Visibility as an Intrinsic Property of Geometric Models

Adi Bar-Lev Gershon Elber

Department of Computer Science
Technion, Israel Institute of Technology
Haifa 32000,
Israel
Email: gerhson@cs.technion.ac.il
Phone: 972-4-829-4338
Fax: 972-4-822-1128

Abstract

This paper presents a technique to improves the performance of algorithms that exploits visibility based analysis in computer graphics, such as ray tracing. The presented approach maintains a list of visible polygons for each and every one of the polygons in the model as well as for, optionally, the light sources in the scene. A pre-processing visibility analysis stage that is view-independent is computed once per scene, creating a Visibility Data Structure (VDS) that becomes part of the model. This, before any visibility intensive computer graphics algorithm takes place. The approach presented may be combined with most acceleration methods for ray tracing such as octree *or* voxel *based spatial subdivision. Furthermore, any algorithm that requires visibility computations can exploit the VDS, from* Radiosity *to* NC machining. *Though the presented technique is fairly intuitive, it demonstrates the usability of a visibility data structure of a scene.*

1. Introduction

One of the most important image synthesis methods in contemporary computer graphics is the ray tracing [4] technique, a scheme which can produce almost photo realistic images. Nonetheless, the ray tracing technique consumes an excessive amount of time during the rendering process, a limitation that has been alleviated via numerous optimization methods which we briefly survey here.

Bounding volumes - One of the first acceleration methods that has been used with the ray tracing algorithm. A bounding volume is a volume which completely contains an object in the scene or a group of such objects. The main idea behind this method is to replace objects in the scene with an easy to compute volumes. The intersection test now has two phases - the first involves a simple ray-volume intersection test, and the second test with all the polygons inside the volume, thus making significantly fewer computations than before. Whitted [11] used spheres as bounding volumes, observing that they are the simplest volumes for intersection tests. A more advanced technique which was derived from the bounding-volumes, was the hierarchical bounding volumes technique which was introduced by Rubin and Whitted [7] as a method to decrease time complexity by a logarithmic factor depends on the number of objects in the scene and their order. Later work by Kay and Kajiya [6] et. al., tried to tighten the volume so that a ray intersecting the bounding volume is almost always intersecting the poly-

gons within it.

3-D spatial subdivision - much like the bounding volume method, this method also tries to reduce the time complexity by eliminating unnecessary Ray-Surface Intersection (RSI) tests with polygons which can not possibly intersect with the ray. One of the first techniques developed following this line, divided the space containing the scene into voxels and traversed the voxels along the ray, each time considering the intersections only with the polygons in the voxels that are being traversed along the path. A more advanced spatial subdivision technique exploits *octal trees (octrees)* [5] to recursively subdivide the space into eight sub-regions. This subdivision is conducted until a maximum level of division is achieved or until there is no more than one polygon in the voxel. Using this technique, one can logarithmically reduce the computation time, for the same reason as the hierarchical bounding volumes. See [5, 2, 1] for other varieties of spatial subdivisions. Other approaches to the spatial subdivision use a hierarchical uniform space division approach, thus gain both the simplicity of the uniform space division technique, and the speed of the octal trees technique by using a refined division. [12, 13].

First-hit - the ray tracing algorithm can also be accelerated when taking the viewpoint into consideration. Given a view point, a scan-line algorithm can be used to sort out the first polygon being hit by each first ray. (for more details see [10]). This technique reduces the amount of computation time spent on the first-ray casting, to the level of a simple scan-line computation cost.

We have introduced three basic approaches to acceleration of the ray tracing algorithm. More details on these methods as well as many other methods can be found in [4] and [9].

Consider a scene that is (mostly) static. When a ray, $\mathcal{R}$, bounces off an object, $\mathcal{O}$, only a subset of objects in the scene might get hit by $\mathcal{R}$. Because the scene is assumed static, one can, a-priori, compute the set of visible objects that $\mathcal{O}$ can see. Clearly $\mathcal{R}$ might intersect only one of these objects that are visible to $\mathcal{O}$.

Two fundamental issues then need to be addressed. First and foremost, how can one compute or even approximate this Visibility Data Structure (VDS). In addition, assuming the existence of the VDS, how can it be exploited in various algorithms that require visibility computations.

While the notion of a *model* is typically associated with its geometry, other entities are also frequently considered part of the model, such as the model's topology, material properties and texture, etc. It only seems natural to expect that the VDS will also become part of the (static) model. Research on visibility is not new, including in the context of efficient rendering [8, 3]. This paper attempts to stress the importance in the *consideration of this VDS a portion of the model*.

In section 2, we introduce the concept of visibility between the polygons in a given scene. In section 3, we use this characteristics to reduce the time spent on RSI test for each reflected ray, for ray tracing. Our method does not deal with the first ray since this ray is view-dependent, and the "First Hit" acceleration scheme can be employed in order to reduce the computation time at this stage. Section 4 demonstrates our modified ray tracer that exploits the VDS on several different scenes, while we conclude in section 5.

2. The Construction Of The VDS:

2.1. Overview and Definitions:

Definition 1 *point p see object $\mathcal{O}$, iff there exist a point $q \in \mathcal{O}$ such that the line segment $\overline{pq}$ intersects no other object in the scene.*

Definition 2 *A point set $\mathcal{D} \subset \mathbb{R}^n$ is called a* visibility domain *if every two points $p, q \in \mathcal{D}$ see the same set of objects in the scene.*

Consider line segments P_i in $\mathbb{R}^2$ or polygons P_i in $\mathbb{R}^3$:

Lemma 1 *Given an n-dimensional scene, $n = 2, 3$ with m line segments (polygons), the disjoint partition of $\mathbb{R}^n$ into visibility domains has at most $O(m^4)$ domains.*

proof: Given two polygons, P_1 and P_2, $\mathbb{R}^n$ is subdivided into three visibility domains. One domain where only P_1 is visible, one domain where only P_2 is visible, and one domain where both are visible. These three domains are delineated using a constant number of lines in $\mathbb{R}^2$ (planes in $\mathbb{R}^3$). Given m polygons, we are presented with $O(m^2)$ such lines

(planes). $O(m^2)$ lines in $\mathbb{R}^2$ (planes in $\mathbb{R}^3$) can intersect in at most $O(m^4)$ different points (lines). In general position, each of the $O(m^4)$ intersection points (lines) is shared by four neighboring visibility domains. Each visibility domain employs at least one intersection point (line). Hence, the number of visibility domains is also bounded by $O(m^4)$. ■

The partition imposed by Lemma 1 is most likely to be overly refined because different domains might share the same set of visible polygons. In Figure 1, all the visibility domain inside the circle share the same set of visible polygons. Nevertheless, the bound is hard and in Figure 1 the seven segments in the figure cast $7 * (7 - 1) * 2$ infinite lines with none being parallel. Due to this $O(m^4)$ complexity, the visibility of any scene which is moderately complex is intractable to compute. While numerous approaches exist [14, 15, 16] to heuristically resolve the visibility question, we select in this work a simple method mainly in order to demonstrate its potential capabilities on the computational efficiency, for example, in ray tracing. This method can be further developed and combined with other visibility methods in order to achieve better results in time and space complexity.

> **Definition 3** *Polygon P_j is* invisible *to polygon P_i iff $\forall p \in P_i$ and $\forall q \in P_j$ the line $\overline{pq}$ intersects some polygon P_k. If P_i is invisible to P_j then P_j is invisible to P_i. This relation is symmetric.*

Before we can describe our approach in details, we must define the following:

1. Let Σ be the set of all polygons in the scene.
2. Let $P_i \in \Sigma$ denote a polygon in the given scene.
3. Let $\mathcal{L}_i$ be a light source in the scene.
4. Let $\Gamma_i \subset \Sigma$ be the set of polygons that are considered visible from a polygon P_i, or a light source $\mathcal{L}_i$.

Hereafter, assume that every scene has its VDS that provides for every polygon $P_i \in \Sigma$ and possibly for every light source $\mathcal{L}_i$, the list of its visible polygons Γ_i.

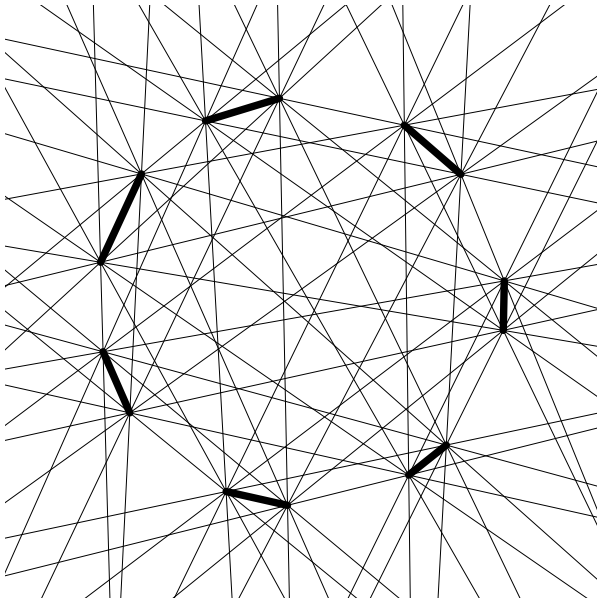

Figure 1. The complexity of computing all the visibility domains of a given scene is $O(m^4)$, where m is the number of segments or polygons in the scene.

> **Definition 4** *A VDS is said to be* conservative *iff for every two visible polygons P_i and P_j in the scene, $P_j \in \Gamma_i$ and $P_i \in \Gamma_j$.*

Consider the extreme case for which $\Gamma_i \equiv \Sigma$ for all polygons, P_i, in the scene. While highly conservative, this construction is a valid VDS.

Our approach to approximate the VDS exploits this conservative methodology. We indeed start by setting $\Gamma_i \equiv \Sigma$ for all polygons in the scene. We then attempt to remove hidden polygons from P_i's visibility data structure Γ_i, using *single polygons* in the scene as occlusion buffers. Doing so, enables one to alleviate the complexity of the problem down from $O(m^4)$. In contrast, in order to find the exact VDS solution of each polygon in a given scene, one must

test each polygon against each of the $O(m^4)$ domains, raising the total complexity even higher. Hence, the exact solution is practically intractable. By computing the approximated VDS of occlusions of single polygons, we significantly reduce this complexity while remaining conservative. It is clear that with this reduction of the complexity, we also give up efficiency due to polygons which will be included in the VDS although they are actually occluded by several polygons simultaneously, in the scene.

Two types of visibility testing are conducted during the VDS construction. The first involves the reduction of the amount of polygons that are considered visible in each polygon's Γ_i. The second tries to reduce the amount of polygons in each Γ_i of the light sources, thus aiming at the reduction of the amount of light rays which will be cast during, for example, image ray tracing process.

The first type of visibility test employs two main stages that are denoted the 'Half Space Visibility Test' and the 'Volume Containment Test', while the second type employs only a 'Volume Containment Test'. Starting with $\Gamma_i \Leftarrow \Sigma - \{P_i\}$ for each P_i and $\Gamma_i \Leftarrow \Sigma$ for each $\mathcal{L}_i$, we use in each stage a different approach to reduce the number of polygons that are actually visible in Γ_i for each of the polygons and light sources in the scene.

While the VDS can significantly increase the memory that is required to store geometric models, at the worst case one needs to invest one bit for each (in)visible polygon in Γ_i list of P_i. Therefore, for example, for a scene with 10000 polygons, the memory requirements are bounded from above by ≈ 12 megabytes. Nevertheless, it is clear that in large scenes, the levels of occlusions are going to be quite high, typically over 90%, and therefore the size of the VDS can be made much smaller by using different representation schemes such as hash table of indices.

2.2. Half-Space Visibility Test

This stage eliminates from Γ_i all polygons P_j, that are completely behind the plane containing the polygon P_i. This back-face culling stage applies only to the computation of the visibility of polygons, and is not used on light sources. An example of the test is shown in Figure 2.

For each $P_j \in \Gamma_i$, P_j is behind P_i iff all the vertices of P_j are behind the plane of P_i (see Figure 2), a test that can be easily carried out using the signed result of substituting each of the vertices of P_j into P_i's plane equation. This test does not perform any occlusion test by other polygons, it only attempts to validate the possibility that P_j can be hit by a reflected ray from P_i.

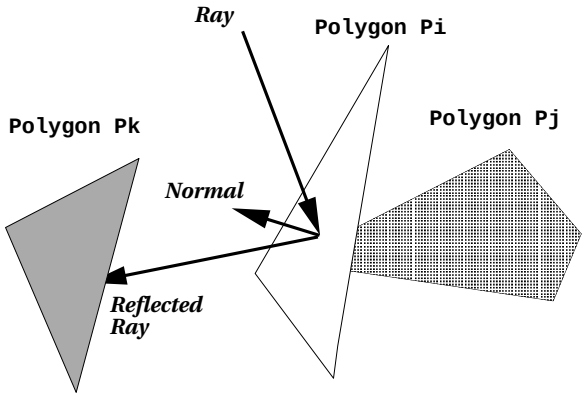

Figure 2. Polygon P_j is behind polygon P_i and so no ray can be reflected from P_i and hit P_j, while polygon P_k can be hit by a reflection as is shown.

In summary of this stage,

Algorithm 1 *Half Space Visibility Test of Polygons.*

For all $P_i \in \Sigma$ *do*
  $\Gamma_i \Leftarrow \Sigma - \{P_i\}$;

For all $P_i \in \Sigma$ *do*
  For all $P_k \in \Gamma_i$ *do*
    If P_k *is behind* P_i*'s plane then*
      $\Gamma_i \Leftarrow \Gamma_i - \{P_k\}$;

The time complexity of this stage equal $O(m^2)$, where m is the number of polygons in the scene.

2.3. Volume Containment Test

The Half-Space testing stage of the visibility algorithm eliminates approximately half of the polygons. Here, we continue to further improve that result, yet remaining conservative. We assume that the polygons in the scene are all convex polygons. Otherwise, they can all be decomposed into convex polygons.

We start by approximating the Γ_i list of light source $\mathcal{L}_i$,

and then continue to consider the computation of the Γ_i list of polygon P_i, in the scene.

Approximating the Γ_i list of a Light Source: Denote by $e_j^l,\ l = 1, r$ the l'th edge of polygon P_j. Given a point light source $\mathcal{L}_i$ at location O_i, and a convex polygon P_j, compute the equations of the planes $\mathcal{P}_i^{jl}$, through O_i and e_j^l, so that the positive side of the equation of $\mathcal{P}_i^{jl}$ contains P_j. Let $\mathcal{P}_i^{j0}$ be the plane containing polygon P_j so that the negative side of the equation of $\mathcal{P}_i^{j0}$ contains O_i.

Denote the *shadow volume* of P_j and $\mathcal{L}_i$ as the intersection of all the half-spaces which are created by the planes $\mathcal{P}_i^{jl},\ l = 0, r$ by,

$$SV_{\mathcal{L}_i}^{P_j} = \bigcap_{l=0}^{r} \mathcal{P}_i^{jl} \tag{1}$$

Clearly, $SV_{\mathcal{L}_i}^{P_j}$ is convex as it is the intersection of convex objects (half spaces). Denote by $v_k^l,\ l = 1, r$ the l'th vertex of polygon P_k. Then,

Lemma 2 *If all the vertices v_k^l of polygon P_k satisfy $v_k^l \in SV_{\mathcal{L}_i}^{P_j},\ \forall l = 1, r$ then P_k is invisible to $\mathcal{L}_i$.*

Proof: Both P_k and $SV_{\mathcal{L}_i}^{P_j}$ are convex. Therefore any convex combination of $v_k^l,\ l = 1, r$ must be contained in $SV_{\mathcal{L}_i}^{P_j}$. On the other hand, all points $p \in P_k$ are convex combination of $v_k^l,\ l = 1, r$. ■

This result is not constraint to point light sources. For a directional light source, $SV_{\mathcal{L}_i}^{P_j}$ is simply the extrusion of P_j in the direction that is prescribed by $\mathcal{L}_i$.

Finally, it should be noted that $SV_{\mathcal{L}_i}^{P_j}$ need not be computed explicitly and the containment of vertex v_k^l in $SV_{\mathcal{L}_i}^{P_j}$ can be verified by testing the sign of the substituted vertex into all the equations of all the planes that form $SV_{\mathcal{L}_i}^{P_j}$. In summary, the visibility test for light sources,

Algorithm 2 *Shadow Volumes of Light Sources.*

For all $\mathcal{L}_i \in$ Scene
$\Gamma_i \Leftarrow \Sigma$;

For all $P_j \in \Sigma$ do
Construct $SV_{\mathcal{L}_i}^{P_j}$ from $\mathcal{L}_i$ and P_j;
For all $P_k \in \Gamma_i$ do

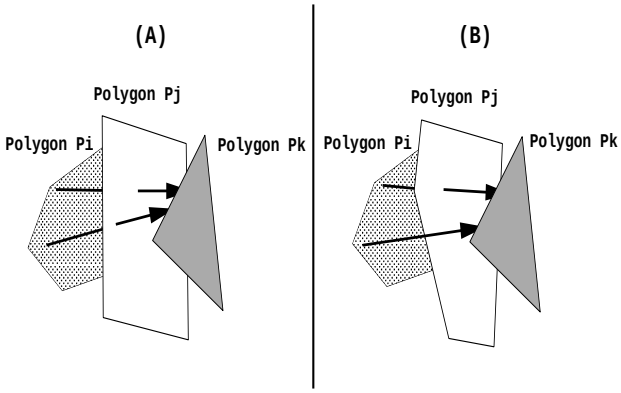

Figure 3. Figure (A) shows the case of two polygons which are invisible to one another due to a complete occlusion by polygon P_j, while in (B) Polygons P_i and P_k are visible by our definition because polygon P_j does not occlude them completely.

If $P_k \subset SV_{\mathcal{L}_i}^{P_j}$ then
$\Gamma_i \Leftarrow \Gamma_i - \{P_k\}$;

The time complexity of this stage for one light source is $O(m^2)$, where m is the number of polygons.

Approximating the Γ_i list of a polygon: For the Γ_i list of a light source, we have considered only one projection point which was the light source's origin or direction. Here, we must consider all points on the surface of a given polygon P_i (see Figure 3).

Given two convex polygons P_i and P_j, construct the shadow volumes $SV_{v_i^l}^{P_j}$ for all vertices $v_i^l,\ l = 1, r$ of polygon P_i, in a similar way to the construction of a shadow volume between a point light source and a polygon. Denote the intersection of these shadow volumes by,

$$SV_{P_i}^{P_j} = \bigcap_{l=1}^{r} SV_{v_i^l}^{P_j}. \tag{2}$$

Again, because all $SV_{v_i^l}^{P_j}$ are convex, so is their intersection, $SV_{P_i}^{P_j}$. Then,

Lemma 3 *Polygon P_k is invisible to P_i if there exist a polygon P_j, such that P_k is contained in $SV_{P_i}^{P_j}$ (see Figure 4).*

Proof: Consider a point $p \in P_i$ and construct a shadow volume from p through polygon P_j, $SV_p^{P_j}$. Because p is a

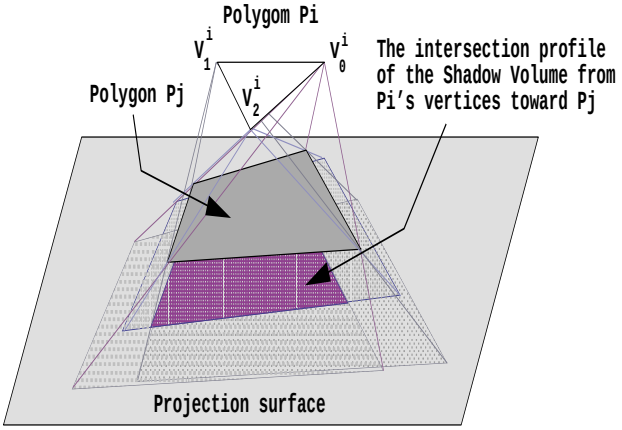

Figure 4. An example of a shadow volume that is created for polygon P_i by polygon P_j (the occluding polygon). The volume itself starts from polygon P_j and through the plane to infinity.

convex combination of v_i^l, $SV_{P_i}^{P_j} \subset SV_p^{P_j}$. We know that $P_k \subset SV_{P_i}^{P_j}$ and therefore $P_k \subset SV_p^{P_j}$. Then, P_k must be contained in all shadow volumes $SV_p^{P_j}$, for all $p \in P_i$, and hence P_k must be invisible to P_i. ■

In summary:

Algorithm 3 *Shadow Volumes' Visibility Test of Polygons.*

For all $P_i \in \Sigma$ *do*
  For all $P_j \in \Gamma_i$ *do*
    Construct $SV_{P_i}^{P_j}$*;*
    For all $P_k \in \Gamma_i$, $k \neq i,\ j$ *do*
      If $P_k \subset SV_{P_i}^{P_j}$*, then*
      begin
        $\Gamma_i \Leftarrow \Gamma_i - \{P_k\}$*;*
        $\Gamma_k \Leftarrow \Gamma_k - \{P_i\}$*;*
      end;

The Complexity of this step is clearly $O(m^3)$ where m is the number of polygons in the scene.

2.4. Optimizations

While the VDS is computed only once per scene, a reduction in the computational cost of the VDS is still desired. With a time complexity of $O(m^3)$ where m is the number of polygons in the scene, the approach continues to be too slow, especially in complex scenes. This cost can be immensely reduced by exploiting a heuristic which takes into account the locations where occlusions might occur. Occlusions are likely to occur by large polygons in the scene, and between polygons that are relatively close to one another. With that in mind, the list of polygons is kept sorted by area. The user can now specify a threshold for an allowed ratio between polygons. Given two polygons P_i and P_j in the scene, the distance between the two polygons as well as their areas are compared against this threshold in order to determine whether or not to conduct a shadow volume occlusion test between the two. This heuristic may reduce the amount of occlusion tests in cases where polygons are far away from each other or are too small in relation to the average polygons' area. A similar heuristic may be employed for occlusion tests of light sources.

It is, however, important to notice that these introduced heuristics preserve the correctness of the *conservative* VDS construction algorithm. This reduction in the number of occlusion tests can only enlarge the number of the polygons in each Γ_i list, and possibly affect the original performance during the exploitation of the VDS by, for example, ray tracing. Nonetheless, with the use of the sorting heuristic, reductions of more than a magnitude in computation time during the construction of the VDS have been observed, and yet the average Γ_i list increased by only 1%.

This section proposed a conservative approach to the computation of the VDS. While simple to construct, this highly conservative VDS has significant effect on the reduction of the computational costs of algorithms that depends on visibility testing such as image ray tracing as we are about to demonstrate in the next section.

3. Ray Tracing Using VDS

We now consider one way to combine and employ the VDS in image ray tracing applications. For each ray, $\mathcal{R}$, that is reflected from polygon P_i, one needs to consider only $P_j \in \Gamma_i$ against $\mathcal{R}$.

Assume that the ray tracer employs a spatial subdivision scheme such as an *octree* data structure. let Ω_l be the polygons' list of voxel l of the *octree*. On each entry of $\mathcal{R}$ into voxel l, for each $P_k \in \Omega_l$, an RSI test is conducted only if

$P_k \in \Gamma_i$. If $P_k \notin \Gamma_i$, P_k is invisible to P_i, and so P_k can not be hit by any ray reflected from P_i.

For shadow computation, a light-ray $\mathcal{R}$ needs to be traced between the given light source, and some point $p \in P_i$. If an intersection is detected before reaching P_i, point p is in shadow. The Γ_i list of the light source can improve this test, by testing first if the polygon exists in Γ_i of the current light source, and skip the ray traversal if the polygon is invisible, immediately deducing that the point on P_i is in shadow.

In summary of this acceleration step:

Algorithm 4 *Shadow Computation Using the VDS.*
For each point $p \in P_k$.
For all light sources $\mathcal{L}_i$ *do*
If $P_k \in \Gamma_i$ *then*
Trace a ray toward $\mathcal{L}_i$ *to test for occlusion of* p *from* $\mathcal{L}_i$*;*
else
p *is in shadow with respect to* $\mathcal{L}_i$*;*

4. Examples and Performances:

In this section, we present some results of ray tracing acceleration with the VDS. All the scenes were tested with one or two light sources and both tests where conducted with and without shadow generation.

We have considered three scenes. The first one is denoted the *Open* scene and contains a shelf with three teapots above it. The teapots are partially occluded by stands. Another teapot can be found below the shelf (see Figure 5). In a scene like this, one can expect a high degree of occlusion of polygons, and a moderate number of occlusions from the light sources. In this scene, a gain of up to 40% in ray tracing time have been observed due to the exploitation of the polygonal VDS, and only a few percent due to the light sources' VDS. This result is understandable considering that the light sources see most of the polygons in the scene, thus the VDS of the light sources is rarely beneficial. All the described results refer to the times of each of the occlusion stages separately.

The second scene is similar to the *Open* scene, except for the walls that were added around the scene in order to allow greater level for reflectance. In this scene, a gain of approximately 30% in ray tracing time have been observed due to the exploitation of the polygonal VDS. This closed scene can be seen at Figure 6.

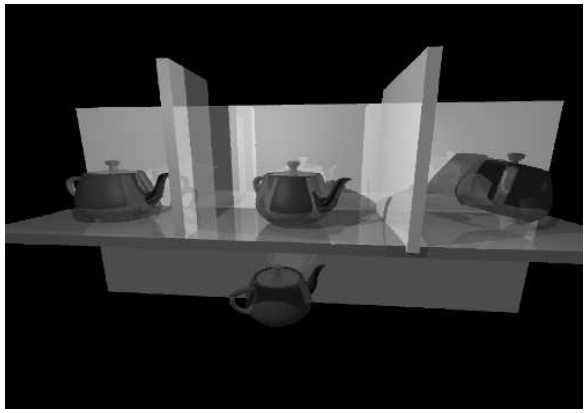

Figure 5. The *Open* scene of several Utah Teapots separated by stands

The third scene is denoted the *Room*, and is more general (see Figure 7). The gain in computation time is about 18% due to the exploitation of the polygonal VDS and only a few percent due to the light sources' VDS.

Table 1 presents the ray tracing times, while Table 2 shows the time of the pre-processing stage, and the average number of occluded polygons in the VDS.

One can think of models such as architectural buildings, where the acceleration will be much more significant due to the higher levels of occlusion. Even in the relatively simple scenes we have presented, the average occlusion of the polygons was above 90%.

5. Conclusions

We presented a method that can be used in static scenes toward the acceleration of photo-realism rendering based on the ray tracing algorithms. Reductions of 10% to 40% were found in practice.

The ray tracing application is one example for the use of this visibility paradigm and the VDS extension to the notion of a "model". Other applications in the field of computer graphics and modeling can employ the VDS just as well. For example, the *Radiosity* method can employ the VDS in order to reduce the amount of *elements* each *patch* is considering on it's *hemicube*, before getting to the rendering stage.

Scene	Number of Light Sources	Shadows	No VDS	Polygonal VDS (Relative Time)	Light Source VDS (Relative Time)	Combined VDS (Relative Time)
Closed	1	no	149.32	89.18 (60%)	-	-
Closed	2	no	153.96	95.41 (62%)	-	-
Closed	1	yes	352.73	295.62 (84%)	351.9 (100%)	294.8 (84%)
Closed	2	yes	724.15	665.63 (92%)	712.0 (98%)	660.3 (91%)
Open	2	no	41.82	28.97 (69%)	-	-
Open	2	yes	84.16	69.04 (82%)	79.45 (94%)	64.66 (77%)
Room	2	no	309.10	257.04 (83%)	-	-
Room	2	yes	432.43	379.41 (88%)	430.26 (99%)	376.96 (87%)

Table 1. Ray tracing timings. Shown here, are the results of the ray tracing stage on the three scenes. All the results are in seconds. All times were measured on a 166Mhz Pentium machine on top of the Windows NT 4.0 operating system.

Scene	Total Number of Polygons	Voxels	Polygonal VDS (invisibles)	time	Lights VDS (invisibles)	Time
Closed scene	8160	20^3	8115.3	416.9	5630	114.3
Open scene	8088	20^3	7903.8	319.24	5630	114.1
Room	3307	30^3	3072.4	974.23	939	18.25

Table 2. VDS computation: Presented herein are the computation time of the polygonal as well as the light sources' VDS. Also presented are the number of polygons that are found to be invisible, on the average, for each polygon and light source. All the scenes represented in the table contain one light source in the middle of the scene. All the results are in seconds. All times were measured on a 166Mhz Pentium machine on top of the Windows NT 4.0 operating system.

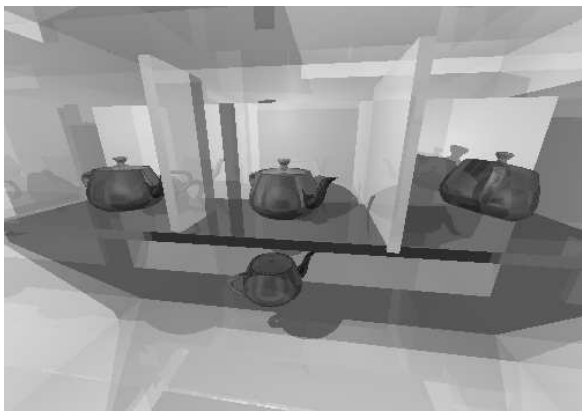

Figure 6. Similar scene to Figure 5 that is embedded in a reflective closed box. Denoted the *Closed* scene.

Figure 7. An image of the *Room* with two light sources.

For *NC-machining* verification, the VDS can also be used to more efficiently detect collisions with the moving tool.

The visibility analysis technique presented here is fairly simple. Never the less, it can be combined with other visibility analysis techniques such as *Cells and Portals* [14, 15, 16], thus reduce both the overhead in the computation time, and the size of the data structure of each polygon.

Finally, we would like to emphasize the importance of embedding a data structure similar to the VDS into com-

puterized geometry models. Much like color and texture attributes, the visibility data structure is an intrinsic property of the model that should be treated as such.

References

[1] H. Fuchs. On Visible Surface Generation by A-priori Tree Structure. *Computer Graphics* 14, pp. 124-133, 1980.

[2] A. Fujimoto, T. Tanaka, and K. Iwata. Accelerated Ray Tracing System. *Comp. Graph. and App.*, pp. 16-25, April 1986.

[3] T. A. Funkhouser and C. H. Sequin and Seth J. Teller. Management of Large Amounts of Data in Interactive Building Walkthroughs. *1992 Symposium on Interactive 3D Graphics*, pp. 11-20, March 1992.

[4] A. S. Glassner. *An Introduction to Ray Tracing*.

[5] A. S. Glassner. Space Subdivision for Fast Ray Tracing. *Computer Graphics and Applications IEE* 4(10), pp. 15-22, October 1984.

[6] T. L. Kay, and J. Kajiya. Ray Tracing Complex Scenes. *Computer Graphics* 20(4), pp. 269-278, August 1986.

[7] S. Rubin, and T. Whitted. A Three Dimensional Representation for Fast Rendering of Complex Scenes. *Com. Graph.* 14(3), pp. 110-116, July 1980.

[8] S. J. Teller and C. H. Sequin. Visibility Preprocessing for Interactive Walkthroughs. *SIGGRAPH '91 Proceedings*, pp. 61-69, July 1991.

[9] A. Watt. *3D Computer Graphics*, second edition.

[10] H. Weghorst, G. Hooper, and D. P. Greenberg. Improved Computational Methods for Ray Tracing. *ACM Trans. on Graph.* 3(1), pp. 52-69, 1984.

[11] T. Whitted. An Improved Illumination Model for Shaded Display. *Communication ACM* 23(6), pp. 343-349, June 1980.

[12] Krzystof S. Klimaszewski, and Thomas W. Sederberg. Faster Ray Tracing Using Adaptive Grid. *Computer Graphics and Applications IEEE*, pp. 42-51, January-February 1997.

[13] Frederic Cazals, George Drettakis, Claude Puech. Filtering, Clustering and Hierarchy Construction: a New Solution for Ray-Tracing Complex Scenes. *Computer Graphics Forum*, pp. 371-382, Volume 14, Number 3, 1995.

[14] David Luebke and Chris Georges. Simple, Fast Evaluation of Potentially Visible Sets. *1995 Symposium on Interactive 3D Graphics*, pp. 105-106, 1995.

[15] Thomas A. Funkhouser. Database Management for Interactive Display of Large Architectural Models. *Graphics Interface*, pp. 1-8, 1996.

[16] Seth Teller and Pat Hanrahan. Global Visibility Algorithms for Illumination Computations. *1993 Computer Graphics Proceedings*, pp. 239-246, 1993.