World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:9, No:10, 2015

Computing Visibility Subsets in an Orthogonal Polyhedron

Jefri Marzal, Hong Xie, Chun Che Fung

Abstract**—**Visibility problems are central to many computational geometry applications. One of the typical visibility problems is computing the view from a given point. In this paper, a linear time procedure is proposed to compute the visibility subsets from a corner of a rectangular prism in an orthogonal polyhedron. The proposed algorithm could be useful to solve classic 3D problems.

Keywords**—**Visibility, rectangular prism, orthogonal polyhedron.

I. Introduction

VISIBILITY is an important concept in computational geometry. Problems involving the visibility of objects have arisen in many fields, such as in computer graphics, VLSI layout, motion planning, and surveillance. Therefore, any improvement to visibility concept will have contributions to those fields and relevant applications. Typical visibility problems include computing the view from a given point, determining whether two objects partially see each other, mutual visibility of objects for lighting, and computing the umbra and penumbra [1]. In a given scene, two points are visible if the segment joining them does not intersect any obstacle in the scene. The study of visibility is thus intimately related to the study of free line segment in a scene. Visibility also can be defined in terms of graph, in which the nodes are the vertices of the scene, and an arc joins two vertices p and q if they are mutually visible [2].

The computational geometry research community has shown great interest in 2D visibility. For example, Pocchiola and Vegter provided a powerful and elegant framework to comprehend visibility properties in the plane, where visibility problems and coherence are expressed naturally [3]. Frequently the underlying structure of the visibilities is critical, and one types of structure for visibility is a graph. Gosh and Mount presented an optimal $O(k + n \log n)$ algorithm for computing the visibility graph of a polygonal scene, where n is the number of vertices and k is the number of arch of the visibility graph [4].

Unfortunately, theoretical literature is limited when 3D visibility is concerned. Researchers typically study the theoretical complexity of problems such as view computation or ray-shooting. The results available are often reductions of algorithms. Even in theory, the approach is mostly constructive and not analytical. Bygi and Ghodsi explained some reasons for this condition, and they tried to define geometry structure in 3D space [2].

This paper is concerned about a variant of the visibility problem that is defined as follows: *let an orthogonal polyhedron is decomposed into a set of rectangular prisms Π, and let c be a corner of a rectangular prism, then what is a subset of Π that is totally visible to c?*

In this study, a procedure is developed to compute the visibility subset from a corner point as defined above. Furthermore, the time complexity for executing the procedure is calculated, and it is found that the procedure works in linear time.

II. Definition and Terminology

A *polyhedral surface* is defined as a finite, connected set of plane polygons, such that every edge of each polygon belongs to just one other polygon, with the proviso that the polygons surrounding each vertex form a single circuit (to exclude anomalies such as two pyramids with a common apex) [5]. An edge that belongs to exactly two polygons is called *two-manifold edge*, and a vertex that is the apex of only one cone of polygons is called a *two manifold vertex* [6]. A *cone* is defined as a three-dimensional geometric shape that tapers smoothly from a base to a point called the *apex*. Hence, a polyhedral surface contains connected polygons that have only two-manifold edge and two-manifold vertex, and this kind of boundary is called a *two-manifold boundary*.

The polygons in a polyhedral surface are called *faces*, and the faces do not cross each other. A *polyhedron* is defined as a closed subset of a 3-dimensional Euclidean space whose boundary can be expressed as a polyhedral surface [5].

Faces, edges and vertices are important elements of a surface of a polyhedron. Their definitions are essential. A *face* is a polygon in a surface of a polyhedron. An *edge* is a line segment on its boundary where two or more faces meet. A *vertex* is a point on its boundary three or more edges meet [7].

A boundary of a polyhedron divides the space into two regions, one of which, called the *interior*, is finite. A *point of a polyhedron* is a point either on the boundary, or in the interior, of the polyhedron. A *point on the boundary*, or boundary point of a polyhedron, is a point on the boundary of the polyhedron. An *interior point* of a polyhedron is a point in the interior of the polyhedron [5].

One of the most widely studied classes of polyhedron is the orthogonal polyhedron. Tang defined an *orthogonal polyhedron* as a polyhedron in which every edge is parallel to one of the three orthogonal directions [8]. An orthogonal polyhedron is also called *isothetic polyhedron*.

J. Marzal is with the School of Education, University of Jambi, Indonesia (e-mail: jefri_marzal@unja.ac.id).

H. Xie and C. Fung are with the School of Engineering and Information Technology, Murdoch University, Western Australia (e-mail: H.Xie@murdoch.edu.au, l.fung@murdoch.edu.au).

To facilitate discussion in visibility subset, the following terms, some of which were introduced in [9], are defined.

- Definition 1. Two points x and y in an orthogonal polyhedron are said to be visible from each other if and only if the segment xy does not intersect the boundary of the orthogonal polyhedron.
- Definition 2. Let c be a point of an orthogonal polyhedron, the visibility region of c, denoted Vr(c), is the set of points of the orthogonal polyhedron that are visible from c.
- Definition 3. A piece ρ of an orthogonal polyhedron is said to be totally visible from c if every point of ρ is visible from c (i.e., $\rho \subseteq Vr(c)$. ρ is said to be partially visible from c if some, but not all, points of ρ is visible from c.

Generally, there are two type of data structures in polyhedra representation: edge-based data structure and vertex based data structure [7]. Aquilera and Ayala represented an orthogonal polyhedra by using extreme vertices only [10]. This method requires less number of vertices compare with other methods that involved all vertices. A brief review of some common features of this method is given below.

As stated by Juan-Arinyo [11], the number of incident edges at any vertex on an orthogonal polyhedron can be three, four, and six. They are labeled as *V3*, *V4* and *V6* to represent the degree of the vertex. *V3* means three edges meet at vertex the *V*, and similar meanings exist for *V4* and *V6* vertices.

A brink is the longest uninterrupted line segment, built out of a sequence of collinear and contiguous two-manifold edges of an orthogonal polyhedron *OP*. Every edge belongs to a brink and each brink has at least one edge. A brink may be contain *V3*, *V4* and *V6*, but the two ending vertices of a brink is *V3*. A *V4* and *V6* may only be an intermediate vertex of a brink. The ending vertices of all brinks in *OP* are called *extreme vertices*.

Extreme Vertices Model (EVM) is defined as a model that only stores all extreme vertices of an orthogonal polyhedron. The vertices of *OP* is accessed by ABC-Sorted EVM in which its extreme vertices are sorted first by coordinate *A*, then by *B*, and then by *C* [10].

In ABC-sorted EV model, vertices are arranged in a set of brinks. The k^{th} C-brinks has $v_b = v_{2k-1}$ as the beginning vertices and $v_e = v_{2k}$ as the ending vertices, for k=1, 2,.. be two consecutive vertices (*C*-brink refers to those brinks parallel to the *C*-axis).

In an ABC-sorted EVM array, the sequence of vertices can be viewed as a list of pairs of vertices starting from the first vertex in the array. The two vertices in each pair have the same coordinate values in the A-axis and the B-axis, but different coordinate values in the C-axis. These two vertices are actually the two end vertices of the same brink that parallel to the C-axis. Furthermore, the set of pairs of vertices in this ABC-sorted EVM array represents all brinks in the orthogonal polyhedron that are parallel to the C-axis.

The above method can be used to find all brinks that are parallel to Z-axis by sorting the EVM array into XYZ-sorted. The set of brinks parallel to Y-axis can be obtained by sorting the EVM array into XZY-sorted. Similarly the set of brinks parallel to the X-axis can be obtained by sorting the EVM array into YZX-sorted.

As the size of any ABC-sorted EVM array is same as that of the original EVM array, it is obvious that the number of brinks in an orthogonal polyhedron that are parallel to each of the three axes is the same, i.e., it is always a half of the extreme vertices.

Fig. 1 (a) is an example of a solid orthogonal polyhedron object. This object is represented by coordinates of its extreme vertices that may be inputted in any order, and the object can be reconstructed perfectly by the above EVM method. Fig. 1 (b) shows the order of the XYZ-sorted extreme vertices that were sorted by the *x*-coordinate first, followed by *y*-coordinate, and then followed by *z*-coordinate. By connecting the two extreme vertices in each pair, all vertical brinks (hence all vertical edges) would be reconstructed. The same procedure can be used to reconstruct all horizontal edges (parallel to the X-axis) and all back-front edges (parallel to Y-axis).

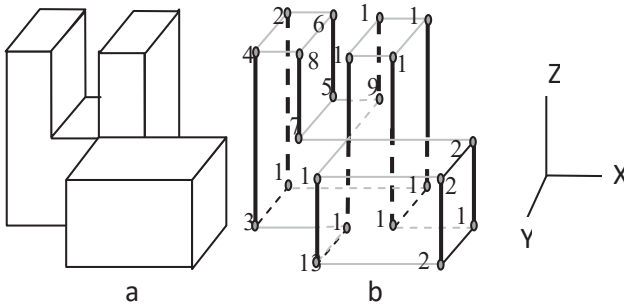

Fig. 1 Reconstruction of Orthogonal Polyhedron

III. Visibility in an Orthogonal Polyhedron

Suppose a set of rectangular prism Π is decomposed from an orthogonal polyhedron by an algorithm that was proposed in [12]. The purpose of computing the visibility subsets is to construct a collection non empty sets $S = \{ S_j \mid j=1,...,k \}$, where $S_j = \{\rho \mid \rho \in \Pi$ and $\rho \subseteq Vr(c_j)\}$ is the visibility subset for corner point c_j. Fig. 2 depicts an orthogonal polyhedron which is partitioned into a set of rectangular prisms Π. After the partitioning, v_1 is a vertex, and u_1 is an arbitrary point of the polyhedron. Both of v_1 and u_1 are also corner points of the same rectangular prism. A j^{th} corner point is symbolized as c_j. In addition, ρ_1 and ρ_2 are pieces in Π.

The procedure of computing the visibility subsets from a corner point of a rectangular prism rests on the following observation: i) each rectangular prism has six faces. These six faces can be divided into two types — a Type I face is also a face of the original orthogonal polyhedron, and a Type II face is completely made up of the interior points of the original orthogonal polyhedron except possibly at the edges of the face. If an edge of a face is also an edge on the original orthogonal polyhedron, the edge is said to be Type I edge. Otherwise the edge would consist of only interior points of the original orthogonal polyhedron and is called Type II edge. ii)

for a given view point, a rectangular prism (the first rectangular prism) is totally visible from the view point if and only if there exists no other rectangular prism with a Type I face intersecting the line connecting the view point and a point in the first rectangular prism.

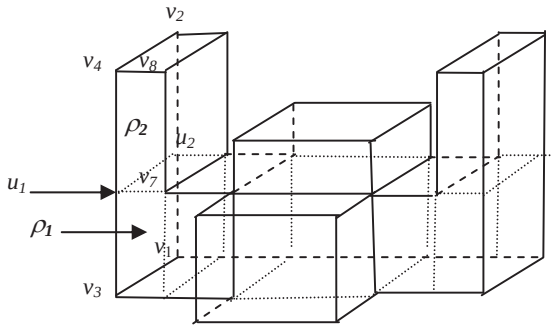

Fig. 2 Partitioning on an Orthogonal Polyhedron

```
for ( i = 1 to k ) {
    Si = ∅;
    for ( j = 1 to m ) {
        If (ci is a corner point of rpj) {
            Si = Si + { rpj };
            continue;
        }
        blocked = false;
        for (l = 1 to m ){
            if (l != j )
                blocked = IsViewBlocked (ci, rpj, rpl);
            if ( blocked )
                break;
        }
        if ( not blocked )
            Si = Si + { rpj };
    }
}
```

Fig. 3 Algorithm for Constructing Visibility Subsets

The algorithm in Fig. 3 assumes the availability of a corner point c_i (i=1, 2, . . . , k) from m rectangular prisms rp_j (j=1, 2, …, m) which are resulted from the partition of an orthogonal polyhedron. Note that some corner points are shared by more than one rectangular prism, therefore $k \leq 8m$. It attempts to construct k visibility subsets S_i (i=1, 2, . . . , k). For each corner point, it checks each rectangular prism to see whether it is completely visible from that corner point. If every point in the rectangular prism is visible from the corner point, the rectangular prism is said to be completely visible from the corner point. Otherwise, it is said to be (fully or partially) blocked from the corner point. At the end of the outmost loop, S_i would contain all rectangular prisms that are completely visible from corner point c_i. Fig. 3 illustrates the detailed algorithm,.

The function *IsViewBlocked*, as shown in Fig. 4, takes a corner point c_i, and two rectangular prisms, rp_j and rp_l. It returns true if the view from c_i to rp_j is blocked in anyway by the presence of rp_l. Otherwise, it returns false.

For any given rectangular prism rp_j and a point c_i lying outside of rp_j, there are between one and three faces of rp_j that are visible from c_i, depending on the position of the point relative to the rectangular prism. These visible faces and the corner point can form up to three rectangular pyramids, with each visible face at the base and the corner point at its apex. If another rectangular prism rp_l blocks the view from c_i to rp_j, whether fully or partially, it must contain at least one Type I face or Type I edge. Otherwise the rectangular prism would consist of only interior points of the original orthogonal polyhedron, it would be "transparent".

Rectangular prism rp_l blocks the view from c_i to rp_j if and only if rp_l contains a Type I face or Type I edge that intersects with one of the aforementioned rectangular pyramids. To see why this is a necessary condition, let's assume that rp_l does block the view from c_i to rp_j. This means that there exists at least one point s in rp_j that is blocked by rp_l. The line connecting c_i and s would intersect with one or more points of rp_l. One of these intersection points must lie on a Type I face or Type I edge, because otherwise all intersection points would be interior points of the original orthogonal polyhedron which are transparent and would not block the view. This proves that if rp_l blocks the view from c_i to rp_j, then rp_l must contain a Type I face or Type I edge that intersects with one of the rectangular pyramids. To see that the condition is also sufficient, we only need to take any intersection point s between the Type I face of rp_l and one of the rectangular pyramids. Since s lies in the pyramid, the line from c_i to s can be extended to the base of the pyramid, ending at point t. It is clear that point t on a face of rp_j is not visible from c_i because the sight is blocked by point s which is on a Type I face or Type I edge of rp_l. This means that rp_l blocks the view from c_i to rp_j.

To determine whether a rectangle and a rectangular pyramid intersect with each other, one can check whether any of the four corner points of the rectangle lies in the pyramid. If one is found to be inside the pyramid, the rectangle and pyramid intersect with each other. If none of the corner points lies inside the pyramid, we still need to consider the case when the rectangle cuts through the pyramid however all corner points are outside of the pyramid. This can be easily verified by taking each of the eight edges of the pyramid and see whether any one of the edge intersects with rectangle. If one edge is found be intersecting with the rectangle, the pyramid and the rectangle intersect with each other. Otherwise, they do not intersect with each other.

```
function IsViewBlocked (c_i, rp_j, rp_l)
{
    var edge, base, rectangle, pyramid;
    for ( base = each of the rectangular faces of rp_j that are visible from point c_i ) {
        if ( base is a Type I face )
            return true;
        pyramid = the rectangular pyramid formed by point c_i and rectangle base;
        for ( rectangle = each face of rp_l ) {
            if (rectangle is a Type I face)
                if (rectangle intersects with pyramid ){
                    return true;
                else
                    return false;
            for (edge = each of rectangle's Type I edges) {
                if (edge intersects with pyramid)
                    return true;
            }
        }
    }
    return false;
}
```

Fig. 4.Function `IsViewBlocked` for Blocking Determination

It is relatively easy to determine whether an edge intersects with a rectangular pyramid. Firstly one can check each of the two end points of the edge. If at least one of the end points is inside the pyramid, the edge must intersect with the pyramid. If both end points of the edge lie outside of the pyramid, there is still possibility that the edge intersects with the pyramid. It is noted that in such a scenario, the edge intersects with the pyramid if and only if the edge intersects with one of the five faces of the pyramid. Hence the intersection can be determined by checking whether the face of the pyramid intersect with the edge.

IV. Discussion

The time cost of the guard placement algorithm is calculated as following. Visibility subsets can be constructed in $O(m^3)$ time as shown in the previous section. It can be establish that $m < n^3$, hence the visibility subset can be constructed in polynomial time in n. This algorithm could be useful to solve some classic 3D problems such as the art gallery problem.

References

[1] F. d. Durand, *et al.*, "The 3D Visibility Complex," *ACM Transactions on Graphics,* vol. 21, pp. 176 - 206, 2002.
[2] M. N. Bygi and M. Ghodsi, "3D Visibility Graph," presented at the Computational Science and its Applications, Kuala Lampur, 2007.
[3] M. Pocchiola and G. Vegter, "Topologically sweeping visibility complexes via pseudotriangulations," *Discrete & Computational Geometry,* vol. 16, pp. 419–453, 1996.
[4] S. K. Ghosh and D. Mount, "An output sensitive algorithm for computing visibility graphs," *SIAM Journal Computing,* vol. 20, pp. 888-910, 1991.
[5] H. S. M. Coxeter, *Regular polytopes*. New York: Dover Publications, 1973.
[6] J. R. Rossignac and A. A. G. Requicha, "Constructive Non-Regularized Geometry," *Computer - Aided Design,* vol. 23, pp. 21-32, 1991.
[7] T. Biedl and B. Genc, "Reconstructing orthogonal polyhedra from putative vertex sets," *technical reports,* 2007.
[8] K. Tang and T. C. Woo, "Algorithmic aspects of alternating sum of volumes. Part 1: Data structure and difference operation," *Computer-Aided Design,* vol. 23, pp. 357-366, June 1991.
[9] A. P. Tomas, *et al.*, "On visibility problems in the plane - solving minimum vertex guard problems by successive approximation," in *the 9th Int. Symp. on Artif. Intel. and Math.*, 2006.
[10] A. Aquilera and D. Ayala, "Solving point and plane vs orthogonal polyhedra using the extreme vertices model (EVM)," presented at the The Sixth International Conference in Central Europe on Computer Graphics and Visualization'98, 1998.
[11] R. Juan-Arinyo, "Domain extension of isothetic polyhedra with minimal CSG representation," *Computer Graphics Forum,* vol. 5, pp. 281-293, 1995.
[12] J. Marzal, *et al.*, "Vertex Configurations and Their Relationship on Orthogonal Pseudo Polyhedra " *World Academy of Science, Engineering and Technology* pp. 1-8, 2011.