let the minimum vertex of an object be $(2.32, 6.1, 3.69)$, then the object is inserted into the voxel indexed by $(2, 6, 3)$. With the overlapping voxel, the dynamic object can be exactly assigned to a voxel only so the amount of memory space required to record object list is fixed. Also, scaling voxel to a unit cube speeds up the object list maintenance for dynamic objects.

3.2 View Frustum Culling

The View frustum culling (VFC) approach culls away objects which are not inside the view frustum. Approaches, [34], [35], [36] and [37], used the intersection test to decide the visibility for nodes and the view frustum. The intersection test is effective and efficient. However, for partial visible nodes the intersection test makes the traversal of hierarchy deeper and results in more occlusion queries invoked while performing OC. In a densely occluded scene, the occlusion possibility of partial visible nodes to the view frustum is high so the partial visible node needs to be checked to see if it can be pruned away before traversing its children nodes. OQ can also be exploited to perform VFC because the hardware clips the primitives with respect to the view frustum before rasterization. In a densely occluded scene, using the occlusion information improves the performance of VFC. However, the cost

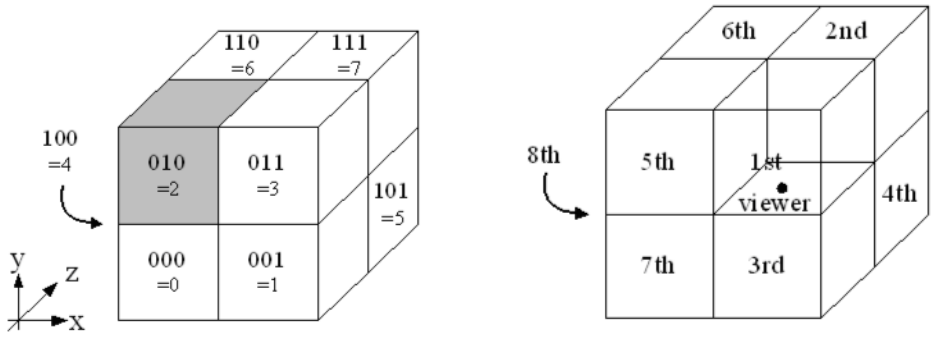
of intersection test is lower than that of OQ, especially for large nodes in the hierarchy. We propose a 2-tier VFC approach by taking both advantages of the intersection test and OQ. While culling nodes against the view frustum, the intersection test is used for larger nodes and OQ applies for smaller. For large nodes, the cost of OQs is high and the occlusion effectiveness is low, so the intersection test is used to cull them against the view frustum. When the hierarchy traversal reaches the smaller nodes, due to the high occlusion possibility, the occlusion queries are invoked for VFC.

Note that while evaluating the occlusion for nodes, the bounding box of each node is the querying primitive for each OQ. This leads to incorrect visibility results for VFC when the bounding box contains the view frustum, the projected faces of the bounding box are too small to draw, and the bounding box intersects the projection plane but the projected faces are occluded. To have the correct result, a near face intersection test is applied. The near face is the rectangle, projection window, of the view frustum on the near plane. The test checks to see if the volume of bounding box intersects the near face. If it does, the node is visible regardless of the result of OQ. This intersection test goes recursively for nodes in the hierarchy, but stops whenever a node does not intersects the near face.

3.3 Occlusion Culling

[**Occlusion Front-to-Back Order Enumeration**] Traversing the hierachy in an occlusion front-to-back order with respect to a given viewpoint benefits the effectiveness of OC. The input scene is organized as an octree-like hierarchy, and it can be regarded as an axis-aligned BSP tree trivially. In a short, the mechanism used to determine the front-to-back order for the BSP tree applies to the input scene. Bernardini [2] determined the front-to-back order by looking up a pre-constructed table using the viewpoint. However, the look-up result might not be further used in maximizing the number of parallelizable OQs in our approach. Instead, we propose the concept of eye-siding number for octants so we can efficiently enumerate them in an occlusion front-to-back order.

Octants in a node are encoded using 3-bit codes first. The bits represent the partition planes, orthogonal to x, y, and z axes, respectively. A bit is set if the octant is in the positive halfspace of the corresponding partition plane, i.e., y-z, x-z or x-y plane. Figure 1(a) shows an example of the encoded octants. Then, we sort the octants into an occlusion front-to-back order, O_0, O_1,..., O_7 , by the eye-siding number. The eye-siding number indicates how many times the node lies at the same halfspace of partition planes with the viewpoint. The 3-eye-siding octant, O_0, containing the viewpoint is the first node. The



(a) (b)

Fig. 1. (a) An encoded octants. The gray octant is in the positive halfspace of y axis but in negative one of both x and z axes. (b) An occlusion front-to-back order for the viewpoint in octant 011.

0-eye-siding octant, O_7, which is not at the same halfspace with the viewpoint for all partition planes is the last node. Three 2-eye-siding octants, O_1, O_2, and O_3, which are at the same halfspace with the viewpoint with respect to two partition planes out of three partition planes are the second order, and the three 1-eye-siding octants, O_4, O_5, and O_6, which locate at the same halfspace for one partition plane out of three partition planes are the third order. The algorithm of an occlusion front-to-back order determination for children octants of a given node is described as follows:

```
// procedure DetermineFront2BackOrder() determines an occlusion front-to-back order for children
// octants of a given node.
// Node: an internal node of the octree, Node.Center: center position of the node
// E: the eye position
// O: the front-to-back order array
// eyeside_i: indicate on which side the eye lies for three axes
// oppside_i: indicate the opposite side to viewer's position for three axes
procedure DetermineFront2BackOrder(Node)
{
    SetBit= 1;
    for i in {x, y, z} {
        eyeside_i = (E_i > Node.Center_i) ? SetBit : 0 ;
        oppside_i = (E_i > Node.Center_i) ? 0 : SetBit;
        SetBit = SetBit << 1; // <<: shift left bitwise operator
    }// end of for i in x, y, z
    O_0 = eyeside_x|eyeside_y|eyeside_z; // |: bitwise OR operation
    O_1 = eyeside_x|eyeside_y|oppside_z;
    O_2 = eyeside_x|oppside_y|eyeside_z;
    O_3 = eyeside_x|oppside_y|oppside_z;
    O_4 = oppside_x|eyeside_y|eyeside_z;
    O_5 = oppside_x|eyeside_y|oppside_z;
    O_6 = oppside_x|oppside_y|eyeside_z;
    O_7 = oppside_x|oppside_y|oppside_z;
    return O;
}// end of DetermineFront2BackOrder(Node)
```

Figure 1(b) demonstrates an occlusion front-to-back order for the viewpoint in octant 011. Let *Node.Center* be (0, 0, 0), according to our method *eyeside* and *oppside* vectors are (1, 2, 0) and (0, 0, 4) respectively. Therefore, an occlusion front-to-back order sequence is $O_0 = 1|2|0 = 011 = 3$, $O_1 = 1|2|4 = 111 = 7$, O_2

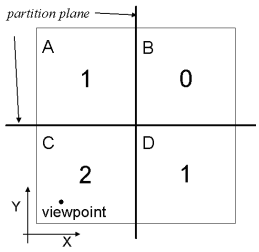


Fig. 2. The eye-siding number for four children in a quad-tree node.

= 1|0|0= 001= 1, O_3 = 1|0|4= 101= 5, O_4 = 0|2|0= 010= 2, O_5 = 0|2|4= 110= 6, O_6 = 0|0|0= 000= 0, and O_7 = 0|0|4= 100= 4. Note that the order of selection of partition plane will influence the sequence of occlusion front-to-back order, but the actual occlusion orders are equivalent. Also, it is easy to adapt this algorithm to the hierarchies of voxels with arbitrary dimensions, not limited to the dimension with power of 2.

[**Maximum the Number of Parallelizable Node Occlusion Queries**] Using OQ to estimate the visibility, the parallelism must be exploited to reduce the stalling. To maximize the number of parallelizable OQs while traversing the hierarchy, the nodes in the same level are divided into parallel units such that the bounding boxes of all nodes in a unit can be sent for occlusion evaluation at a time. The eye-siding number encoded for octants of a node can be further exploited in the exploration of parallelizable nodes. The nodes with the same eye-siding number are parallelizable because the rendering order doesn't affect the occlusion result. Hence the children octants of a node can be classified into parallel units by their eye-siding number. There are four parallel units for octants in a node. The 3-eye-siding unit includes only one octant in which the viewpoint lies. The 0-eye-siding unit includes only one octant that is at the opposite side of the 3-eye-siding octant. The 2-eye-siding unit has three octants which locate at the eye side of two partition planes out of three partition planes. The 1-eye-siding unit also has three octants which locate at the eye side of one partition plane out of three partition planes. Figure 2 shows a quad-tree case where node C is 2-eye-siding because it contains the viewpoint and both node A and D are 1-eye-siding since they are at the eye side of x and y partition planes respectively. Node B is not at the eye side for two partition planes, so it is 0-eye-siding. Hence, there are three parallel units; $\{C\}$, $\{A, D\}$, and $\{B\}$. For each parallel unit the eye-siding numbers of its descendent are determined by their corresponding partition planes respectively, and all children in a level can be classified into parallel units. For example, as shown in Figure 3, the children nodes of the parallel unit $\{A, D\}$ are divided into $\{A_3, D_3\}$, $\{A_1, A_4, D_1, D_4\}$, and $\{A_2, D_2\}$ parallel units according to their eye-siding number.

While traversing the hierarchy, the parallel units are examined one-by-one recursively rather than evaluating the nodes. The parallel unit on the top of

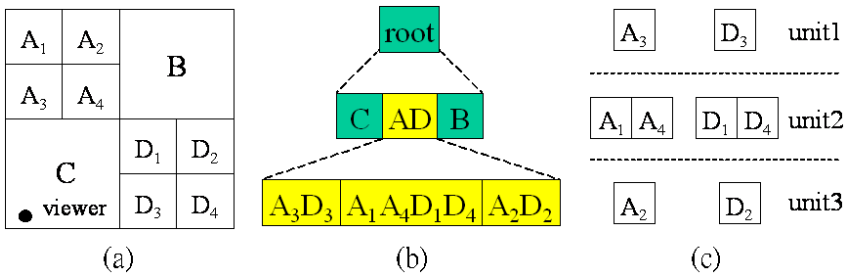


Fig. 3. A 2D example of the parallel unit hierarchy. (a) A node hierarchy where nodes A and D are partitioned into four subnodes respectively. (b) The children of the root node are grouped into three parallel units by the eye-siding number, $\{C\}$, $\{A, D\}$, and $\{B\}$. Similarly, the parallel unit $\{A, D\}$ can be further divided into three parallel units: $\{A_3, D_3\}$, $\{A_1, A_4, D_1, D_4\}$, and $\{A_2, D_2\}$. (c) An occlusion front-to-back order of parallel unit $\{A, D\}$.

the scene graph contains the root node only. For each visited unit, the visibility of its nodes is evaluated using parallel OQs, and the hidden ones are excluded for further processing. The visible children nodes are classified into the four eye-siding sets and inserted to the corresponding parallel unit, P_i, i= 3, 2, 1, 0. These four units are recursively traversed in depth first order until the leave node is reached. Eventually, the number of parallel OQs is not unlimited due to the hardware implementation. If the number exceeds the upper bound, then the queries are divided into multiple passes. The algorithm of multi-pass parallel OQ is described as follows:

```
// procedure MultipassParallelOcclusionQuery() performs occlusion queries for parallelizable nodes
// in pNodes in multiple passes
// pNodes: the set of parallelizable nodes
// nNodes: a subset of pNodes, it's maximum cardinality is no more N
// Q: the set of pre-generated identifiers of occlusion queries
// N: the maximum number of parallel occlusion queries, depending on hardware
// V: the set of visible nodes
procedure MultipassParallelOcclusionQuery(pNodes)
{
    V = φ;
    While (pNodes != φ) {
        Get nNodes={Node_1, Node_2, ..., Node_N} out of pNodes
        // switching hardware setting
        UpdateColorBuffer(DISABLE);
        UpdateDepthBuffer(DISABLE);
        // sent n(<= N) nodes for OQ at a time
        for i = 1 to Cardinality(nNodes) {
          // Q_i is the OQ identifier for Node_i
          BeginOcclusionQuery(Q_i);
          Draw(nNodes.Node_i);
          EndOcclusionQuery(Q_i);
        }
        UpdateColorBuffer(ENABLE);
        UpdateDepthBuffer(ENABLE);
        // get the pixel count of occlusion evaluation back
        for i = 1 to Cardinality(nNodes)
          V = V∪(RequestPixelCount(Q_i) > 0 ? Node_i : φ)
        pNodes = pNodes - nNodes;
    }// end of while (pNodes != φ )
    return V;
}// end of MultipassParallelOcclusionQuery(pNodes)
```

Furthermore, we must keep the occlusion front-to-back order while OQs are requested for parallel units. The order for children of a node is determined using procedure *DetermineFront2BackOrder()*. The returned array O contains an occlusion front-to-back order of the children nodes, and elements in array O reveal the eye-siding order. Namely, the parallel unit traversal sequence, $P_3 = \{O_0\}, P_2 = \{O_1, O_2, O_3\}, P_1 = \{O_4, O_5, O_6\}, P_0 = \{O_7\}$, is in an occlusion front-to-back order. Figure 3 shows an example of the hierarchy of parallel units and their occlusion front-to-back order. We summarize the traversal scheme as follow:

```
// procedure ParallelUnitOcclusionQueryTraversal() traverses the hierarchy by parallel unit
// and evaluates the occlusion in an occlusion front-to-back order.
// P: the parallel unit, P={P_i|i = 1,2,3,4}, P_i: i-eye-siding parallel unit.
// B: the set of bounding volumes of the nodes in.
// V: the set of visible nodes
// O: the front-to-back order array
procedure ParallelUnitOcclusionQueryTraversal(P)
{
  P_3 = P_2 = P_1 = P_0 = φ;
  // performing view frustum culling
  V = φ;
  for  i = 1 to cardinality(P.Nodes)
    V = V∪ 2tierVFC(Node_i);
  if ((V = MultipassParallelOcclusionQuery(V)) = φ ) return;
  while (Node_i in V) {
     if ( Node_i = LeafNode ) {
        draw( Node_i );
        return;
     }
     // determine the front-to-back order for children of Node_i
     O = DetermineFront2BackOrder(Node_i);
     // insert children of Node_i with eye-siding number i into parallel unit P_i
     P_3 = P_3 ∪ {Node_i.Child_O[0]};
     P_2 = P_2 ∪ {Node_i.Child_O[1], Node_i.Child_O[2], Node_i.Child_O[3]};
     P_1 = P_1 ∪ {Node_i.Child_O[4], Node_i.Child_O[5], Node_i.Child_O[6]};
     P_0 = P_0 ∪ {Node_i.Child_O[7]};
  }// endof while (Nodei in V)
  ParallelUnitOcclusionQueryTraversal(P_3);
  ParallelUnitOcclusionQueryTraversal(P_2);
  ParallelUnitOcclusionQueryTraversal(P_1);
  ParallelUnitOcclusionQueryTraversal(P_0);
  return;
}// end of ParallelUnitOcclusionQueryTraversal(P)
```

3.4 Contribution Culling and Importance Culling

The OQ returns the number of visible pixels. This pixel count can be used to measure the contribution of rasterized objects at screen space, whenever applications accept the approximate result. The mechanism of contribution culling is trivially that a node or an object is to be culled away if the returned pixel count of its OQ is no more than a given threshold. Carefully adjusting the threshold value gains the performance without loss of visual quality in some scenes.

In addition to applying OQ to contribution culling, the returned pixel count

can also be used to facilitate the importance culling. The importance culling is based on the observation that the human eye often focuses on a portion of the screen. The importance for different regions of the screen space can be specified by an importance mask. The stencil buffer is overridden using a manually predefined or procedurally generated importance mask, and the stencil test is enabled while evaluating the occlusion. While performing occlusion evaluation using OQs, the number of returned pixels means the number of visible pixels in the importance region, i.e. not masked off. In other words, zero pixel returned means that the object is actually occluded or visible but located in the non-importance region. Apparently, using importance culling causes more objects to be culled away so that the complex scene can be rendered faster. Moreover, we can combine the importance culling with the contribution culling to obtain even more efficient rendering performance.

4 Experimental Results

We have implemented the proposed approaches and carefully tested them on a PC, running Windows XP, with one CPU, Intel Pentium 4 2.4G, and 1GB main memory. The graphic card is based on the chipset of GeForce 4 Ti 4400.

[**2-tier View Frustum Culling**] Experiments have conducted to compare the average number of invoked OQs for VFC using intersection test and using OQ only and the proposed 2-tier VFC with depth 2, 3, and 4 at 8x8x8, 16x16x16, 32x32x32, and 64x64x64 grid resolutions respectively. As Figure 4 shows, the number of OQs invoked by VFC approach using intersection test is much higher. The intersection test often makes the hierarchy traversal deeper because all partial visible nodes perform the test for their children nodes. As for the 2-tier VFC approach, the intersection test only applies for larger nodes and OQ for the rest of nodes (smaller) in the hierarchy. In a high occlusion scene, a node that is partial visible in the view frustum might be occluded. As you can see in Figure 4, it is clearly that the number of invoked OQs in the proposed 2-tier VFC is reduced. The timing of performing intersection test is less than that of OQ, especially for large nodes in the hierarchy. To our experiments, the performance of 2-tier VFC with threshold depth 3 is the best.

[**Contribution and Importance Culling**] While performing contribution culling, a querying node in the hierarchy is considered as visible if the returned pixel count exceeds a given upper bound rather than zero. Importance culling is implemented by loading a mask of the specified important region to the stencil buffer and enabling the stencil test while occlusion evaluation. A node which is visible means that it can be seen in the specified important region. Figure 5 shows two sample masks, types I and II, of important regions (white

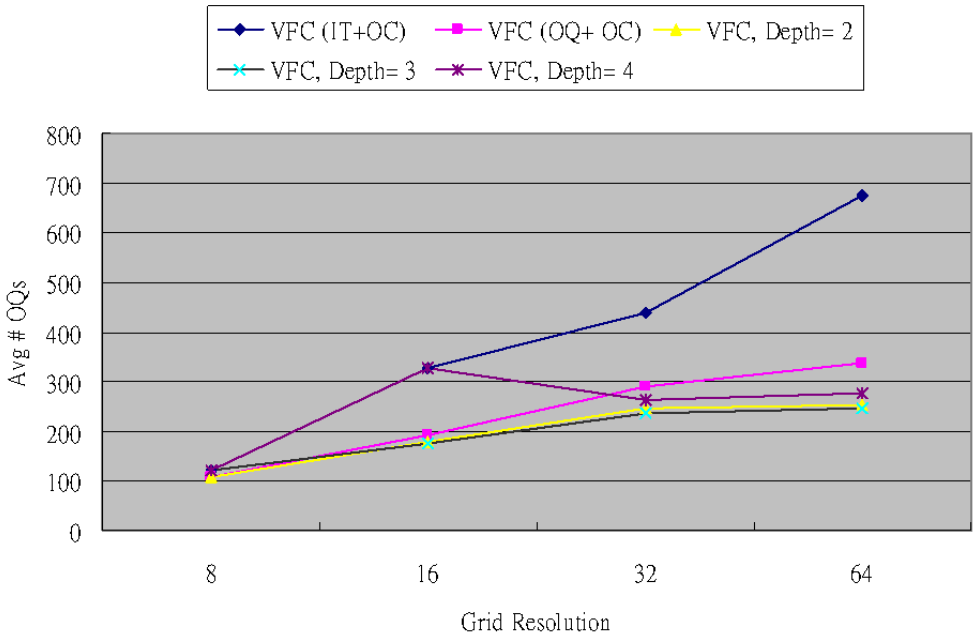


Fig. 4. Statistics of average number of OQs for VFC using intersection test and using OQ only and 2-tier VFC with depth 2, 3, and 4 at 8x8x8, 16x16x16, 32x32x32, and 64x64x64 grid resolutions respectively.

area) used in the test. Each frame (black area) in a mask is associated with a fill ratio to indicate the mask-off density of the frame, namely the higher fill ratio masks off more pixels projected in the frame. Note that the shape of mask doesn't need to be rectangle. One can design any shape they would prefer.

We made a comparison of average frame rate as shown in Figure 6. As you can see, the higher pixel upper bound of contribution culling and the smaller importance region with higher fill ratio achieve faster frame rate. Note that in place of using an exact pixel count for contribution culling the pixel ratio (CC Pixel ratio) is used. Multiplying this ratio by the screen resolution determines the pixel count upper bound. The proposed contribution and importance culling approaches using OQ are not conservative. We measure the errors using the ratio of the number of error pixels to the pixel amount of a screen. Of course, as shown in Figure 7, smaller important regions with lower fill ratios gain more efficiency whereas result in higher error ratio.

[**Overall Test**] To show the overall performance of our approach for a complex and interactive walkthrough environment, we constructed a scene consisting of one million objects, totally $778,497,348$ polygons. Half of the objects are dynamic and the others are static. Static objects include torus knot (1920 polygons), hose (920 polygons), and hollow box (135 polygons), and dynamic objects include teapot (1024 polygons), torus (576 polygons), star (96 polygons). Note that the initial position and velocity of dynamic objects are gener-

ated randomly. The collision detection is only performed for dynamic objects against the scene boundary to prevent objects from moving away. If an object collides the boundary, a new reverse velocity is randomly assigned. As for the grid, 32x32x32 voxels are used to represent the scene, and the grid is scaled such that all voxels in the grid are unit cubes for speeding up the actual position update for dynamic objects.

Figure 8 shows the statistics of frame rates of the proposed parallel OQ with 2-tier VFC, non-parallel OQ with 2-tier VFC, Hillesland's method [29], and hardware Z-buffer. All the scenes are rendered by Gouraud shading with one directional light at screen resolution 1024X768 pixels. A snapshot of the test walkthrough environment is shown in Figure 10. We implemented the nested grid decomposition version of Hillesland's method to compare with our approach. For all frames of the walkthrough, it shows that the performance of using parallel OQ with 2-tier VFC is the best. On average, we have 17.96 fps for parallel OQ with 2-tier VFC, 15.48 fps for non-parallel OQ with 2-tier VFC, 16.79 fps for Hillesland's method, and 0.95 fps for z-buffer. Namely, we have 13.8% speed-ups of parallel OQ with 2-tier VFC over the non-parallel OQ with 2-tier VFC and 7.8% speed-ups of parallel OQ with 2-tier VFC over Hillesland's method.

Also, we tested the performance of proposed approaches using parallel OQ with 2-tier VFC combined with importance and/or contribution culling techniques. Figure 9 shows the statistics of frame rate for the parallel OQ with 2-tier VFC approach, the approach combined with importance culling using type II importance region with fill ratio 0.6 (II(0.6)), and the approach combined with II(0.6) and contribution culling with pixel upper bound ratio 0.001 (CC(0.001)), in the same walkthrough as in Figure 8. On average, we have 17.58 fps for parallel OQ with 2-tier VFC, 18.38 fps (4.58% speed-ups) for parallel OQ with 2-tier VFC com-bined with importance culling, and 22.05 fps (21.03% speed-ups) for parallel OQ with 2-tier VFC combined with importance and contribution culling. Although non-conservative approaches introduce error, however, as you can see in Figure 11, the image quality is visually acceptable if we carefullly control the contribution culling pixel ratio and depict the importance mask. So, sacrificing visibility correctness for the overall performance is feasible in our approach.

5 Conclusion

We have presented a visibility pipeline, VFC, OC, IC and CC, using hardware-accelerated OQs. The regular grid is used to organize the spatial data for efficiently updating the actual position of objects in dynamics so the object list in each node keeps consistent for each frame. A 2-tier mechanism with

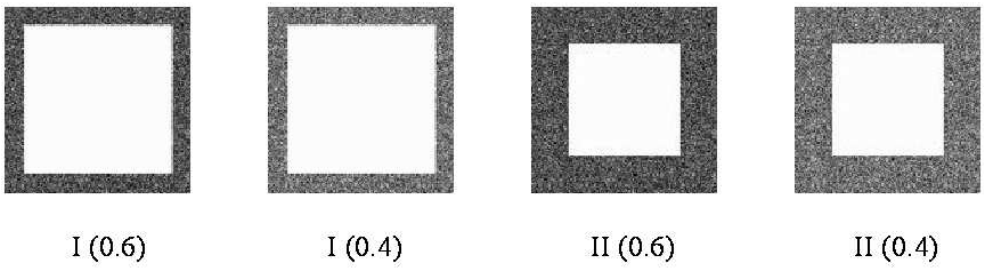


Fig. 5. Two types of important regions with fill ratios, 0.6 and 0.4 respectively.

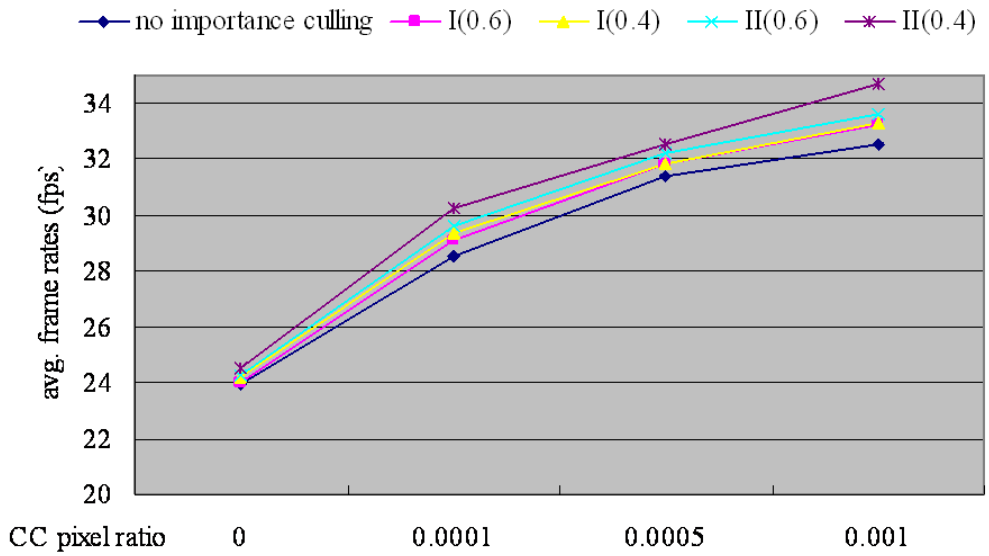


Fig. 6. The average frame rates comparison for different parameters values in the importance and contribution culling approaches.

advantages of intersection test and OQ is used to efficiently perform VFC. The grid is easy to be represented as an octree-like hierarchy for hierarchical traversal. The encoded eye-siding number for nodes in the hierarchy can not only be used to effectively and efficiently enumerate the node's occlusion front-to-back order, but also nodes with the same eye-siding number in a front-to-back order sequence can be grouped into a parallel unit and sent for occlusion evaluation at a time. Pre-loading a specified importance mask to the stencil buffer and enabling stencil test while performing occlusion evaluation, we utilize the OQ for implementing importance culling and the contribution culling by given a pixel ratio. If objects pass the OQ but not projected in the defined importance region and/or the contributed pixel count is less than the pixel count upper bound, then they are culled away. As experiments show, the proposed 2-tier VFC method is more efficient, maximizing the number of parallelizable OQs makes the utilization of hardware-accelerated mechanism even better and leads to a better overall performance, and together with the importance culling and contribution culling produces acceptable image quality but significantly improves the rendering performance.

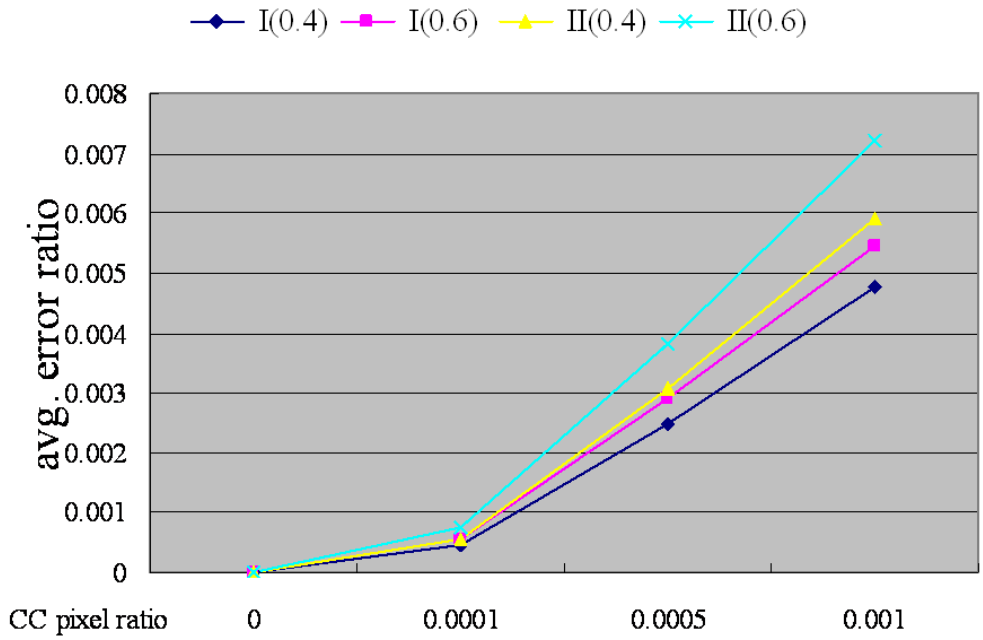


Fig. 7. A comparison of the average error ratios of contribution and importance culling approaches for different parameters values.

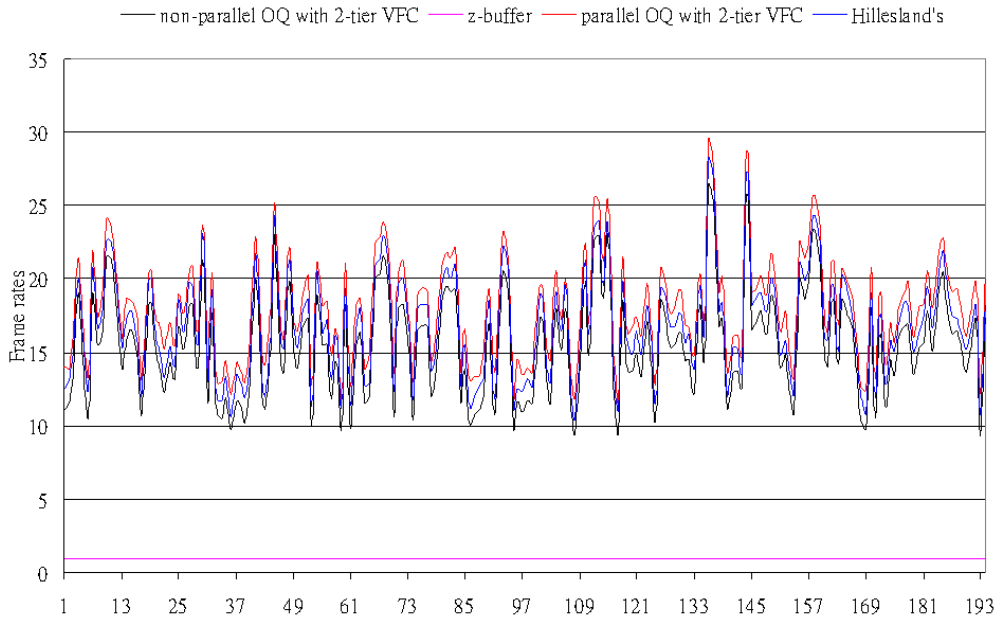


Fig. 8. The statistics of frame rates in the input walkthrough environment for the proposed approach using parallel and non-parallel OQ with 2-tier VFC, Hillland's method and Z-buffer.

In this approach objects inside a visible voxel are all sent for rendering. When the number of objects is large, they could be sorted into an approximate front-to-back order for OQ provided that the rendering time of objects is much higher than that of OQ. In the future, a mechanism to see if OQ is worthy of applying for objects in a visible node is to be explored. The grid

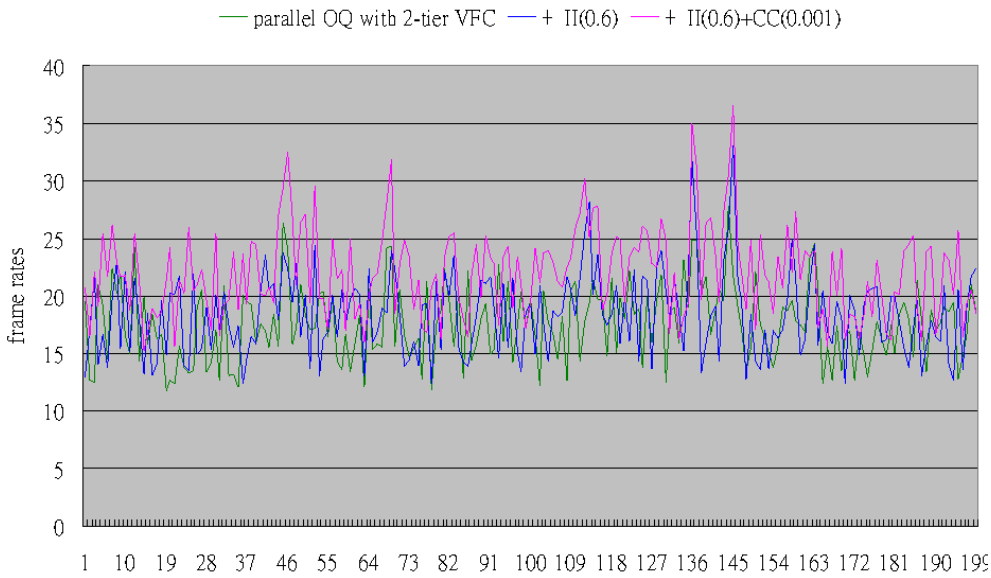


Fig. 9. The statistics of frame rates in an complex and interactive walkthrough environment as Figure 8 for the proposed approach using parallel OQ with 2-tier VFC, the approach combined with importance culling with II(0.6), and the approach combined with with II(0.6) and contribution culling with CC(0.001).

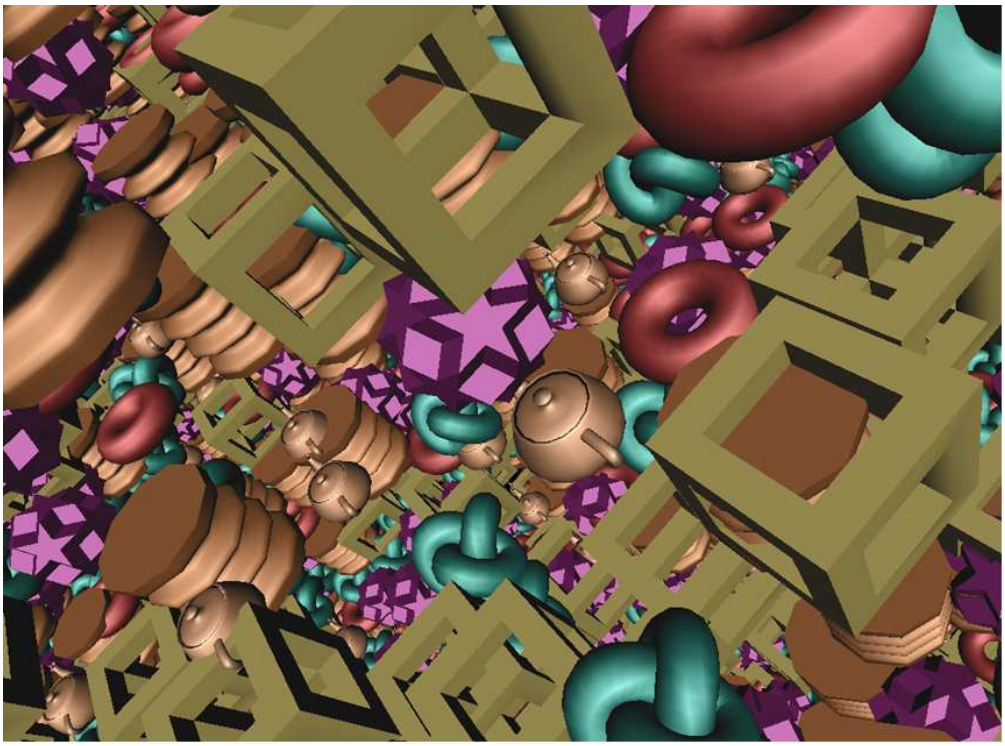


Fig. 10. A snapshots of the input scene used for the overall performance test.

resolution is empirically selected, not depending upon the number and size of objects. Moreover, with experiments we have made, the hierarchy traversal time, the time for performing node OQ, and the actual position update

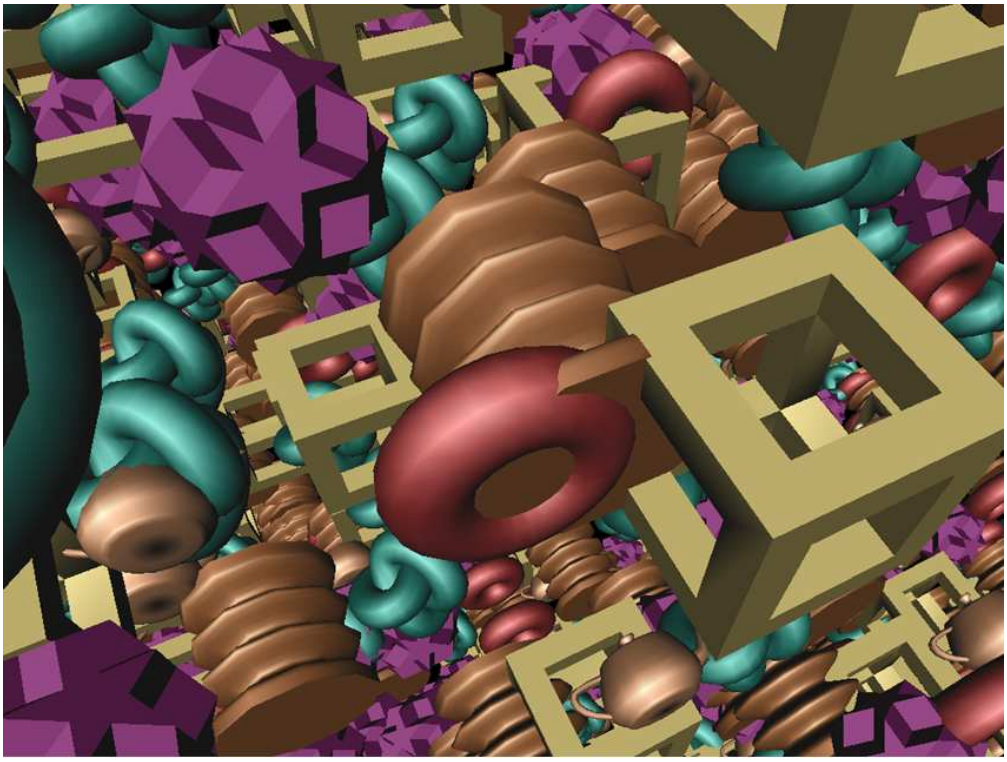


Fig. 11. A snapshots of the input scene rendered by using the visibility pipeline composed of 2-tier VFC, OC with parallel OQ, importance culling with II(0.6) and contribution culling with CC(0.001).

time for dynamic objects are relevant to the grid resolution. An automatic grid resolution determination scheme should be studied for improving or even optimizing the overall performance.

References

[1] D. Cohen-Or, Y.L. Chrysanthou, C.T. Silva, and F. Durand, "Survey of visibility for walkthrough applications," EEE Tran. on Vis. and Comp. Graph., vol.9, no.3, 2003.

[2] F. Bernardini, J. El-Sana, and J.T. Klosowskix, "Directional discretized occluders for accelerated occlusion culling," Proc. of EUROGRAPHICS 2000, vol.19, no.3, p.35, Aug. 2000.

[3] J. Bittner, V. Havran, and P. Slavík, "Hierarchical visibility culling with occlusion trees," Proc. of Comp. Graph. International, pp.207–219, June 1998.

[4] F. Durand, G. Drettakis, J. Thollot, and C. Puech, "Conservative visibility preprocessing using extended projections," SIGGRAPH, pp.239–248, July 2000.

[5] J. Heo, J. Kim, and K. Wohn, "Conservative visibility preprocessing for walkthroughs of complex urban scenes," ACM Symp. on VRST, 2000.

[6] J. Kim and K. Wohn, "Conservative visibility preprocessing for complex virtual environments," VSMM, 2001.

[7] V. Koltun, Y. Chrysanthou, and D. Cohen-Or, "Hardware-accelerated from-region visibility using a dual ray space," Rendering Techniques 2001: 12th Eurographics Workshop on Rendering, pp.205–216, June 2001.

[8] O.E.M. Pastor, "Visibility preprocessing using spherical sampling of polygonal patches," Eurographics 2002, Short / Poster Presentations, 2002.

[9] G. Schaufler, J. Dorsey, X. Decoret, and F.X. Sillion, "Conservative volumetric visibility with occluder fusion," SIGGRAPH, pp.229–238, July 2000.

[10] P. Wonka, M. Wimmer, and D. Schmalstieg, "Visibility preprocessing with occluder fusion for urban walkthroughs," Rendering Techniques 2000: 11th Eurographics Workshop on Rendering, pp.71–82, 2000.

[11] D. Bartz, M. Meibner, and T. Huttner, "Opengl assisted occlusion culling for large polygonal models," Comp. & Graph., vol.23, no.3, pp.667–679, 1999.

[12] J.T. Klosowski and C.T. Silva, "The prioritized-layered projection algorithm for visible set estimation," IEEE Trans. on Vis. and Comp. Graph., vol.6, no.2, pp.108–123, 2000.

[13] L. Shou, Z. Huang, and K. Tan, "Performance guaranteed rendering using the HDoV tree," GRAPHITE, 2003.

[14] D. Cohen-Or and A. Shaked, "Visibility and dead-zones in digital terrain maps," Comp. Graph. Forum, vol.14, no.3, pp.171–180, 1995.

[15] V. Koltun, Y. Chrysanthou, and D. Cohen-Or, "Virtual occluders: an efficient intermediate pvs representation," Eurographics Workshop on Rendering, pp.59–70, June 2000.

[16] D. Luebke and C. Georges, "Portals and mirrors: simple, fast evaluation of potentially visible sets," Proc. SIGGRAPH Symp. on Interactive 3D Graphics, pp.105–106, 1995.

[17] S. Coorg and S. Teller, "Real-time occlusion culling for models with large occluders," Symp. on Interactive 3D Graph., pp.83–90, Apr. 1997.

[18] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang, "Accelerated occlusion culling using shadow frustra," Proc. ACM Symp. Comp. Geom., pp.1–10, 1997.

[19] N. Greene, M. Kass, and G. Miller, "Hierarchical z-buffer visibility," SIGGRAPH, pp.231–240, 1993.

[20] H. Zhang, D. Manocha, T. Hudson, and K.E.H. III, "Visibility culling using hierarchical occlusion maps," SIGGRAPH, pp.77–88, Aug. 1997.

[21] D. Bartz, J. Klosowski, and D. Staneker, "K-dops as tighter bounding volumes for better occlusion performance," SIGGRAPH Visual Proc. 2001, 2001.

[22] P. Wonka and D. Schmalstieg, "Occluder shadows for fast wakthroughs of urban environments," Comp. Graph. Forum, vol.18, pp.51–60, 1999.

[23] H.C. Batagelo and S. Wu, "Dynamic scene occlusion culling using a regular grid," Proc. of the XV Brazilian Symp. on Comp. Graph. and Image Processing, pp.43–50, 2002.

[24] O. Sudarsky and C. Gotsman, "Dynamic scene occlusion culling," IEEE Trans. on Vis. and Comp. Graph., vol.5, no.1, pp.13–29, 1999.

[25] P.C. Ho and W. Wang, "Occlusion culling using minimum occluder set and opacity map," International IEEE Conf. on Info. Vis., p.292, 1999.

[26] J.T. Klosowski and C.T. Silva, "Efficient conservative visibility culling using the prioritized-layered projection algorithm," IEEE Trans. on Vis. and Comp. Graph., vol.7, no.4, pp.265–379, 2001.

[27] N. Greene, "Occlusion culling with optimized hierarchical z-buffering," ACM SIGGRAPH Course Notes on visibility # 30, 2001.

[28] M. Meissner, D. Bartz, T. Huttner, G. Muller, and J. Einighammer, "Generation of subdivision hierarchies for efficient occlusion culling of large polygonal models," Comp. & Graph. 2002, 2002.

[29] K. Hillesland, B. Salomon, A. Lastra, and D. Manocha, "Fast and simple occlusion culling using hardware-based depth queries," Techinical report TR02-039, 2002.

[30] N.K. Govindaraju, A. Sud, S. Yoon, and D. Manocha, "Interactive visibility culling for complex environments using occlusion-switches," Symp. on Interactive 3D Graphics, Apr. 2003.

[31] D. Staneker, D. Bartz, and M. Meibner, "Improving occlusion query efficiency with occupancy maps," IEEE Symposium on Parallel and Large-Data Visualization and Graphics, 2003.

[32] H. Samet, Applications of Sspatial data structures: computer graphics, image processing and GIS, Addison-Wesley, Reading, Massachusetts, 1989.

[33] T. Akenine-Möller and E. Haines, Real-time rendering, 2nd, A K Peters.

[34] U. Assarsson and T. Moller, "Optimized view frustum culling algorithms for bounding boxes," JGT, vol.5, no.1, pp.9–22, 2000.

[35] L. Bishop, D. Eberly, T. Whitted, M. Finch, and M. Shantz, "Designing a pc game engine," Comp. Graph. in Entertainment, pp.46–53, Jan. 1998.

[36] M. Slater and Y. Chrysanthou, "View volume culling using a probabilistic caching scheme," ACM VRST Lausanne Switzerland, 1997.

[37] K. Hoff, "Fast aabb/view-frustum overlap test," 1997.