LOD Visibility Culling and Occluder Synthesis

Carlos Andújar Carlos Saona-Vázquez Isabel Navazo

Dept. LSI. Universitat Politècnica de Catalunya
Diagonal 647. E-08028 Barcelona, Spain
{andujar, carloss, isabel}@lsi.upc.es

July 13, 2000

Abstract

Level-of-detail occlusion culling is a novel approach to the management of occluders that can be easily integrated into most current visibility culling algorithms. The main contribution of this paper is an algorithm that automatically generates sets of densely overlapping boxes with enhanced occlusion properties from non-convex subsets. We call this method occluder synthesis because it is not sensitive to the way the objects are tesselated but to the space enclosed by them. The extension of this technique by allowing a bounded amount of image error is also discussed. We show that visibility computations can be based on a multiresolution model which provides several representations of these occluders with varying visibility accuracy. Our tests show that occlusion performance in tesselated scenes is improved severely even if no image-error is allowed.

1 Introduction

Real-time inspection of very large models, with hundreds of thousands of faces, often surpasses the hardware performance of current high-end workstations. This has led to substantial research into devising complimentary software-based techniques for image acceleration, including level-of-detail rendering, visibility culling, texturing, compression and adaptive processing.

Level-of-detail rendering (LOD-rendering for short) refers to the possibility of rendering objects that cover a small portion of the screen using a simplified version of them instead of the original representation. Accurate representations are reserved for close, large or important objects [11].

Visibility culling deals with the identification of those portions of the scene potentially visible from a dynamic viewpoint. At least two sufficient conditions for invisibility can be identified. *View frustum culling* discards the parts of the scene that are outside the field of view. *Occlusion culling* keeps the graphics hardware from drawing the parts that are occluded by front-end objects. The simplest form of occlusion culling is backface culling, which discards those polygons whose normal is facing away from the viewer.

All visibility culling algorithms have computationally intensive pre-processing stages. Preprocessing typically includes the computation of some kind of hierarchical data structure to store the scene and an occluder selection step [8, 12, 7, 20]. Only objects obscured by a single occluder will be identified as not visible. The moment when this identification takes place separates purely preprocessing algorithms [20, 15, 7] from those that perform non-trivial computations while navigating [8, 12]. Algorithms in the former family subdivide the navigational space into cells of constant visibility. As part of the pre-process, the algorithm computes the potentially visible set (PVS for short) for each cell. Algorithms in the latter family compute the PVS at navigation-time and re-compute it each time the viewpoint changes. In either case, elements in the PVS are those objects of the scene that were not obscured by any of the selected occluders when viewed from the current cell or viewpoint. The overestimation of the PVS is sensitive

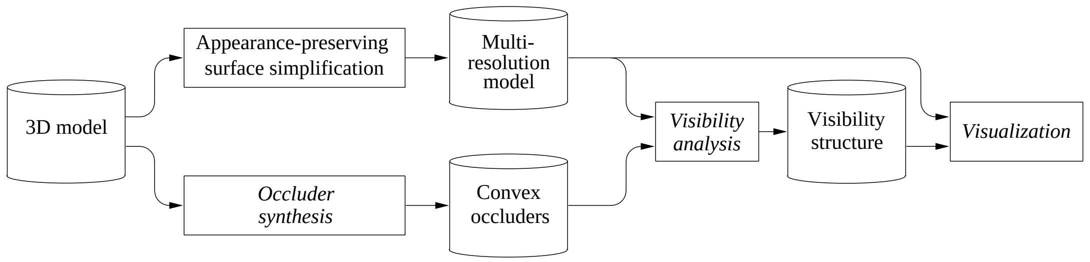


Figure 1: Overview of LOD-rendering (*top*) and LOD-occlusion (*bottom*) preprocessing.

to the selected occluder set.

Almost all occlusion culling algorithms require the occluders to be convex for both efficiency (faster algorithms are known for some geometric problems when the input is a convex polytope) and simplicity (the visibility inside a polyhedral volume can be easily determined from the visibility at its vertices if the occluders are convex [18, 20]). In fact, current algorithms consider an object as invisible only if it is obscured completely by a single occluder. An object with one half obscured by occluder C_i and the other half occluded by occluder C_j will not be classified as invisible because the involved computation is comparable with that of working with concave occluders (recall that the union of convex objects is not necessarily convex).

Moreover, current occlusion culling algorithms compute the occluder set by using *raw occluder selection*. They traverse the scene looking for convex faces or polyhedra. Each time a convex is found, its potential occlusion degree is estimated [8] and if it is greater than a given threshold value, the element (face or polyhedra) is selected as an occluder. Since there is no scene preprocessing prior to the occluder selection, each occluder corresponds to an actual face of the scene. Therefore raw occluder selection is much more sensitive to the boundary of the objects and the way they are tesselated (e.g. as a triangle mesh), than to the underlying pointsets. Even if a basic face merge preprocessing is carried out before raw occluder selection, the occluders remain on the boundary faces of the scene geometry and objects with small details and bevels on their surfaces make impossible to get big convex polygons from them just by merging coplanar faces.

For all the above reasons, current occlusion culling algorithms work with occluders much smaller than they could actually be and often need human intervention for finding effective occluders.

An open problem addressed in this paper is occluder synthesis. Given an arbitrary scene involving concave and highly tesselated objects, our proposal is to compute a minor set of possibly-overlapping new convex objects with enhanced occlusion properties that will act as occluders during the PVS computation. The computation of synthesized occluders has two main steps: aggregation, which builds a new discrete representation of the scene from which both exact-visibility and bounded-error occluders can be extracted, and convex extraction, which computes a densely-overlapping set of boxes from the previous representation.

Another consideration to be taken into account in occluder synthesis is that in a highly occluded scene (as urban and ship environments), there are usually narrow holes among objects. Classical visibility culling algorithms produce a potential visible set that also includes objects that are visible only through these holes and which have not significant contribution on the final image. The second contribution of this paper is a multiresolution version of our occluder synthesis procedure that produces a simplification of the scene by connecting disconnected shells. Obviously, this approach could produce visibility errors on the image but as we show they are almost unnoticeable.

Figure 1 shows an overview of our proposed modification of the classical LOD rendering pipeline including occluder synthesis. The visibility analysis computation can be done in a pre-process step for each cell in which the scene is subdivided, or at navigation-time. During this visibility analysis step, and according to the viewpoint and the error tolerance, occluders of the correct LOD resolution are selected to compute visibility.

Some highlights of our proposal are:

- It always extracts convex parts of underlying polyhedra, even if the input model does not provide polyhedral information (as in the case of polygon soups).

- It can generate many overlapping convex occluders from one single object.

- It can take the whole scene as its input data and deals with polygon soups as effectively as with highly structured scenes.

- The output of the algorithm is restricted to boxes, which have a very compact representation and whose shadow frustra can be trivially computed in O(1) time. Furthermore, 2D convex polygons can be extracted from these boxes in a straightforward manner, if required for visibility computations.

The rest of the paper is structured as follows. Section 2 discusses previous research on occluder synthesis. Section 3 details our contribution on loss-less occluder synthesis based on the construction of an intermediate volumetric model of the scene and on the extraction of overlapping boxes. The novel concept of LOD occlusion culling is presented in Section 4 jointly with the analysis of image error. Finally, Sections 5 and 6 show some results and summarize some conclusions and future work.

2 Previous work

Occlusion culling literature has focused mainly on the type of spatial data structures and on the method of finding whether a volume is obscured (there is a survey on this field elsewhere [21]). Though current visibility culling algorithms are known to be very sensitive to the boundary representation of the scene, very little effort has been devoted to lessen this shortcoming.

Recently, Law and Tan [15] proposed a new framework that integrated simplification and occlusion culling techniques. In their proposal, occluders are synthesized in three steps. First, a potential object is selected from the scene and its geometry is decimated using current simplification algorithms. In the second step, and in order to preserve occlusion properties, those vertices of the decimated object that lie outside the original one are translated till bounded by it. Finally, edges in the decimated model are also perturbed so to ensure convexity. Their tests showed the algorithm to be able to generate synthesized convex occluders for significant frame-ratio improvements.

However, Law and Tan's algorithm has some drawbacks. As noted by the authors, validity of the resulting occluder is not always ensured by the occlusion synthesis procedure as the perturbation stages can yield null objects with no occlusion power. Furthermore, the algorithm outputs one single occluder for each input and it is sensitive to the tessellation of the input objects.

Klosowski and Silva [14] presented a different but also lossy approach that does not compute PVS explicitly. Instead, it is based on computing a priority order for the polygons that maximizes the likelihood of rendering visible polygons before occluded ones. The user sets a bound on the number of polygons to render in each frame, and the algorithm renders those with the maximum visibility likelihood. As a result of the polygon bound, it is usual to obtain errors in the final image due to the absence of visible objects and the presence of obscured ones.

3 Occluder synthesis

In this section a new occluder synthesis algorithm that enhances significantly the occlusion power of most current visibility culling algorithms is presented. It generates sets of large overlapping boxes from non-convex polyhedra with an arbitrary number of shells. The remainder of this section is organized as follows. First, we state the occluder synthesis problem in terms of the properties of the occluders. Next we survey some previous work related to convex object extraction, and finally the aggregation and convex extraction stages are detailed.

3.1 Problem statement

Let P be an arbitrarily complex scene. A set $B = \{B_i\}$ of convex synthesized occluders must be generated from P satisfying these conditions:

1. Each B_i must be either a 2D convex polygon or a 3D solid with a convex silhouette from any

viewpoint. In both cases the shadow casted by B_i is convex. Since the shadow casted by a polyhedron is greater than the individual shadows of its faces, and for an object to be identified as not visible it must be completely obscured by a single occluder, 3D occluders are preferred over 2D ones whenever they are supported by the occlusion analysis strategy adopted.

2. As the number of occluders affects directly the speed of the visibility analysis, the cardinality n of B must be kept relatively small. This is specially important if the visibility analysis is not a preprocess but occurs on-line [8, 12].

3. The volume of each individual convex B_i should be maximized. This is due to the fact that, as we pointed out in the introduction, for an object to be identified as not visible it must be completely obscured by a single occluder.

4. In order to preserve visibility properties, each B_i must be completely contained in P.

5. The contribution of each B_i must be significant, i.e. a large part of B_i should not be contained in any of the other convex objects in B. Note that B_i can contribute even if $B_i \subset \bigcup_{j \neq i} B_j$.

3.2 Related work

A brief survey on related problems seems to point that finding a global optimal to this problem is by no means an easy task. For instance, the potato-peeling problem (a.k.a. the convex skull problem) consists in computing the largest convex contained in a given n-vertex object. It has been studied both in 2D and 3D with several convex shapes. Arbitrary convex shapes lead to prohibitive solutions even in 2D cases [5]. Better solutions are known for restricted convex objects. The axis-parallel rectangle of largest area inside a general polygon can be found in $\mathrm{O}(n \log^2 n)$ time [10]; the same bound holds for orthogonal polygons [16] unless further constraints such as orthogonally convexity are met. In three dimensions, no satisfactory solutions are known for arbitrary convex polytopes. Finding the largest bounded box can be reduced to convex programming and therefore can be solved in expected linear time only if the input object is convex [1]. An algorithm to compute an approximate axis-parallel box of a given polyhedron in $\mathrm{O}(n^4 \log^2 n)$ time has also been presented [23], but it requires convexity of the input polyhedron.

In the above inclusion approaches, the output model consists always in a single convex object. But in order to improve occlusion as much as possible, we should extract several convex objects from every input component. The partitioning problem (divide a given object into a disjoint set of simple components) has turned out to be generally well solvable. In 3D, a general polyhedron can be partitioned in $\mathrm{O}(N^2)$ convex parts in $\mathrm{O}(nN^3)$ time, where n and N denote the numbers of edges and reflex vertices, respectively [6]. Unfortunately, partitioning tends to produce small convex parts which are not useful as occluders.

For the covering problem (where overlapping is allowed), however, no satisfactory solutions are known. Even for the 2D case, finding the minimal covering is an NP-complete problem [9]. The only polynomial time algorithms known are for covering orthogonal polygons. Finding the minimum covering of a simply connected (i.e. without holes) n-vertex orthogonal polygon takes $\mathrm{O}(n \log n + nm)$ time [13], where m is the number of edges in the visibility graph that are either horizontal, vertical or form the diagonal of an empty rectangle. However, it is restricted to non-piercing (and thus smaller) rectangles. Better solutions are known for rectangles and squares. Covering a simply-connected orthogonal polygon with a minimum number of squares can be done in $\mathrm{O}(n + k)$ time, k being the number of output squares [3]. Unfortunately, these techniques have not been extended to 3D. Moreover, the covering of potential occluders does not share the optimality criteria of classic covering algorithms. For occlusion culling, we are interested in as large as possible convex components, while classic covering literature looks for low-density (i.e. hardly overlapping) coverings.

3.3 Strategy adopted

Our solution for occluder synthesis involves two stages (Figure 2): *aggregation*, which simplifies the topology of the scene while maintaining conditions 3 and 4 of the problem statement, and *convex extraction*, which extracts a densely overlapping

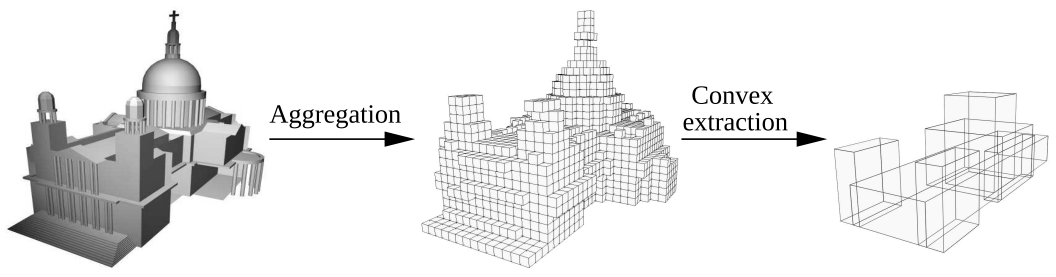


Figure 2: Overview of the synthesis stage

convex covering of the aggregated pointset fulfilling conditions 1 to 5.

In the *aggregation* stage, the computer generates a spatial subdivision representation for the whole polyhedral scene. This volumetric representation, which allows better retrieval of the occlusion properties of the underlying pointset, is the key ingredient for achieving goal 3.

In the *convex extraction* stage, the computer employs a seed algorithm to obtain many overlapping boxes from the volumetric representation. The algorithm is greedy, so the boxes have local maximal occlusion properties. Our tests show that the resultant boxes are quite tight, so the lack of global maximality and the restricted nature of boxes do not diminish occlusion significantly.

A precise description of the Synthesis procedure is given by algorithm 1, where P is the scene and max refers to the maximum depth of an especial octree model discussed below.

```
function Synthesis($P, max, num\_seeds, num\_occluders$)
1.  $mat$:=compute isoorientation matrix ($P$)
2.  transform($P$, $mat$)
3.  $O$:=compute MDCO($P$, $max$)
4.  $L$:= convex extraction($O, num\_seeds$)
5.  $L$:= sort and cut($L, num\_occluders$)
6.  transform($L, mat^{-1}$)
7.  return $L$
end
```

Algorithm 1: Occluder synthesis algorithm

Lines 1 and 2 apply a linear transformation to the scene P so that the main directions of P become axis-aligned [22]. This step improves the size of the occluders generated by the algorithm due to the isothetic nature of the octree.

Line 3 deals with the construction of a maximal division classical octree [4] representation of P with max levels. This is a well known problem based on a simultaneous space subdivision and clipping of the boundary of the polyhedron [4]. Section 3.4 deals with the details of this stage.

A series of convex occluders is added to the list L in Line 4. The function *convex extraction*, whose implementation is discussed in Section 3.5, returns a set of boxes extracted from the octree nodes.

These boxes are then ordered by their volume contribution to the set. Finding the optimal solution would take exponential time, so a heuristic approximation is used instead (algorithm 2). Begining with the biggest box, boxes from the original set B are inserted into the ordered set $B2$. As the contribution is computed with the elements that are already present in $B2$, the order of elements in B will affect the result.

Finally, the inverse transformation given by mat^{-1} is applied to the boxes so that the iso-orientation transformation is undone.

```
function SortAndCut($B$, $num$)
  $x$:= biggest_box($B$)
  $B2$:=insert($B2, < x$,volume($x$)$>$)
  for each $c \in B \setminus$ biggest_box($B$) do
    $v$:=volume($c$)
    $x$:=$c \bigcap (\bigcup B2.first)$
    $v$:=min($v$, volume($x$))
    $B2$:= insert_ordered($B2, < c, v >$)
  end for
  for $i$:=0; $i < num$; $i++$ do
    $B3[i]$:= $B2[i].first$
  end for
  return $B3$
end
```

Algorithm 2: Sort and cut

3.4 Aggregation

The aggregation step has been developed following the Discretized Polyhedra Simplification (DPS) framework of Andújar [2] which is based on a specialization of the classical octree definition. DPS methods do not create convex objects, but their output models are far more convenient for convex extraction than polyhedral representations. This section also discusses some properties of the octree model that will be useful for convex extraction and for bounding the image error of LOD-visibility culling in section 4.

The *maximal division classical octree* [4] (MDCO for short) uses a recursive subdivision of a cubic universe into eight octants that are arranged into an 8-ary tree. As in the classical octree representation [19], each node consists of a code (called color) and eight pointers towards eight sons. Nodes corresponding to cubic regions completely inside the object are labeled as *black* (B), and nodes corresponding to cubic regions completely outside the object are labeled as *white* (W). Black and white nodes are no further subdivided. Nodes containing a part of the boundary are labeled as *grey* (G) and are recursively subdivided until some maximum depth. Leaf grey nodes are called *terminal grey* (TG) nodes.

Given an MDCO, a TG node is said to be a *border terminal grey (BTG)* node if at least one of its 6-neighbors is W; otherwise it is said to be an *interior terminal grey (ITG)* node.

From now on we use calygraphic letters for denoting set of nodes. $\mathcal{B}(O_i)$ denotes the set of black nodes of an octree O_i with depth i, $\mathcal{W}(O_i)$ is the set of white nodes, and so on.

DPS methods are based on two versions of the Hausdorff distance between two pointsets A, B. The *directed* version, denoted as $d_H(A,B)$, is defined as:

$$d_H(A,B) = max_{a \in c(i(A))} min_{b \in c(i(B))} dist(a,b)$$

where the inclusion of $c(i(A))$, which denotes the closure of the interior of A, is a matter of emphasizing the fact that the distance is defined on the points in the inner side of the objects instead of taking into account just the boundary points. The *symmetric* version, denoted as $d_{SH}(A,B)$, is defined as $max(d_H(A,B), d_H(B,A))$.

Now consider the following three pointsets implicitly defined by an MDCO (see Table 1).

Definition 3.1 *Given a MDCO O corresponding to a scene P, the pointsets $R^B(O)$, $R^{B+ITG}(O)$ and $R^{B+TG}(O)$ are defined as:*

$$R^B(O) = \mathcal{B}(O) \tag{1}$$

$$R^{B+ITG}(O) = \mathcal{B}(O) \cup \mathcal{ITG}(O) \tag{2}$$

$$R^{B+TG}(O) = \mathcal{B}(O) \cup \mathcal{TG}(O) \tag{3}$$

Note that the maximum radius of a sphere enclosed either in $R^{B+ITG} - P$ or in $R^{B+TG} - P$ is less or equal than length ε of the diagonal of a terminal node.

3.5 Convex extraction

This step decomposes the pointset $\mathcal{B}(O)$ generated by the aggregation stage into a set B of maximal convex occluders B_i whose union is approximately a covering of O. More precisely,

Definition 3.2 *The shadow from a point p of a set $A \subseteq I\!R^3$ is*

$$S(p,A) = \{q \in I\!R^3 \mid \overline{pq} \bigcap A \neq \emptyset \ \wedge q \notin A\}$$

where $\overline{pq}$ is the segment between p and q

Definition 3.3 *A set B of possibly intersecting convex objects $\{B_i\}$ is called a conforming partial covering*[1] *of a pointset O iff*

$$\forall p \notin O \quad S(p, \bigcup B_i) \subset S(p,O) \tag{4}$$

Note that the containment comparison is established between the shadow frustra instead of the objects themselves and hence in some cases B_i can be partially outside O. The fact that $B \not\subset O$ is another difference with respect to classic covering problems.

Algorithm 3 describes our proposal to obtain a conforming partial covering from a MDCO O. It first traverses the octree nodes in pre-order, inserting black nodes in a list S. This list is then sorted by volume so that the biggest nodes appear first.

Table 1: Properties of octree subsets.

Pointset	Defined as	Difference with P	Distance bound
$R^B(O)$	$\mathcal{B}(O)$	$R^B(O) - P = \emptyset$	$d_H(R^B(O), P) = 0$
$R^{B+ITG}(O)$	$\mathcal{B}(O) \cup \mathcal{ITG}(O)$	$R^{B+ITG} - P \subset \mathcal{ITG}(O)$	$d_H(R^{B+ITG}, P) = \varepsilon$
$R^{B+TG}(O)$	$\mathcal{B}(O) \cup \mathcal{TG}(O)$	$R^{B+TG} - P \subset \mathcal{TG}(O)$	$d_H(R^{B+TG}, P) = \varepsilon$

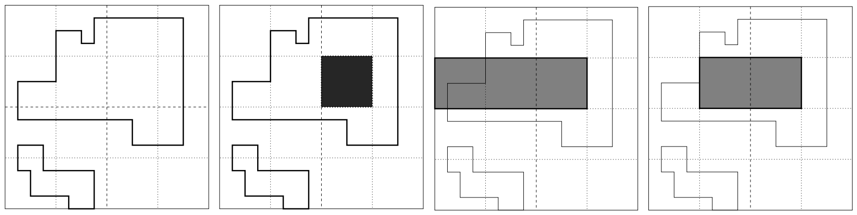

(a) Original octree (b) Input seed (c) Expanded box (d) Trimmed expanded box

Figure 3: Example of one step (-X direction) of the *expand* function.

```
function ConvexExtraction ($O$, $num\_seeds$)
   $S$:=preorder($O$)
   sort($S$)
   $B$:=$\emptyset$
   while $num\_seeds > 0$
      $n$:=head($S$)
      insert($B$, expand(box($n$),$O$))
      $num\_seeds$:= $num\_seeds - 1$
   end while
   return $B$
end
```

Algorithm 3: Convex extraction

The elements of this list will be used as seeds by the expansion function (algorithm 4).

The *expand* function (algorithm 4) employs a greedy heuristic procedure to enlarge as much as possible a given seed c within the limits imposed by the black nodes of the MDCO O. For each of the six axis-aligned directions d, it first computes a tentative box x by stretching c in direction d such that x reaches the border of the universe of o (see Figure 3). This maximally stretched box x is then trimmed against the black nodes of o by the *trim* function. This function performs a sweep of the octree in the direction specified by the parameter d (algorithm 5). As no octree node is visited twice by the *trim* function (and many are not visited at all), its cost is $O(n)$, where n stands for the number of nodes of the octree.

Note that, as usually happens with greedy algorithms, the order in which directions are chosen by *expand* drastically affects the shape (and therefore the goodness) of the resulting box. This could be alleviated by running *expand* several times with randomly shuffled orderings of directions. However, our tests show that this is not necessary if several seeds are used.

The *expand* function ends with a call to *fill_surface_concavities*. The goal of this function is to avoid concavities due to non-crossing holes of the object. The occluder will keep the same visibility properties provided that these concavities do not modify the object silhouette. The $fill\_surface\_concavities$ function enlarges extracted boxes beyond the limits of the octree while preserving visibility properties. For each of the six

[1]Technically it is not a covering, but we kept the term for its intuitive connection.

faces of the input box, the filling function generates a band of interior radius equal to the width of the smallest possible MDCO node. This band is formed by four boxes. They are then stretched in the direction of the face normal till the border of the octree is reached and then trimmed against the black nodes of the octree. The smallest box is then used to measure the actual stretch that can be performed on the input box. Figure 4 shows a complete example in 2D.

```
function Expand($box$, $O$)
  for each $d \in \{+X, -X, +Y, -Y, +Z, -Z\}$ do
    $x$:= MaximumBox($box, d, O$)
    (* Compute the maximum trimmable length $l$ *)
    $< \_, l >$:= trim($x$, $d$, rootnode($O$))
    (* And enlarge $x$ accordingly *)
    $box$:=stretch($x, d, l$)
  end for
  $x$:= fill surface concavities ($x, O$)
  return $x$
end
```

Algorithm 4: expand

```
function Trim($box$, $d$, $node$)
  $x$:= $box \cap node$
  if $x$=$\emptyset$ then
    $< b, l >$:=$< false, \_ >$
  end if
  if colour($node$)=WHITE then
    $< b, l >$:=$< true, 0 >$
  else if colour($node$)=BLACK
    $< b, l >$:=$< true$,length($x$)$>$
  else
    $< b, l, nexts >$:= trim_sons ($box, d, node, Left[d]$)
    if $\neg b$ then
      $< b, l, \_ >$:= trim_sons ($box, d, node, Right[d]$)
    else
      if reaches right neighbours($box, d, node, l$)
        $< \_, l2, \_ >$:= trim_sons ($box, d, node, nexts$)
        $l$:= $l + l2$
      end if
    end if
  end if
  return $< b, l >$
end
```

Algorithm 5: Trim. $Left$ and $Right$ are 6×4 tables that provide an ordering for the traversal of the sons of the node n. $Left$ nodes are always visited first.

```
function TrimSons($box$, $d$, $node$, $s$)
  $< min_l, n >$:= $< \infty$,EmptyVector()$>$
  for $i$:=0;$i < size(s)$;$i++$ do
    $< b, l >$:= trim($box, d$,son($node, s[i]$))
    if $b \wedge l < min_l$ then
      $min_l$:=$l$
      $n.push\_back(Neighbor[d][s[i]])$
    end if
  end for
  return <size($n$) $\neq 0, min_l, n >$
end
```

Algorithm 6: trim sons

4 LOD visibility culling

In this section we present a novel approach to the visibility analysis of very large models. We call our approach LOD visibility culling because visibility computations are carried out using several LOD representations of the occluders with varying visibility accuracy.

The basic idea is that the occluders can be processed using a coarse representation of them without seriously affecting the resulting image quality. When considering an occluder at low resolution, small see-through regions such as holes and transparent polygons can be considered as being opaque, therefore allowing the generation of large occluders. Similar occluder enlargement can be achieved by combining nearby pointsets into larger pointsets and extracting convex occluders from them. In this case some occluders are lossy and an error measure is required.

A precise description of the LOD-based version of the synthesis procedure is given in Algorithm 7, where O_l stands for the octree of the scene processed at resolution l.

```
function Synthesis($P$, $max$)
1.   $mat$:=compute isoorientation matrix ($P$)
2.   transform($P$, $mat$)
3.   $O$:=compute MDCO($P$, $max$)
4.   $L$:=$\emptyset$
5.   insert($L$, Convex extraction( $R^B(O_{max})$, $mat$))
6.   $l$:=$max - 1$
     while $l > 2$ do
7.       if significant topology changes($O_l$, $O_{l+1}$) then
8.         insert($L$, Convex extraction( $R^{B+ITG}(O_l)$, $mat$))
         end
9.       $l$:=$l - 1$
   end
   return $L$
end
```

Algorithm 7: LOD-based synthesis algorithm

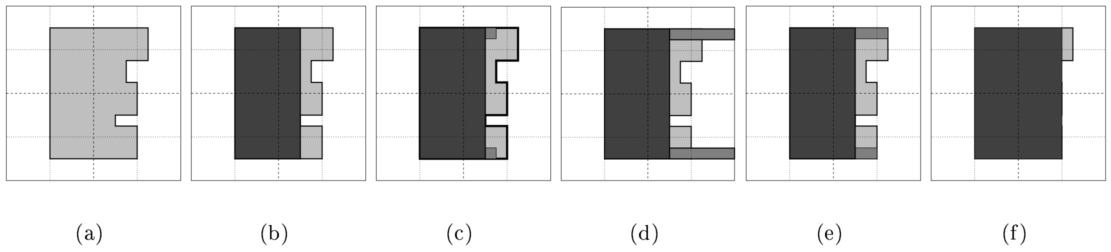


Figure 4: Filling surface concavities in 2D

Note that loss-less occluders are obtained by considering the black nodes of the octree at its maximum resolution (Line 5) and lossy occluders are extracted from the pointset defined by $R^{B+ITG}(O_l)$ for several values of l (Line 8).

Line 7 checks for topology changes between $R^{B+ITG}(O_l)$ and $R^{B+ITG}(O_{l+1})$ (see below). If there are topology changes, then Line 8 adds further convex occluders to L by considering the octree at a lower resolution. Note that in this case, the lossy pointset R^{B+ITG} is considered instead of R^B.

The question that must be answered in Line 7 is whether the convex extraction function will be able to extract larger occluders from $R^{B+ITG}(O_{l-1})$ with respect to those extracted from $R^{B+ITG}(O_l)$. Obviously, the answer is related to the topology changes experimented by the pointset when we double the size of the terminal nodes of the octree. The answer is affirmative if there exists an ITG node in O_{l-1} with at least a white or BTG node among their sons in O_l. In this case, there exists a region in $R^{B+ITG}(O_{l-1})$ that does not belong to $R^{B+ITG}(O_l)$ and hence we have a chance of finding a larger convex object inside it. Note that the previous test requires only a simple traversal of the octree.

4.1 Error evaluation

Since our occluders are overflow approximations of real scene objects (except those extracted from R^B pointsets), we are interested in bounding the image error produced by removing from the display list some objects that were not completely obscured by actual scene objects.

Let c be a camera definition including the viewpoint p, the target t, the horizontal and vertical field-of-view angles fov_h, fov_v and the viewport resolution in pixels, $w \times h$. The image produced by rendering a given set S of scene objects from c will be referred to as $\mathcal{I}(S, c)$.

Let A, B be two subsets of the scene objects, and c be a camera definition. The *differential region* $\Pi(A, B, c)$ is defined as:

$$\{(x,y) \mid \mathcal{I}(A,c)[x,y] \neq \mathcal{I}(B,c)[x,y]\}$$

i.e. $\Pi(A, B, c)$ is the set of pixels that are different on the renderings of A and B.

We propose two intuitive and useful metrics over $\Pi(A, B, c)$ for determining a scalar value representing the difference between the corresponding two images. The first metric has some connection with the L_1 norm and is defined as $L_1(\Pi)$=number of pixels in Π. The second metric, which derives from the L_∞ norm, is denoted as $L_\infty(\Pi)$ and defined as the diameter of the largest enclosed circle in Π.

Clearly, the former expresses the area of the differential region, and the later represents its maximum *thickness*. Both are complimentary because human perception of n randomly distributed pixels is different from that of n pixels forming a compact block. Note that other more sophisticated error metrics between two images (see Neumann et al. [17]) make no sense in this context because the pixels in $\mathcal{I}(A, c) - \Pi(A, B, c)$ remain unchanged.

Now, let us consider the differential region generated by using approximate occluders. Let B_{ij} be the set of boxes generated from O_j. Scene objects lying inside $S(p, B_{ij})$ are not completely invisible from viewpoint p because B_{ij} obscures some regions that were not obscured by the original scene. Therefore, some objects that would be visible through small holes will not be rendered.

We associate along with each box B_{ij} two device-independent errors $e_a(B_{ij})$ and $e_t(B_{ij})$ indicating a bound of the area and the thickness of $\Pi(Scene, Scene - S(p, B_{ij}))$. By device-independent we mean that this measure does not depend on the screen resolution, which is not generally available at preprocessing time. This is achieved by considering the viewpoint P at infinity (i.e. a parallel projection) and a 1×1 screen.

The computation of $e_t(B_{ij})$ is straightforward. It follows from the properties in Table 1 that $e_t(B_{0j}) = 0$ and $\forall i > 0 \; e_t(B_{ij}) = 2\varepsilon_i$.

The *AssignErrors* procedure computes the error $e_a(B_{ij})$ corresponding to the boxes B_ij extracted from the octree O_i where ε_i is again the length of the diagonal of a terminal node of O_i.

```
procedure AssignErrors($B_{ij}$, $O_i$)
  for $j$:=1 to $b$ do
    1. $\Gamma_{XY}$:=front back ITG($B_{ij}$, $O_i$, XY)
    2. $\Gamma_{YZ}$:=front back ITG($B_{ij}$, $O_i$, YZ)
    3. $\Gamma_{XZ}$:=front back ITG($B_{ij}$, $O_i$, XZ)
    4. $\Gamma_{max}$:= $\sqrt{3}$ max $\{\Gamma_{XY}, \Gamma_{YZ}, \Gamma_{XZ}\} * \varepsilon_i$
    5. $e_a(B_{ij})$:=$\Gamma_{max}$
  end
end
```

Lines 1-4 deal with the computation of the area of the maximum parallel projection of the *ITG* nodes inside B_{ij}. Let $\Gamma_w(x)$ be the parallel projection of an arbitrary pointset x over the plane w. It follows from the properties in Table 1 that for any w

$$\Gamma_w(B_{ij} - P) \subset \Gamma_w(\mathcal{ITG}(O_i) \cap B_{ij}) \; .$$

Let max be the plane over which the projection $\Gamma_{max}(\mathcal{ITG}(O_i) \cap B_{ij})$ is maximum. A precise bound of the area of this maximum parallel projection can be easily computed from the orthographic projections of the ITG nodes (lines 1 to 4).

Note that the area of any orthographic projection of the *ITG* set can be trivially computed by a traversal of the front and back ITG nodes (Lines 1 to 3). The *assign errors* algorithm runs in O(bn) time, b and n being respectively the number of boxes and the number of octree nodes at resolution i.

Finally, we group all the boxes in $\{B_{ij}\}$ by similar thickness error e_t. For example, we create five categories by considering the intervals $[0,0]$, $(0, 0.01]$, $(0.01, 0.02]$, $(0.02, 0.05]$, $(0.05, 0.1]$ and $(0.1, \sqrt{2}]$. Note that e.g. $e_t = 0.01$ corresponds to limiting the thickness of the differential region to one percent of the screen diagonal.

Working with LOD representations of the occluders requires some modifications of the occluder selection once the viewer's position in known. At runtime or pre-processing time, depending on the visibility determination approach adopted, the user provides two error tolerances E_t, E_a indicating the maximum thickness and maximum area of the differential region. Occluders are selected with increasing e_a so that $\sum e_a(B_{ij}) \leq E_a$ and $\max\{e_t(B_{ij})\} \leq E_t$, where the perspective correction has been applied to e_a and e_t. Actually, $L_\infty(\Pi(S, S - V_i), c_1)$ is bounded by $e_t(V_i)$ provided that the projections of *ITG* of different occluders do not overlap, which is unlikely to happen.

5 Results

We have implemented all the algorithms presented in this paper and tested them on several publicly available 3D models. Figure 5 shows the results of our lossy version of the occluder synthesis algorithm on a model of the St. Pauls cathedral from *http://www.3dcafe.com/models/stpauls.zip* (14,780 faces). On both cases the covering was above 70% and the maximum error, measured as the deviation from the occluder to the original surface was less than 2.7% and 1.35%, respectively.

We did some experimentation to check the dependence of our method on the number of occluders and seeds. In order to get an accurate evaluation of the goodness of the synthesis algorithm, we ran the examples with a naive version of the *sort and cut* filter which only considers the difference between the volume of the box whose contribution is being evaluated and the volume of its maximum intersection with the boxes selected previously.

Figure 7 shows that twelve occluders and twice that number of seeds suffice to cover significant portions (more than 60%) of the St. Pauls cathedral. And Figure 6 reveals that regardless of the number of occluders adopted, it is easy to achieve coverings above 65% with a few seeds.

Figure 8 shows graphically the differential region using the lossy occluders generated from the 7-level octree of the St. Pauls model ($e_t \leq 1.35\%$). The red pixels in Figure 8 (b) correspond to the screen region covered by the occluders but not covered by the original object. Although any red pixel can be erroneous, usually only a little subset of these

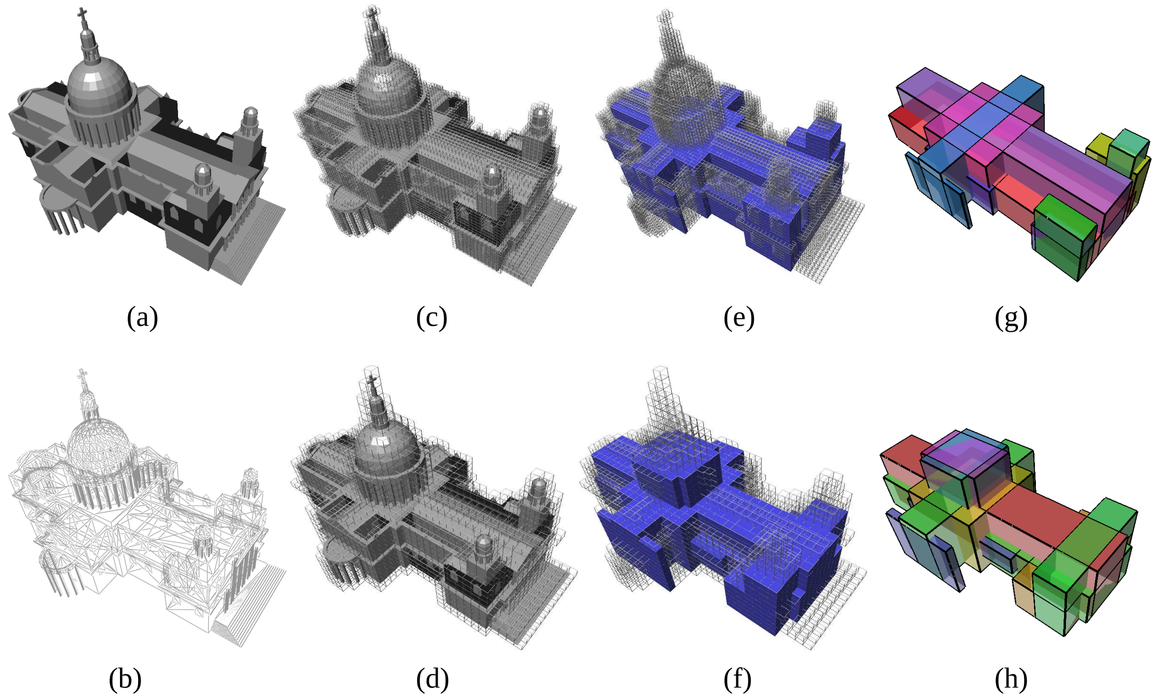
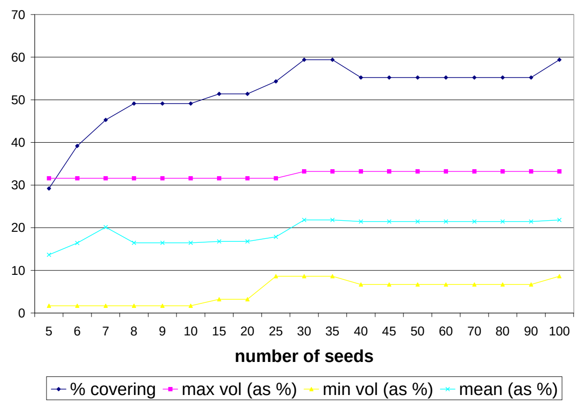
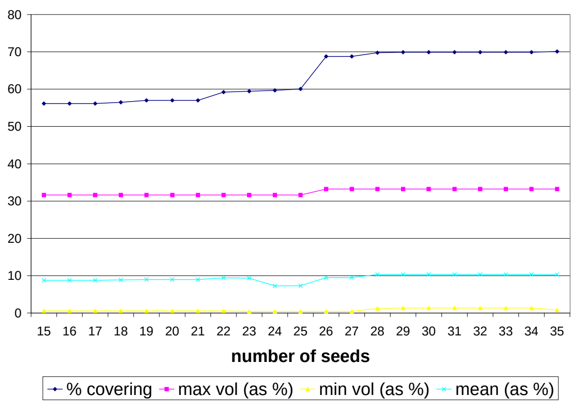


Figure 5: Occluder synthesis on the St. Pauls model. St. Pauls model *(a)* and its tessellation *(b)*; MDCO with seven *(c)* and six levels *(d)*; the 10 largest occluders extracted from each resolution *(e-h)*.



(a) 5 occluders



(b) 15 occluders

Figure 6: Performance of the synthesis algorithm with 5 and 15 occluders from the St. Pauls model. Each graph shows the covering percentage (with respect to the total volume), and the maximum, minimum and mean volume of the boxes.

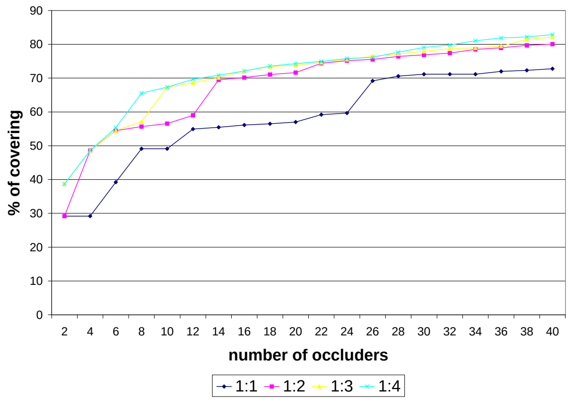


Figure 7: Performance on the percentage of covered volume with different occluder/seeds ratios in the St. Pauls model.

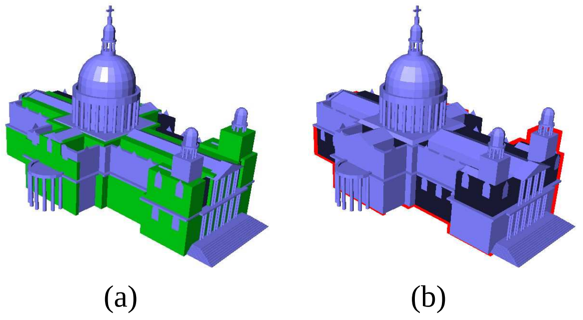


Figure 8: The original St. Pauls cathedral in blue. In green and on the left, its synthesized occluders. Those parts of the occluders that are not contained in the model are depicted on the right in red.

points are actually wrong because for a visible polygon to be considered as not visible, its screen projection must be completely enclosed in the red and blue area.

In order to measure the actual improvement on occlusion power, we tested our method on a highly occluded environment. We chose a publicly available city model (Figure 9) from *http://www.3dcafe.com/models/pcacity.zip* with 182,915 triangles. We took 993 samples along a path crossing the city and culled the scene using raw and loss-less synthesized occluders separately. For each occluder set, and for each sample, occluders that were inside a fixed-size box centered at the sample viewpoint were selected. The box was sufficiently big to include all the significant occluders. As a matter of fact, it included many more: thousands of triangles in the case of raw occluders and less than fifty in the case of synthesized ones. The simulation algorithm then chose the 30 occluders with better occlusion behavior and discarded the rest. The sets of visible polygons of the remaining occluders were then mixed to obtain a potentially visible set of the current sample. Note that this occluder selection step does certainly select the best possible triangles, so the simulation is a little biased in favor of raw occlusion. Visibility culling algorithms do not have the time to do such an extensive search at each viewpoint.

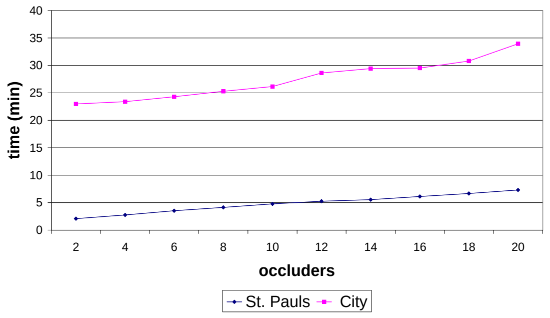


Figure 11: Running times of the convex extraction algorithm on the St. Pauls and the city model.

Figure 10 shows the differences between the number of culled polygons using only raw occluders and only synthesized occluders. It is clearly favorable to the use synthesized occluders. The minimum and maximum values of the difference between synthesized and raw were -20,860 and 161,884 polygons, respectively. Its mean was of 78,694 polygons with a standard deviation of 34,570. Our method allowed an extra culling of a 40% of the scene and enhanced occlusion significantly at the vast majority of the samples.

Running times of the convex extraction algorithm on the St. Pauls and the city model are shown in Figure 11. The 9-level octree construction of the St. Pauls cathedral and the city model took 4 and 20 minutes respectively on a 194 MHz R10000 MIPS processor.

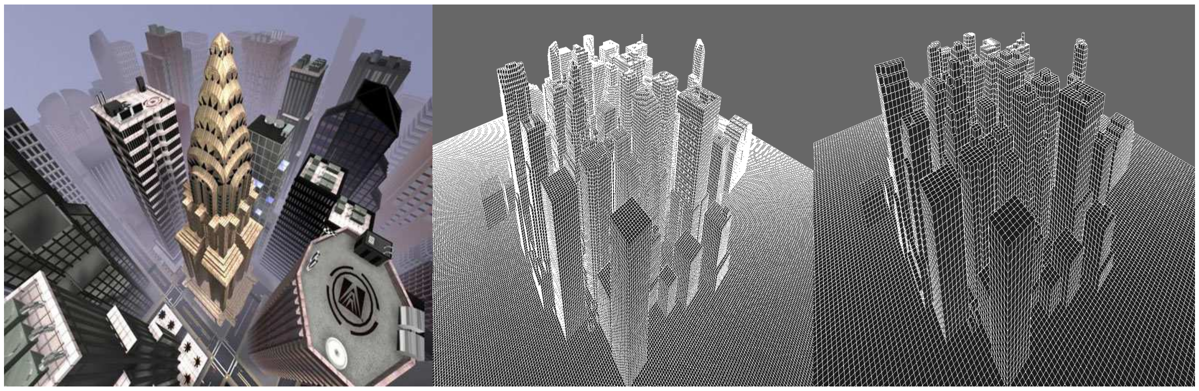


Figure 9: From left to right, the city model and its octree representation with nine and eight levels.

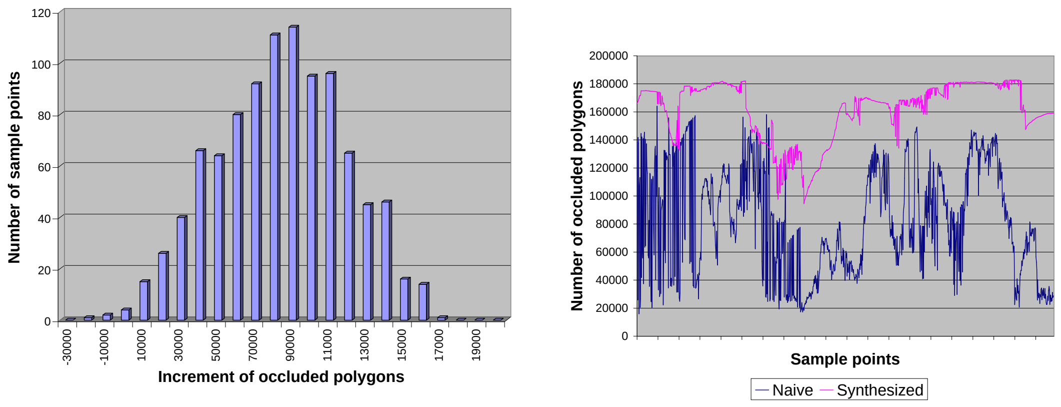


Figure 10: Charts on the occlusion improvement

6 Conclusions and future work

We have introduced LOD occlusion culling, a new concept that accelerates navigation in complex scenes and that can be incorporated into current navigation frameworks that use visibility culling. The experimental results show an improvement of visibility computations, the generation of more accurate PVS, and high quality images without visibility error (loss-less visibility) or with a fitted image error (lossy visibility) by means of two intuitive parameters indicating the maximum thickness and area of the error region.

We have also developed a new algorithm for the generation of synthesized occluders. The algorithm has two main steps: aggregation and convex extraction, and it guarantees the visibility validity of the output occluders. It generates sequences of densely overlapping boxes that need not to be inside the original concave pointset. Each of these two features enhances occlusion performance. Modifying the aggregation step, the algorithm can be used to obtain multiresolution occluder sets. This algorithm benefits not only from complex and concave regions, but also from almost opaque ones, thus greatly accelerating the visualization.

The occluder selection step also benefits from the use of synthesized occluders. Dynamic occlusion algorithms restrict the number of selected occluders used at each viewpoint to avoid overhead. But as the number of elements of the synthesized set is many orders of magnitude smaller than the number of polygons in the input, the selection of occluders at each viewpoint is faster. And more occluders can be taken into consideration without incurring in excessive overhead.

Our future work includes the integration of LOD-occlusion into our visibility algorithm [20]. The occluder synthesis algorithm will allow the generation of level-of-detail hierarchies of synthesized occluders with varying degrees of visibility accuracy. As in LOD rendering, image fidelity can be traded for a higher frame ratio. We plan to use lossy occluders for determining both completely visible and hardly visible objects, which can be displayed using a coarse representation of them.

Storing a high-resolution octree is very memory-consuming. Fortunately, it is not necessary to have simultaneously all the terminal nodes of the octree subdivided until the maximum subdivision level. During the top-down construction of the octree, it can be decomposed into strongly connected subsets which can be refined separately.

Acknowledgments

This work has been partially supported by the TIC98-0586-C03-01 project and an FPI grant of the Spanish Ministry of Education and Science. We would also like to thank CESCA, CEPBA and C^4 for the use of their resources, and 3DCafe and Platinum Pictures for providing the cathedral and the city datasets.

References

[1] Nina Amenta. Bounded boxes, hausdorff distance, and a new proof of an interesting helly theorem. In *Proceedings of the 10th Annual Symposium on Computational Geometry*, pages 340–347, Stony Brook, NY, USA, June 1994. ACM Press.

[2] Carlos Andújar. *Octree-based Simplification of Polyhedral Solids*. PhD thesis, CS Dept, Universitat Politècnica de Catalunya, Barcelona, Spain, 1999.

[3] Reuven Bar-Yehuda and Eyal Ben-Hanock. A linear-time algorithm for covering simple polygons with similar rectangles. *International Journal of Computational Geometry & Applications*, 6(1):79–102, 1996.

[4] P. Brunet, R. Joan, Isabel Navazo, A. Puig, J. Sole, and D. Tost. Modeling and visualization though data compression. In R.E. Earnshaw and L. Rosenblum, editors, *Data Visualization*, pages 157–169. Academic Press, 1994.

[5] J. S. Chang and C. K. Yap. A polynomial solution for the potato-peeling problem. *Discrete & Computational Geometry*, 1:155–181, 1986.

[6] B. Chazelle and D. P. Dobkin. Optimal convex decompositions. In *Proceedings of the Symposium on Computational Geometry (Baltimore, MD,June 5–7, 1985)*, pages 63–133. ACM, ACM Press, 1985.

[7] Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–253, 1998.

[8] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 83–90. ACM Press, April 1997.

[9] Joseph C. Culberson and Robert A. Reckhow. Covering polygons is hard. *Journal of Algorithms*, 17(1):2–44, July 1994.

[10] K. Daniels, V. Milenkovic, and D. Roth. Finding the largest area axis-parallel rectangle in a polygon. *Computational Geometry: Theory and Applications*, 7:125–148, 1997.

[11] T.A. Funkhouser and C.H. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH*, pages 247–254, 1993. Computer Graphics Proceedings, Annual Conference Series.

[12] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frusta. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 1–10, June 1997.

[13] J. Mark Keil. Covering orthogonal polygons with non-piercing rectangles. *International Journal of Computational Geometry & Applications*, 7(5):473–484, 1997.

[14] James T. Klosowski and Cláudio T. Silva. Rendering on a budget: A framework for time-critical rendering. In *IEEE Visualization*, 1999.

[15] Fei-Ah Law and Tiow-Seng Tan. Preprocessing occlusion for real-time selective refinement. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 47–53, 1999.

[16] M. McKenna, J. O'Rourke, and S. Suri. Finding the largest rectangle in an orthogonal polygon. In *Proc. 23rd Allerton Conference on Communication, Control and Computing*, pages 486–495, 1995.

[17] L. Neumann, K. Matkovic, and W. Purgathofer. Perception based color image difference. In *Proc. Eurographics, Lisbon, Portugal.*, 1998.

[18] T. Nishita, I. Okamura, and E. Nakamae. Shading models for point and linear sources. *ACM Transactions on Graphics*, 4(2):124–146, April 1985.

[19] H. Samet. Applications of spatial data structures. *Computer Graphics, Image Processing and GIS*, 1990.

[20] Carlos Saona-Vázquez, Isabel Navazo, and Pere Brunet. The visibility octree. A data structure for 3D navigation. *Computers & Graphics*, 23(5):635–643, 1999.

[21] Carlos Saona-Vázquez, Isabel Navazo, and Pere Brunet. Data structures and algorithms for navigation in highly polygon-populated scenes. In P. Brunet, C.M. Hoffman, and D. Roller, editors, *CAD Tools and Algorithms for Product Design*. Springer, 2000.

[22] Xiaolin Wu. A linear-time simple bounding volume algorithm. In David Kirk, editor, *Graphics Gems III*, pages 301–306. Academic Press, 1992.

[23] Binhai Zhu. Approximating convex polyhedra with axis-parallel boxes. *International Journal of Computational Geometry & Applications*, 7(3):253–267, 1997.