

THE THREE-DIMENSIONAL ART GALLERY PROBLEM AND ITS SOLUTIONS

Jefri Marzal

School of Information Technology
Murdoch University

**This thesis is presented for the degree of Doctor of Information Technology
School of Information Technology
Murdoch University
2012**

I declare that this thesis is my own account of my research and contain as its main content work which has not previously been submitted for a degree at any tertiary education institution.

Jefri Marzal

ABSTRACT

This thesis addressed the three-dimensional Art Gallery Problem (3D-AGP), a version of the art gallery problem, which aims to determine the number of guards required to cover the interior of a pseudo-polyhedron as well as the placement of these guards. This study exclusively focused on the version of the 3D-AGP in which the art gallery is modelled by an orthogonal pseudo-polyhedron, instead of a pseudo-polyhedron. An orthogonal pseudo-polyhedron provides a simple yet effective model for an art gallery because of the fact that most real-life buildings and art galleries are largely orthogonal in shape. Thus far, the existing solutions to the 3D-AGP employ mobile guards, in which each mobile guard is allowed to roam over an entire interior face or edge of a simple orthogonal polyhedron. In many real-world applications including the monitoring an art gallery, mobile guards are not always adequate. For instance, surveillance cameras are usually installed at fixed locations.

The guard placement method proposed in this thesis addresses such limitations. It uses fixed-point guards inside an orthogonal pseudo-polyhedron. This formulation of the art gallery problem is closer to that of the classical art gallery problem. The use of fixed-point guards also makes our method applicable to wider application areas. Furthermore, unlike the existing solutions which are only applicable to simple orthogonal polyhedra, our solution applies to orthogonal pseudo-polyhedra, which is a super-class of simple orthogonal polyhedron.

In this thesis, a general solution to the guard placement problem for 3D-AGP on any orthogonal pseudo-polyhedron has been presented. This method is the first solution known so far to fixed-point guard placement for orthogonal pseudo-polyhedron. Furthermore, it has been shown that the upper bound for the number of fixed-point guards required for covering any orthogonal polyhedron having n vertices is $\Omega(n^{3/2})$, which is the lowest upper bound known so far for the number of fixed-point guards for any orthogonal polyhedron.

This thesis also provides a new way to characterise the type of a vertex in any orthogonal pseudo-polyhedron and has conjectured a quantitative relationship between the numbers of vertices with different vertex configurations in any orthogonal pseudo-polyhedron. This conjecture, if proved to be true, will be useful for gaining insight into the structure of any orthogonal pseudo-polyhedron involved in many 3-dimensional computational geometrical problems. Finally the thesis has also described a new method for splitting orthogonal polygon

using a polyline and a new method for splitting an orthogonal polyhedron using a polyplane. These algorithms are useful in applications such as metal fabrication.

TABLE OF CONTENTS

ABSTRACT	iii
TABLE OF CONTENTS	v
ACKNOWLEDGEMENTS	viii
LIST OF PUBLICATIONS	ix
CONTRIBUTIONS OF THE THESIS	x
LIST OF FIGURES	xi
LIST OF TABLES	xiii
CHAPTER 1: THE ART GALLERY PROBLEMS	1
1.1 The Classical Art Gallery Problem	1
1.2 Three-Dimensional Art Gallery Problems	5
1.3 Aims and Significance of this Research	8
1.4 Research Objectives	9
1.5 Outcomes of the Research	10
1.6 Structure of the Thesis	11
CHAPTER 2: CONCEPTS AND TERMINOLOGY FOR POLYGONS AND POLYHEDRA	14
2.1 Polygon	14
2.1.1 Definitions and terminology	14
2.1.2 Decomposition of polygons	18
2.1.3 Optimisation issues in the decomposition of polygon	20
2.2 Polyhedron	21
2.2.1 Definitions and terminology	21
2.2.2 Orthogonal polyhedron	25
2.2.3 Decomposition of polyhedron	29
2.3 Data Representation for Polygon and Polyhedra	31
2.3.1 Data representation for an orthogonal polygon	31
2.3.2 Data representation for an orthogonal polyhedron	32
2.3.3 Data representation for a polyhedron	35
CHAPTER 3: VERTEX CONFIGURATIONS OF ORTHOGONAL PSEUDO-POLYHEDRA	
3.1 Introduction	38
3.2 Vertex Configuration Conjecture	39
3.3 The Number of Defferent Vertex Configurations in any OPP	40
3.4 Constructing Orthogonal Pseudo-Polyhedra	47
3.4.1 Joining operation	48

3.5	The Vertex Configurations Conjecture	52
3.6	Duality of Vertex Configurations	59
3.7	Summary	61

CHAPTER 4: SPLITTING OPERATIONS 62

4.1	Introduction	62
4.2	Splitting an Orthogonal Polygon Using a Polyline	64
4.2.1	Definition and Terminology	64
4.2.2	An algorithm of splitting an orthogonal polygon using a polyline	67
4.2.3	Combining the vertices of an orthogonal polygon and a polyline	68
4.2.4	Grouping vertices	70
4.2.5	Implementation of the algorithm	71
4.2.6	Time complexity analysis and discussion	72
4.3	Splitting an Orthogonal Polyhedron Using a Polyplane	72
4.3.1	Definition and terminology	73
4.3.2	An algorithm of splitting an orthogonal polyhedron using a polyplane	77
4.3.3	Combining vertices	77
4.3.4	Grouping vertices	79
4.3.5	Implementation of the algorithm	82
4.3.6	Time complexity analysis and discussion	83
4.4	Splitting Polyhedra in a Bounding Box	84
4.4.1	Definition and terminology	85
4.4.2	The algorithm for splitting polyhedra	87
4.4.3	Intersection line and plane	88
4.4.4	Computing a splitting plane	89
4.4.5	Calculating the intersection point on an edge of a polyhedron	91
4.4.6	Splitting a polyhedron into two polyhedra	92
4.4.7	Implementation of the algorithm	94
4.4.8	Time complexity analysis and discussion	96
4.5	Summary	97

CHAPTER 5: PLACEMENT OF FIXED-POINT GUARDS IN AN ORTHOGONAL PSEUDO-POLYHEDRON 98

5.1	Terminology and Related Research	100
5.2	The Fixed-Point Guard Placement Algorithm	103
5.3	Partitioning of an Orthogonal Pseudo-Polyhedron	104
5.4	Computing Visibility Subsets in an Orthogonal Pseudo-Polyhedron	108
5.5	Mapping 3D-AGP into Minimum Set Cover Problem	112
5.6	Solving the Minimum Set Cover Problem — An Example	113
5.6.1	Integer Linear Programming formulation of the MSC problem	115
5.7	The time Complexity of the Fixed-Point Guard Placement Algorithm	116
5.8	Reducing the Number of Guards	117
5.9	Summary	121

CHAPTER 6: AN UPPER BOUND ON THE NUMBER OF FIXED-POINT GUARDS FOR ORTHOGONAL POLYHEDRA 122

6.1	Introduction	122
-----	--------------	-----

6.2	Determining the Dominant Pieces	123
6.2.1	Around pieces of various types of vertex	124
6.2.2	Remote pieces	126
6.3	Reducing the Number of Input Data for the MSC Problem	127
6.4	Upper Bound of Fixed-Point Guards for an Orthogonal Polyhedron	128
6.4.1	Counting the number of dominant visible pieces	128
6.4.2	Counting the number of partially visible pieces	129
6.4.3	Relationship between the number of vertices and reflex vertices	133
6.5	Summary	134
CHAPTER 7: CONCLUSION AND FUTURE RESEARCH		135
7.1	Conclusion	135
7.2	Future Research	138
REFERENCES		140
APPENDIXES		145
Appendix 1 : The solution of 3D-AGP case (input as Table 5.1) by using dual simplex algorithm		145
Appendix 2: The solution of 3D-AGP case (input as Table 5.2) by using dual simplex algorithm		150

ACKNOWLEDGEMENTS

Foremost, I would like take this opportunity to express my sincere appreciation to my thesis supervisor Dr. Hong Xie for his guidance and encouragement throughout my Doctoral of Information Technology (DIT) study and research at Murdoch University. His guidance helped me in all the time of research and writing this thesis. I could not have imagined having a better supervisor for my DIT study. To him I owe too much.

I am also grateful to my thesis co-supervisor Associate Professor Chun Che Fung. He has always offered me a stimulating, supportive, and comfortable environment to grow.

I would like to thank all my friends in Murdoch University for creating a friendly and supportive environment. This includes Chareen, Ning, Boom, and Mohammed

This thesis would not have been possible without the financial support from Indonesian Government. I would like to express my gratefulness to Ms. Avianti Amir for her assistance.

Last but not the least, I would like to thank my wife Timang Sutera for her understanding and love, my daughter Puti Intan Mayang Sani, my son Arung Sulthan Pamuka, and my parents ayahanda Masril and ibunda Zuarda for supporting me spiritually throughout my life.

LIST OF PUBLICATIONS

The following papers have reported some of the results contained in this thesis.

Journal Papers:

- J1. Marzal, J., H. Xie, and C. C. Fung. (2011). "Vertex Configurations and Their Relationship on Orthogonal Pseudo Polyhedra" World Academy of Science, Engineering and Technology (77): 1-8.
- J2. Marzal, J., H. Xie, and C. C. Fung. (2012). "An Algorithm for Point-Guard Placement on an Orthogonal Polyhedron " International Journal of Computational Geometry and Applications (IJCGA). Submitted for review.

Conference proceedings:

- C1. Marzal, J. and H. Xie (2009). Guards placement in the art gallery problem. The 10th Postgraduate Electrical Engineering and Computing Symposium. Edith Cowan University, Perth, Australia on October 1, 2009
- C2. Marzal, J., H. Xie, and C. C. Fung. (2011). An Algorithm for Splitting an Orthogonal Polyhedron with a Polyplane. International Conference on Uncertainty Reasoning and Knowledge Engineering. Bali Indonesia on August 4-7, 2011
- C3. Marzal, J., H. Xie, and C. C. Fung. (2011). An Algorithm for Splitting Polyhedra in a Bounding Box from a View Point. International Technical Conference of IEEE Region 10, Bali Indonesia on November 21-24, 2011.
- C4. Marzal, J., H. Xie, and C. C. Fung. (2012). Computing Visibility Subset in an Orthogonal Polyhedron. International Conference on Computer Science, Information System & Communication Technologies (ICCSISCT 2012), Bangkok, Thailand on September 1-2, 2012. Accepted for oral presentation.

CONTRIBUTIONS OF THE THESIS

The contributions in this thesis are described below.

In this thesis, a general solution to the guard placement problem for 3D-AGP on any orthogonal pseudo-polyhedron has been presented. To our knowledge, this method is the first solution to fixed-point guard placement for orthogonal pseudo-polyhedron. This is documented in Chapter 5. Some of the results have been reported in the conference paper C4 and the journal paper J2.

Furthermore, this thesis has shown that the upper bound for the number of fixed-point guards required for covering any orthogonal polyhedron having n vertices is $\Omega(n^{3/2})$, which is the lowest upper bound known so far for the number of fixed-point guards for any orthogonal polyhedron. This result is documented in Chapter 6.

This thesis also provides a new way to characterise the type of vertex in any orthogonal pseudo-polyhedron and has conjectured a quantitative relationship between the numbers of vertices with different vertex configurations in any orthogonal pseudo-polyhedron. This result is documented in Chapter 3. Some of this work has been reported in the journal paper J1.

Finally the thesis has also described a new method for splitting orthogonal polygon using a polyline and a new method for splitting an orthogonal polyhedron using a polyplane. These algorithms are useful in applications such as metal fabrication. This is documented in Chapter 4 and part of the work has been published in conference papers C2. A new method for splitting polyhedra in a bounding box has been developed. It is also documented in Chapter 4 and it part of the work has been reported in paper C3.

LIST OF FIGURES

Figure 2.1:	(a) A Polygon and (b) a Polygon with Holes	16
Figure 2.2:	(a) An Orthogonal Polygon and (b) an Orthogonal Polygon with Holes	17
Figure 2.3:	Three Different Convexities on Orthogonal Polygons	18
Figure 2.4:	A Multi-shell Polyhedron	22
Figure 2.5:	A Polyhedron has a Hole	24
Figure 2.6:	Relationship between the Pseudo-Polyhedron Class and the other Classes	24
Figure 2.7:	(a) An Orthogonal Polyhedron and (b) an Orthogonal Pseudo-Polyhedron	25
Figure 2.8:	Number of Faces around a Vertex	26
Figure 2.9:	Different Dihedral Angles around a Vertex v	27
Figure 2.10:	A Degree-Three and a Non-Degree-Three Orthogonal Polyhedron	27
Figure 2.11:	The f Count in Euler Formula	28
Figure 2.12:	Relationship among the Orthogonal Pseudo-Polyhedron Class and other Classes	29
Figure. 2.13:	Number of Tetrahedra in Different Tetrahedralisations	30
Figure 2.14:	Reconstruction of an Orthogonal Polygon	32
Figure 2.15:	Reconstruction of Orthogonal Polyhedron	35
Figure 2.16:	Facet f_i is in the Right Side Oriented Edge Cycle	37
Figure 3.1:	Two Different Vertex Configurations	40
Figure 3.2:	Relative Position of Cubes: Adjacent (a), Diagonally Adjacent (b), Interstitial Cubes (c), 3-Consecutive Cubes (d), and 4-Consecutive Cubes (e)	41
Figure 3.3:	Four Possible Vertex Configuration for V3	46
Figure 3.4:	Four Possible Vertex Configurations for V4	46
Figure 3.5:	Two Possible Vertex Configurations for V5	46
Figure 3.6:	Six Possible Vertex Configurations for V6	47
Figure 3.7:	(a) An OPP. (b) XY Decomposition (4 boxes). (c) YX Decomposition (5 boxes)	48
Figure 3.8:	A tree Diagram of Vertex Configurations	51
Figure 3.9:	P is a Result of Joining $P1$ and $P2$	58
Figure 3.10:	A is a Vertex on the Inner Boundary of $P1$ and Outer Boundary of $P2$	60
Figure 4.1:	Steps to Get an L-Shape Piece	63
Figure 4.2:	An Orthogonal Polygon with a Polyline	65
Figure 4.3:	Polylines on orthogonal polygons	66
Figure 4.4:	One of the Two Intersection Vertices Lies on the Vertex of an Orthogonal Polygon	68
Figure 4.5:	Procedure <code>CombiningVertices</code> for Combining Vertices	69
Figure 4.6:	Algorithm for Grouping Vertices	71
Figure 4.7:	The Vertices of Orthogonal Polygon and a Polyline	71
Figure 4.8:	Valid (a) and not Valid (b) Instances of Polyplane	74
Figure 4.9:	An Orthogonal Polyhedron with a Polyplane	75
Figure 4.10:	Valid (a,b,c) and not Valid (d) Polyplanes in Orthogonal Polyhedra	76

Figure 4.11:	Splitting an Orthogonal Polyhedron with a Polyplane	78
Figure 4.12:	Procedure <code>CVOPolyhedronPolyplane</code> for Combining Vertices	79
Figure 4.13:	Procedure <code>GroupingVertices3D</code> for Grouping the Combined Vertices	81
Figure 4.14:	The Vertices of an Orthogonal Polyhedron after Splitting	82
Figure 4.15:	A Walk for Grouping Vertices	83
Figure 4.16:	One Edge of B is Shared by S in (a); There is no Edge of B Shared by S in (b) or (c)	86
Figure 4.7:	Primitive Procedures and Functions for Splitting in a Bounding Box	88
Figure 4.18:	Algorithm <code>ComputingSplittingPlane</code> for Computing a Splitting Plane	91
Figure 4.19:	Algorithm <code>IntersectionPoint</code> for Determining Intersection Points	92
Figure 4.20:	Grouping Polyhedra	93
Figure 4.22:	Illustration of Splitting Polyhedron in a Bounding Box	94
Figure 4.23:	The Resulting of Splitting Polyhedra	96
Figure 5.1:	(a) an OPP, (b) Partitioning of the OPP	101
Figure 5.2:	Algorithm <code>rectangPrism</code> for decomposing P into rectangular prisms	106
Figure 5.3:	Algorithm for Splitting an OPP into Two Halves	107
Figure 5.4:	Corner Points on an Orthogonal Pseudo-Polyhedron	108
Figure 5.5:	Algorithm for Constructing Visibility Subsets	109
Figure 5.6:	Function <code>IsViewBlocked</code> for Blocking Determination	110
Figure 5.7:	The Guards Position after Deploying Integer Linear Programming	116
Figure 5.8:	Partitioning in Pieces that are Visible by Sections	120
Figure 6.1:	Dominant Pieces Shared by the V_{31} Vertex	123
Figure 6.2:	Around Pieces and Remote Pieces	124
Figure 6.3:	Around Pieces of Vertices of Orthogonal Polyhedra	125
Figure 6.4:	Illustration of a Partially Visible Piece ρ Position in 3D (a) and 2D (b)	129
Figure 6.5:	The Partially Visible Pieces ρ_1 and ρ_2 Share Three Dents d_1 , d_2 and d_3	131
Figure 6.6 :	Illustration of Adjacent Pieces among Several Formats	131
Figure 6.7:	Cube of m where $m = 27$	132

LIST OF TABLES

Table 3.1:	Constructing OPPs Using at most Eight Cubes	43
Table 3.2:	Grouping of OPPs	45
Table 3.3:	Joining Operation on an OPP	49
Table 3.4:	Relationship among the Vertex Configurations on Simpler OPPs	54
Table 3.5:	Operations and their ΔR Value	58
Table 3.6:	Vertex Configurations and Their Duality	60
Table 5.1:	Corner Point and their Visibility Regions	112
Table 5.2:	Corner Points and their Visibility Region	120

CHAPTER 1

INTRODUCTION

1.1 The Classical Art Gallery Problem

Guarding the works of famous painters in art galleries is not an easy task as a work of art is desired by art lovers and coveted by criminals. Art galleries must constantly monitor their collections of art to guard against any unexpected actions by visitors, such as theft, vandalism, and destruction. Art works can be monitored by video cameras, which are usually hung from the ceiling. Images from these cameras are sent to TV screens in the security offices either located at the gallery or some remote management centers.

It is intuitive to think that the number of cameras used to monitor the art gallery should be kept as small as possible. The reason for this is not solely due to financial issues, but also because it is easier to monitor art gallery areas using fewer TV screens than many. On the other hand, art galleries cannot have too few cameras, because they may not cover all of the art gallery's interior. This raises an interesting question in computational geometry, which is usually referred to as the *Art Gallery Problem*: How many cameras do we need to guard a given gallery and how do we decide where to place them? This problem was first posed by Victor Klee to his students in 1973[1].

Although the art gallery problem was motivated by the needs of monitoring the art gallery, the problem posed by Victor Klee is a computational geometry problem that has much wide application than guarding an art gallery, such as computer graphic, databases, image processing, VLSI layout, and artwork analysis [2].

A gallery is, of course, in a three-dimensional space, but its floor plan may give us a lot of information to place the cameras. Therefore, traditionally, an art gallery is modelled as a simple polygon — that is, regions enclosed by a single closed polygonal chain that does not intersect itself [3]. A camera position in the gallery corresponds to a point in the polygon. A camera sees a point as long as the line of sight to the point lies totally inside of the polygon.

Much research has been done to solve the original art gallery problem and its many variations [4-6]. The first solution of the art gallery problem came in 1975 from Chavatal, who proved that $\lfloor n/3 \rfloor$ guards are occasionally necessary and always sufficient to cover an n -gon [1]. Three years later, Fisk gave an elegant proof of Chavatal's theorem by using the concept of triangulation and a three-colouring scheme [7]. Avis and Toussaint developed the $O(n \log n)$ algorithm for placing these $\lfloor n/3 \rfloor$ stationary guards [8]. This algorithm is bounded by two other $O(n \log n)$ algorithms: the triangulation of a simple polygon [9] and three-colouring of the triangulated polygon [8]. The time complexity of Avis and Toussaint's algorithm was further improved by Chazelle, who obtained a linear time triangulation algorithm [10], and by Kooshesh and Moret, who obtained a linear time three-colouring algorithm [11].

In the classic art gallery problem, an art gallery is represented by a simple polygon. Recently more attention was given to an important variation of the classic art gallery problem by restricting the simple polygon to be orthogonal. This is perhaps because most real buildings are largely orthogonal, and thus orthogonal polygons are better models for potential applications. Due to its simplicity, modelling an art gallery with a simple orthogonal polygon allow us to obtain more efficient algorithms and aesthetic results. An orthogonal polygon is a simple polygon whose edges are either horizontal or vertical. Khan, Klawe, and Kleitman showed that $\lfloor n/4 \rfloor$ guards are sufficient and sometimes necessary to cover any simple orthogonal polygon with n vertices [12]. O'Rourke later gave a completely different but

somewhat simpler proof of this result [13]. Edelsbrunner, O'Rourke, and Welzl devised an $O(n \log n)$ algorithm for placing these $\lfloor n/4 \rfloor$ guards [14]. The first $O(n)$ algorithm for placement of the vertex guards for monitoring the inside of an orthogonal polygon was obtained by Sack [15].

The floor plan of any art gallery may be modelled as a simple polygon. However, in a real-world art gallery, there may be obstructions inside the gallery. These obstructions are called *holes*. In 1995, Bjorling-Sachs and Souvaine established that $\lfloor (n+h)/3 \rfloor$ vertex guards are always sufficient and sometimes necessary to guard a polygon with n vertices and h holes [16].

There are several ways to place guards. The first type is the *vertex guard* where the position of any guard is restricted to a vertex of the polygon. The problem of determining the minimum number of vertex guards that can see an n -wall simply connected art gallery is shown to be NP-hard [17]. Meanwhile, Schuchardt and Hecker proved that the problem of determining the minimum number of vertex guards that see a simple orthogonal polygon is also NP-hard [18].

The second type of guard is the *point guard* where each guard can be placed anywhere in the polygon. Hoffmann, Kaufmann, and Kriegel proved that any polygon, possibly with holes, can be monitored by at most $\lfloor (n+h)/3 \rfloor$ point guards where n is the total number of vertices and h the number of its holes. Lee and Lin showed that the problem of determining the minimum number of point guards that can see the inside of a simple polygon is NP-hard [17]. Furthermore, they proved that $\lfloor n/4 \rfloor$ point guards are the exact bound for monitoring the inside of an orthogonal polygon with n vertices.

The third type of guard is the *edge guard*, where a guard is allowed to move along the edge of a polygon. O'Rourke showed that $\lfloor n/4 \rfloor$ edge guards are always sufficient and sometimes necessary to cover the polygon with n vertices [19]. The problem of determining the minimum number of edge guards in a simple polygon is also NP-hard [17]. Meanwhile, Bjorling-Sach showed that $\lfloor (3n+4)/16 \rfloor$ edge guards are always sufficient to guard any simple orthogonal polygon with n vertices. [20].

The fourth type is the *mobile guard* where a guard for a simple polygon is allowed to move along a sequence of closed line segments totally contained in the simple polygon [19]. O'Rourke showed that if the guards are permitted to patrol fixed interior line segments of a simple polygon with n vertices, then $\lfloor n/4 \rfloor$ guards are always sufficient and sometimes necessary for $n \geq 4$ [19]. Aggarwal in [21] proved that $\lfloor (3n+4)/16 \rfloor$ mobile guards are always sufficient and occasionally necessary to cover any simple orthogonal polygon with n vertices, and $\lfloor (3n+4h+4)/16 \rfloor$ mobile guards are always sufficient and occasionally necessary to guard the polygon with n vertices and h holes.

Since most of the minimum guard problems are NP-hard, the focus of the research community has been on developing heuristics and approximation methods for the problem. For example, Ghosh proposed an $O(n^5 \log n)$ time approximation algorithm to find a vertex guard set that is at most $O(\log n)$ times the minimum number of vertex guards needed to cover a polygon with or without holes and with n vertices [22].

Tomas, Bajuelos, and Marques proposed an approximation algorithm to find a vertex guard set, in which the main idea in their approach is that each interior piece of an orthogonal polygon must be totally visible by at least one guard. They proved that the difference between the minimum number of guards and their approximation is quite small, namely $(\log n)$ times, where n is the number of vertices of the polygon [23].

Amit, Mitchell, and Parker reported heuristics for computing a small set of point guards to cover a given polygon. They recommended three heuristics: *guarding quality*, *space* and *time*. The sets of guards obtained using heuristic approach were very satisfactory, and they were always either optimal or close to optimal [24]. A genetic algorithm was applied as an approximation algorithm, in which the average of the minimal number of vertex-guards needed to cover a simple polygon with n vertices was observed to be $n/6.48$ [25].

Recent studies have considered a number of variations of the original art gallery problem. Saleh proposed k -vertex guarding simple polygon, in which a polygon is called k -vertex guardable if there is a subset of vertices of the polygon such that each point in the polygon is seen by at least k -vertices in the subset of vertices [26]. He proved that $2n/3$ is needed for $k=2$ to see the inside of a simple polygon with n vertices. Fragoudakis addressed the problem of efficiently placing guards and paintings in an art gallery by introducing the finest visibility segmentation concept whose goal is to place paintings and guards in an art gallery in such a way that the total value of the guarded paintings is maximised [27]. Bajuelos estimated the maximum hidden vertex set in a polygon [28]. Rana proposed a technique to identify the minimal number of CCTV cameras with the most visual coverage of open spaces [29]. Carevelas considered the problem of monitoring a polygon where the edges of which are arcs of curves [30]. Epstein considered the problem of placing a small number of angle guards inside a polygon [31], and Toth studied the art gallery problem with guards whose range of vision is 180° [32].

1.2 Three-Dimensional Art Gallery Problems

Early studies of the art gallery problem used a polygonal region in the plane as the model of the art gallery. The plane polygon only models the floor outline of the art gallery. It does not always provide adequate information about the complex spatial structure of the building. In

many applications, knowledge of the spatial structure of the building is essential for deciding how the building should be monitored.

Therefore, it is necessary to consider the 3D structure of a building to determine the number of guards required to cover it as well as the placement of these guards. This thesis focused on the *three-dimensional art gallery problem* (3D-AGP), a version of the art gallery problem in which the art gallery is modelled by a pseudo-polyhedron. Hence, in this thesis the classical art gallery problem in which the art gallery is modelled by a polygon is called the two-dimensional art gallery problem (2D-AGP) to distinguish it from the 3D-AGP. Furthermore, as this thesis exclusively focuses on the version of the 3D-AGP in which the art gallery is modelled by an orthogonal pseudo-polyhedron instead of a pseudo-polyhedron, the term 3D-AGP is used to imply the three-dimensional art gallery problem in which the art gallery is modelled by an orthogonal pseudo-polyhedron, unless it is specially pointed out otherwise.

An orthogonal pseudo-polyhedron provides a simple yet effective model for an art gallery because of the fact that most real-life buildings and art galleries are largely orthogonal in shape. In addition, most applications are in a 3D environment (*e.g.*, art galleries, supermarkets, banks, sensor network areas, and robot motion planning); therefore, using an orthogonal pseudo-polyhedron to model the art gallery/building/structure is more desirable than using a plane polygon.

Work on the 3D-AGP is much less extensive than those on the 2D-AGP. In the last two decades, only a small number of studies were reported on some aspects of the 3D-AGP. For instance, Bose, Shermer, Taussaint, and Zhu considered using vertex guards to monitor the surface of a polyhedron. They proved that $\lfloor n/2 \rfloor$ vertex guards are always sufficient and some time necessary to see the surface of the polyhedron having n vertices. They also reported that $\lfloor (4n-4)/13 \rfloor$ edges guards, which are mobile moving guards along the edges, are some time

necessary to guard the surface of a polyhedron having n vertices [33]. Grunbaum and O'Rourke used vertex guards to see the exterior of a simple polyhedron, and stated that $\lfloor (2f-4)/3 \rfloor$ vertex guards are sometimes necessary and always sufficient to monitor the exterior of a convex polyhedron with f faces, for $f \geq 10$ [4].

One may assume that placing a guard at each and every vertex of a polyhedron would cover the interior of the polyhedron. This was proved not be the case by Seidel. He gave an example of a simple orthogonal polyhedron in which guards placed at every vertex of that polyhedron do not fully cover the interior of the polyhedron. He also noted that $\Omega(n^{3/2})$ guards are sufficient to monitor that special type of simple orthogonal polyhedron, where n is the number of vertices [4]. Based on the example given by Seidel, it can be concluded that vertex guards are not suitable for the 3D-AGP. The reason for this is that there could be some areas or points inside an orthogonal polyhedron that are not visible from any vertex.

Souvaine, Veroy, and Winslow recently introduced the *face guard*, which is a guard that roam over an entire interior face of a simple polyhedron. They used face guards to monitor the interior of a simple polyhedron and a simple orthogonal polyhedron. They also proved that $\lfloor f/6 \rfloor$ face guards are sufficient to monitor any simple orthogonal polyhedron with f faces. They also reported that $\lfloor f/2 \rfloor$ face guards are sufficient to guard any simple polyhedron with f faces [34]. However, there is no procedure to place these face guards on the interior of the simple orthogonal polyhedron was given by them.

Based on the above discussion, the 2D-AGP is well known, but very little is known about the 3D-AGP. Progress in 3D-AGP has been difficult because the 3D-AGP does not have a set of established tools such as a triangulation, the main tool used in the 2D-AGP. Although there is a large difference between the problems in two and three dimensions, the 3D-AGP is being actively studied, and researchers have proposed vertex guards and mobile guards for solving

the 3D-AGP. However, these types of guards have shortcomings: (i) guards posted at every vertex of a polyhedron can obviously cover the inside if any polyhedron were tetrahedralizable such that every tetrahedron can be monitored by one guard in the corner; however, not every polyhedron is tetrahedralizable. Hence, vertex guards cannot be applied to fully cover the interior of a polyhedron. (ii) Mobile guards such as an edge guard and a face guard can overcome the limitation of vertex guards; however, in real life a mobile guard is not always suitable. Most real-life guards need to be stationary and remain at fixed positions at all times. For example, in many supermarkets, banks, art galleries, and even in many public places, surveillance cameras are widely deployed to monitor an area. It is impractical and also too expensive to require these cameras to move around in order to cover an area.

Therefore, stationary guards are more suitable than mobile guards in these applications. Because of the limitation of the vertex guards, in this thesis, only stationary point guards will be considered. A stationary point guard is also called a point guard, or a fixed-point guard. It can be placed anywhere inside a pseudo-polyhedron including on the interior of a face, or an edge, or a vertex of the pseudo-polyhedron. Once placed inside a polyhedron, it will remain in the allocated point and will never change its position.

1.3 Aims and Significance of this Research

The primary aim of this thesis is to develop a guard placement algorithm to monitor the entire interior of an orthogonal pseudo-polyhedron using only fixed-point guards and to determine an upper bound for the number of fixed-point guards required to cover the interior of an orthogonal polyhedron.

Current work on the 3D-AGP faces some challenges such as the fact that vertex guards are not suitable to any simple orthogonal polyhedron and mobile guards are not adequate in many real-world applications. This thesis attempts to address these limitations.

First, in this thesis, only fixed-point guards will be used. This formulation of the 3D art gallery problem is much closer to that of the original art gallery problem, which only considered stationary point guards. The use of fixed-point guards also means that any solution we develop will have wider applications. Second, existing work focused on guard placement for a simple orthogonal polyhedron. Unfortunately, many real-world objects cannot be modelled by a simple orthogonal polyhedron. In our work, a broader class of 3D geometric model, i.e., orthogonal pseudo-polyhedron, will be used. The simple orthogonal polyhedron is only a small subset of orthogonal pseudo-polyhedron. Therefore, our solution based on orthogonal pseudo-polyhedron is expected to have more real-world applications than those based on simple orthogonal polyhedron. Third, we believe our method based on orthogonal pseudo-polyhedron is more amenable to orthogonal pseudo-polyhedron with multiple boundaries.

1.4 Research Objectives

The main goal of this thesis is to develop a method for computing a small set of guards and their placement to cover any orthogonal pseudo-polyhedron. To achieve the goal, detailed study of the nature of orthogonal pseudo-polyhedron must be carried. A number of basic operations involving an orthogonal pseudo-polyhedron need to be developed. The following is a list of work we are proposing to carry out during this study:

1. To investigate various properties of an orthogonal pseudo-polyhedron to see whether there are any intrinsic rules governing it. More specifically, we will investigate

different type of vertex configurations and their relationships in an orthogonal pseudo-polyhedron.

2. To investigate an effective way to split an orthogonal polygon, as well as an orthogonal polyhedron, and a polyhedron. The work on polygon splitting may lend us ideas for splitting polyhedron.
3. To investigate the ways to decompose an orthogonal pseudo-polyhedron into a set of simple and primitive 3D shapes, such as rectangular prisms. An effective method for decomposition of orthogonal pseudo-polyhedron will be critical in determining the guard placement.
4. To investigate the way to compute the set of rectangular prisms, or other primitive 3D shapes, that are visible from a given point inside an orthogonal pseudo-polyhedron. This procedure would be useful for the reduction of the number of guards required.
5. To develop methods for determining the number of fixed-point guards needed for monitoring an orthogonal pseudo-polyhedron and the procedures for placement of these fixed-point guards in the orthogonal pseudo-polyhedron.
6. To determine a non-trivial upper bound for the number of fixed-point guards required for monitoring an orthogonal polyhedron. Such an upper bound will have theoretical significance since no non-trivial upper bound is known for a general orthogonal polyhedron at the present.

1.5 Outcomes of the Research

1. A general solution to the guard placement problem for 3D-AGP on any orthogonal pseudo-polyhedron will be developed. To our knowledge, this method will be the first solutions to fixed-point guard placement for orthogonal pseudo-polyhedron.

2. An upper bound for the number of fixed-point guards required for covering any orthogonal polyhedron.
3. A new way to represent a vertex configuration in orthogonal pseudo-polyhedra and a conjecture a quantitative relationship between the numbers of vertices in different vertex configurations in any orthogonal pseudo-polyhedra. This innovative approach will be useful for representing related geometry objects and their computational aspects.
4. A new method for splitting orthogonal polygon using a polyline and a new method for splitting an orthogonal polyhedron using a polyplane. These algorithms will be useful in applications such as metal fabrication.

1.6 Structure of Thesis

This thesis consists of seven chapters and two appendixes.

Chapter 2 provides definitions and terminology for concepts and operations involving polygons and polyhedra. It also discusses the data structures for representing orthogonal polygons, and polyhedra in computer memory.

Chapter 3 introduces the concept of vertex configurations of orthogonal pseudo-polyhedra. It then shows that there are up to 16 different vertex configurations in any orthogonal pseudo-polyhedron. The chapter also discusses the quantitative relationship between different types of vertex configurations in an orthogonal pseudo-polyhedron, and proposes a conjecture called the *Vertex Configuration Conjecture*. A number of related topics are also discussed in the chapter. They include reconstructing the orthogonal pseudo-polyhedron after it has been decomposed. Finally, it discusses the duality of each vertex configuration.

Chapter 4 proposes procedures for splitting different geometry models. Splitting operations are used to separate an object into two halves. The chapter starts with a new splitting procedure for an orthogonal polygon using a polyline. Then, it presents a new algorithm for splitting an orthogonal polyhedron using a polyplane. Finally, it introduces a procedure for splitting polyhedra using a splitting plane that passes through a given view point. This procedure is used in Chapter 5.

Chapter 5 develops a new method for determining the number of fixed-point guards needed to monitor an orthogonal pseudo-polyhedron, as well as where to place these guards. After introducing the related terminology and the existing work, the chapter describes a new algorithm for calculating the positions of a set of fixed-point guard. This algorithm makes use of several basic operations such as partitioning an orthogonal pseudo-polyhedron, computing visibility subsets, and mapping the 3D-AGP into the minimum set cover (MSC) problem, each of which is explained separately in this chapter. Finally, a new method is proposed to reduce the number of guards required to cover the interior of an orthogonal pseudo-polyhedron.

Chapter 6 refines the algorithm presented in Chapter 5 for determining and placement of fixed-point guards for monitoring an orthogonal pseudo-polyhedron. In the refined algorithm, only orthogonal polyhedron, rather than orthogonal pseudo-polyhedron, is considered. The refined algorithm relies on a definition to determine the dominant pieces such that the number of data inputs for the MSC problem which is a component of that algorithm can be reduced. In this chapter, the dominant pieces around various types of vertex configurations in any orthogonal polyhedra are identified. Based of the identification of dominant pieces, a new, non-trivial, upper bound for the number of fixed-point guards required for monitoring the interior of an orthogonal polyhedron is derived.

Chapter 7 presents the conclusions of this thesis and discusses the future research directions.

CHAPTER 2

CONCEPTS AND TERMINOLOGY FOR POLYGONS AND POLYHEDRA

This chapter provides some basic concepts and terminology for polygons and polyhedra. They will be used for developing new concepts and procedures in later chapters. Some issues and related research on polygons and polyhedra are also discussed in this chapter.

2.1 Polygon

In elementary geometry, a *polytope* is a geometric object with flat sides, which exists in any general number of dimensions. For example, a *polygon* is a polytope in two dimensions, while a *polyhedron* is a polytope in three dimensions. Polygons and polyhedra are the most popular polytopes because they are widely used models for many real world objects.

2.1.1 Definitions and terminology

A polygon is one of the basic concepts in computational geometry. A polygon is defined using its boundary, which is called a polygonal curve. A *polygonal curve* consists of a series of line segments, s_1, s_2, \dots, s_n . Each line segment s_i has two end points known as the starting point and the end point. These line segments are connected in the following way: for any two consecutive line segments s_i and $s_{(i+1)}$, the end point of s_i is connected to the starting point of $s_{(i+1)}$ for $1 \leq i < n$. A *polygonal curve* is said to be *simple* if, apart from the aforementioned intersections between consecutive line segments, there are no other intersections between the line segments of the polygonal curve. A *polygonal curve* is *closed* if the end point of its last line segment s_n is connected to the starting point of its first line segment s_1 . A *simple closed polygonal curve* is a polygonal curve that is both simple and closed. A *polygon* is defined as a

closed and bounded region of a plane whose boundary is a simple closed polygonal curve. [3].

From these definitions, there are clearly two basic components in a polygon. They are the vertex and the edge. A *vertex* of a polygon is a point on its boundary in which the boundary changes its slope, and an *edge* is a line segment on the polygon's boundary that connects two vertices. Two vertices p and q are *adjacent* to each other if they are connected by an edge e . In such a case, e is *incident* to vertex p and vertex q , and p and q are *incident* to e .

A polygon divides the 2-dimensional plane into two disjoint regions: the *interior region* which is finite and *exterior region* which is infinite. The simple closed polygonal curve forms the boundary that separates the two regions. A point is said to be *inside* a polygon if that point is located in the interior region of the polygon, or on its boundary. A point is said to be *outside* of a polygon if it is not located inside polygon. An area is said to be inside a polygon if all of its points are located inside the polygon [35].

A *polygon with holes* is defined as a shape that consists of one large polygon and one or more smaller polygons that are located completely inside the large polygon (but they do not intersect with the boundary of the large polygon), and these smaller polygons neither intersect with each other nor overlap with each other [3]. For a polygon with holes, there are two boundaries. The *inner boundary* is the polygonal curves of the smaller polygons, while the *outer boundary* is the polygonal curve of the large polygon in a polygon with holes. The following figures depict a polygon and a polygon with holes.

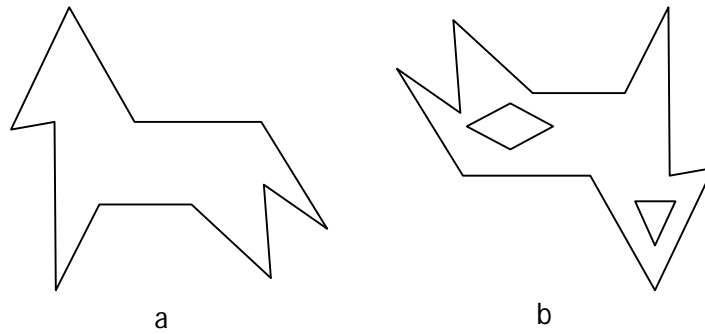


Figure 2.1: (a) A Polygon and (b) a Polygon with Holes

Every polygon has as many corners as it has sides, and each corner has several kinds of angles. The two most important types of angles are the interior angle and the exterior angle. An *interior angle* is the angle between two sides on the interior of a polygon. The *exterior angle* is the complementary angle to an interior angle, so the sum of the interior and exterior angles at any vertex must be 360° . A corner is said to be *convex* if its interior angle is less than 180° [36].

Convexity is an important concept in a polygon. A subset S of the plane is *convex* if and only if for any pair of points $p, q \in S$, the line segment \overline{pq} is completely contained in S . A *convex polygon* is a polygon with a convex interior. A polygon is convex if each corner is convex. A polygon that is not convex is called a *concave polygon* which has at least one interior angle greater than 180° [3].

Many real-world objects, such as books, tables, and rooms, have a rectangular shape. In many applications, these objects can be modelled as orthogonal polygons. An *orthogonal polygon* is a polygon with boundary sides parallel to the axes of the 2-dimensional Cartesian coordinate system. Clearly, the interior angle of any corner of an orthogonal polygon is either 90° (convex) or 270° (concave) [37]. If all angles of a polygon are either 90° or 270° , but the edges are not parallel to any axis, then it is called a *rotated orthogonal polygon* [38].

Orthogonal polygons are also known as *rectilinear polygons*. An orthogonal *polygon with holes* is defined as a shape that consists of one large orthogonal polygon and one or more smaller orthogonal polygons that are located completely inside the large orthogonal polygon in which the smaller orthogonal polygons neither intersect with the large orthogonal polygon, nor intersect with each other, nor overlap with each other. The following figures depict an orthogonal polygon and an orthogonal polygon with holes.

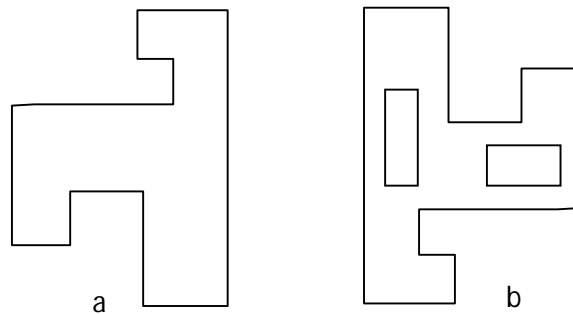


Figure 2.2: (a) An Orthogonal Polygon and (b) an Orthogonal Polygon with Holes

An *inflection vertex* of an orthogonal polygon is a concave vertex where the interior angle is 270° . For any orthogonal polygon having n vertices, the number of inflection vertices i is:

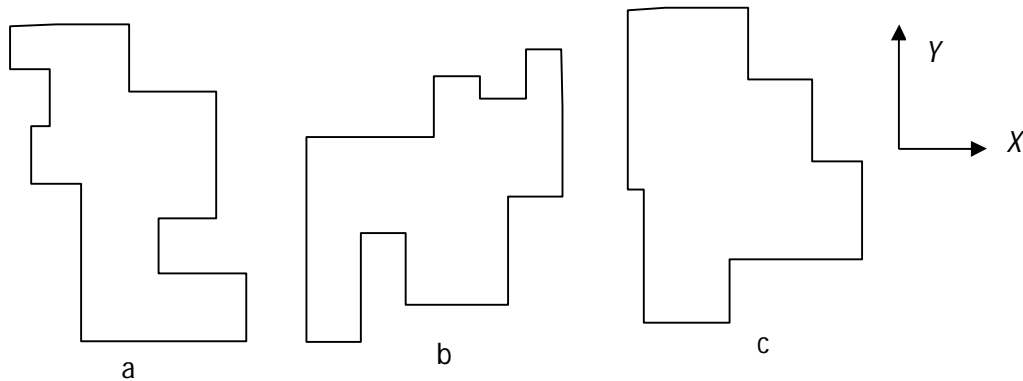
$$i = \frac{(n-4)}{2} \quad [39].$$

The only orthogonal polygon that is also convex is a rectangle; all other orthogonal polygons are concave. Therefore, the term “convexity” has a slightly different meaning when it is used to describe an orthogonal polygon. One way to define the convexity of an orthogonal polygon is by restricting the points when testing the convexity. In the context of an orthogonal polygon, a line is called *orthogonal* if it is parallel to one of the coordinate axes. A line segment is orthogonal if its two end points lie on an orthogonal line.

An orthogonal polygon is called *horizontally (vertically) convex* if its intersection with every horizontal (vertical) line is no more than one line segment. Meanwhile, an orthogonal

polygon is called *orthogonally convex* if it is both horizontally and vertically convex [40].

For examples, see Figure 2.3



a) horizontally convex orthogonal polygon, b) vertically convex orthogonal polygon, and c) orthogonally convex orthogonal polygon

Figure 2.3: Three Different Convexity on Orthogonal Polygons

The relationship between vertices is well known for orthogonal polygons. For an orthogonal polygon, the relationship between the number of convex vertices (HC) and the number of inflection vertices (HR) is: $HC - HR = 4$ [39]. Meanwhile, Voss classified the orthogonal polygon with holes boundary into the inner or outer boundary based on the relationship between the concave and convex vertices in the 2D digital image [41].

2.1.2 Decomposition of polygons

The task of polygon decomposition is to represent a polygon as the union of a number of simpler component parts. Polygon decomposition has many theoretical and practical applications. For examples, Taussaint employs polygon decomposition as tool for pattern recognition [42]; polygon decomposition is also useful for VLSI design, in which the layout is represented by a polygon, and one preparation approach for electron-beam lithography is by decomposing the polygon region into basic figures [43]. In computational geometry, algorithms for problems in general polygons are often more complex than those for restricted

types of polygon, such as the rectangle, star shape, or convex polygon. An example of polygon decomposition is partitioning an orthogonal polygon into fat rectangles [44].

Polygon decompositions are classified according to how the component parts interrelate. A decomposition is called a *partition* if it divides a polygon into a set of simpler polygons that do not intersect with each other, except on their boundaries. Meanwhile, if overlapping pieces are allowed, then the decomposition is called a *cover*.

Triangulation is a decomposition operation that decomposes a polygon into a set of non-overlapping triangles. The triangulation of a polygon results in a set of the diagonals of the polygon that divides the polygon into non-overlapping triangles (a polygon with three sides). A *diagonal* is a line segment that connects two vertices of a polygon and lies in the interior of the polygon. In a triangulation, diagonals do not intersect with each other, except at their endpoints. The sides of triangles produced by a triangulation are either diagonals, or sides, of the triangulation or sides of the original polygon. Every triangulation of an n -vertex convex polygon has $n-3$ diagonals [45]. Furthermore, Berg *et al.* [3] stated that every polygon admits a triangulation, and any triangulation of a polygon with n vertices consists of exactly $n-2$ triangles.

An orthogonal polygon can be partitioned in several ways such as with quadrilateralisation and rectangularisation. Quadrilateralisation is the partitioning of a given orthogonal polygon into a set of non-overlapping quadrilaterals. The number of quadrilaterals is $(n-2)/2$, where n is the number of vertices on an orthogonal polygon [12]. Meanwhile, rectangularisation is the partitioning of an orthogonal polygon into the minimum number of rectangles.

In some applications, such as in VLSI design, an orthogonal polygon has to be partitioned into rectangles. O'Rourke and Tewari proposed a polynomial-time algorithm for partitioning

an orthogonal polygon into fat rectangles, so that the shortest rectangle side is maximised over all rectangles [44].

2.1.3 Optimization issues in the decomposition of polygon

Many problems in computational geometry are related to the optimisation of some aspect of polygons and orthogonal polygons. In most applications, a polygon is decomposed that is minimal in some sense. Some applications seek to decompose a polygon into the minimum number of some basic components, and other applications seek to decompose a polygon into a minimal total length of internal edges. This section will focus on the partitioning problems for orthogonal polygons. These issues are relevant in the following chapters of this thesis.

Rectangle is the most important basic shape to consider in relation to the partitioning of orthogonal polygons. One of such issues concerns the partitioning of an orthogonal polygon into the minimum number of rectangles.

The minimum rectangular partition problem, defined for an orthogonal polygon, can be stated as follows: given an orthogonal polygon on the plane, find a minimally sized set of non-overlapping rectangles, such that every rectangle is contained in the orthogonal polygon and the union of all rectangles is equal to the original orthogonal polygon.

Ku and Leong provided a solution for the minimum rectangular partition problem [46]. However, their formula and its proof are very complicated. Nguyen simplified the formula for a minimum rectangular partition in which a given orthogonal polygon can be minimally partitioned into $i - c - k + 1$ rectangles, where i is the number of inflection vertices, c is the number of chords and k is the number of holes. A *chord* is a cutting line that has a reflection vertex at its two endpoints [47].

Liou, Tan and Lee proposed an $O(n \log \log n)$ algorithm for minimal rectangular partition of an orthogonal polygon, where n is the number of vertices in the polygon [48]. Lopez and Mehta proposed an algorithm for decomposing a polygon into a set of non-overlapping L-shapes and rectangles by using only horizontal cuts. They reported that the algorithm has $O(n + h \log h)$ time, where n is the number of vertices in the polygon and h is the number of H-pairs. Because the parameter h is small in VLSI design, this algorithm is close to linear in n in practice [49].

2.2 Polyhedron

2.2.1 Definitions and terminology

Polyhedron is an extension of the polygon into the three dimensional space. A polyhedron is used to represent a solid object. Using a similar approach to the one for defining polygon, our definition of polyhedron also starts by defining polyhedron's boundary known as polyhedral surface.

A *polyhedral surface* is defined as a finite, connected set of flat polygons or polygons with holes, such that every edge of each polygon or polygons with holes belongs also to just one other polygon or polygons with holes, with the proviso that the polygons or polygons with holes surrounding each vertex form a single circuit (to exclude anomalies such as two pyramids with a common apex) [50]. An edge that belongs to exactly two polygons or polygons with holes is called *two-manifold edge*, and a vertex that is the apex of only one cone of polygons or polygons with holes is called a *two manifold vertex* [51]. 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 a set of connected polygons or polygons with holes that have only two-manifold edges and two-manifold vertices. This kind of surface

is called a *two-manifold surface*.

The polygons in a polyhedral surface are called *faces*, and these faces do not cross each other. A *polyhedron* is defined as a subset of the 3-dimensional Euclidean space whose boundary is a polyhedral surface [50].

In addition to faces, a polyhedral surface also contains edges and vertices. In a polyhedral surface an *edge* is a line segment where two or more faces meet, while a *vertex* is a point where three or more edges meet.

The boundary of a polyhedron divides the space into two regions, one of which, called the *interior region*, is finite, and the other one, which is called the *exterior region*, is infinite. A point is said to be *inside* a polyhedron if that point is located in the interior region of the polyhedron, or on its boundary. A point is said to be outside of a polyhedron if it is not located inside the polyhedron. An area is said to be inside a polyhedron if all of its points are located inside the polyhedron [50].

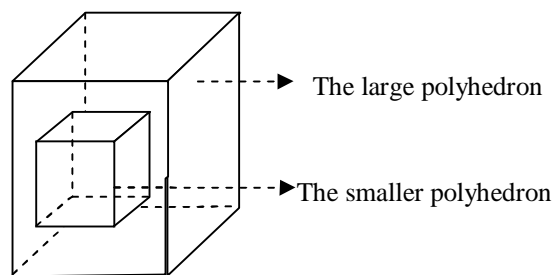


Figure 2.4: A Multi-shell Polyhedron

The polyhedral surface in a polyhedron is also called the *shell* of that polyhedron [36]. A *multi-shell polyhedron* is a solid shape that consists of one large polyhedron and one or more smaller polyhedra that are located completely inside the large polyhedron in which these smaller polyhedra neither intersect with the large polyhedron, nor intersect with each other, nor overlap with each other. Two kinds of boundary exist in a multi-shell polyhedron: the

inner boundary and the outer boundary. The *inner boundary* of the multi-shell polyhedron is the polyhedral surfaces of all the smaller polyhedra, while the *outer boundary* is the polyhedral surface of the large polyhedron [39]. Figure 2.4 depicts a multi-shell polyhedron.

It is possible for a solid object to have a surface with edges that are shared by at least two faces or with vertices that are the apex of more than one cone of faces. An edge that belongs to more than two faces is called a *non-manifold edge*, and a vertex that is the apex of more than one cone of polygons is called a *two manifold vertex* [51].

The existence of the non-manifold edges and non-manifold vertices gives rise to another type of surface, which is called pseudo-polyhedral surface, that is similar to the polyhedral surface, but with some differences. A *pseudo-polyhedral surface* that is a finite, connected set of flat polygons or polygons with holes, such that (a) every edge belongs to at least two polygons or polygons with holes, and (b) if any two polygons or polygons with holes meet, they meet at a common edge [52]. However, there is a possibility that two polygons or polygons with holes meet at a common vertex rather than a common edge. To include this scenario, in this thesis, the definition of the pseudo-polyhedral surface is extended by modifying condition (b) in the above definition: if two polygons or polygons with holes meet, they meet either at a common edge or at a common vertex. With this extended definition, a pseudo-polyhedral surface may have non-manifold edges, as well as non-manifold vertices. A *pseudo-polyhedron* is a subset of the 3-dimensional Euclidean space whose boundary is a pseudo-polyhedral surface.

A *simple polyhedron* is a polyhedron that can be deformed into a solid sphere; that is, a polyhedron that, unlike a torus, has no holes [53]. The polyhedron in Figure 2.5 cannot be deformed into a solid sphere, therefore it is not a simple polyhedron. A simple polyhedron is also called as a polyhedron with genus 0, and it must satisfy Euler's formula, in which the

relationship among the number of vertices v , edges e and faces f must satisfy the following equation: $v - e + f = 2$ [36].

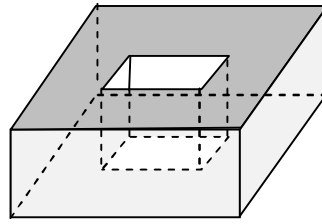


Figure 2.5: A Polyhedron has a Hole

There are two kinds of angles in a polyhedron: facial angles and dihedral angles. Two edges incident to a common vertex may be on the same face. In such case, the angle between the two edges on the same face is referred to as the *facial angle* of the face. The *dihedral angle* is the interior angle between two faces meeting at a common edge [54]. Furthermore, Wenninger defined that a polyhedron is *convex* if no dihedral angle is greater than 180° [55], otherwise the polyhedron is concave.

To conclude this sub-section, the following Venn diagram shows the relationship among the pseudo-polyhedron class and the other classes in which $SP \subset P \subset PP$ and $PP \cap MSP = \emptyset$.

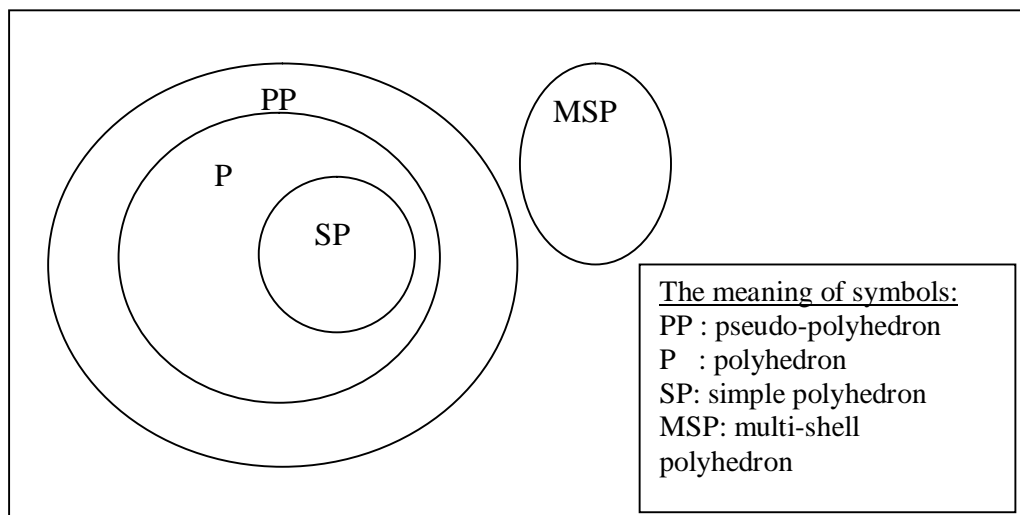


Figure 2.6: Relationship between the Pseudo-Polyhedron Class and the other Classes

2.2.2 Orthogonal polyhedron

The focus of this thesis is orthogonal polyhedron and orthogonal pseudo-polyhedron, which are used to represent art galleries.

An *orthogonal pseudo-polyhedron* is a pseudo-polyhedron in which every edge is parallel to one of the three orthogonal directions. In an orthogonal pseudo-polyhedron, a non-manifold edge is adjacent to exactly four faces and a non-manifold vertex is the apex of exactly two corners [56].

One of the most widely studied classes of pseudo-polyhedra 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 [52]. An orthogonal polyhedron is also called *isothetic polyhedron*. All facial and dihedral angles in an orthogonal polyhedron are either 90° or 270° .

The following figures show an orthogonal polyhedron and an orthogonal pseudo-polyhedron. The shape in Figure 2.7(a) satisfies the definition of a polyhedron, but the shape in Figure 2.7(b) does not satisfy the condition that every edge is shared by exactly two faces, and this shape only satisfies the definition of an orthogonal pseudo-polyhedron.

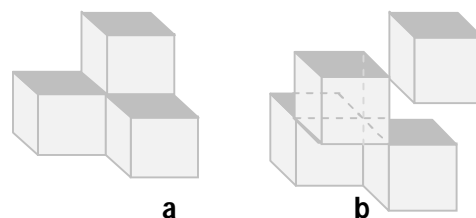


Figure 2.7: (a) An Orthogonal Polyhedron, and (b) an Orthogonal Pseudo-Polyhedron

A *multi-shell orthogonal polyhedron* is a solid shape that consists of one large orthogonal polyhedron and one or more smaller orthogonal polyhedra that are located completely inside

the large orthogonal polyhedron in which these smaller orthogonal polyhedra neither intersect with the large orthogonal polyhedron, nor intersect with each other, nor overlap over each other.

The *degree of a vertex* is the number of edges that meet at a vertex of an orthogonal polyhedron. If all the vertices have the same degree, the orthogonal polyhedron is regular. A rectangular prism is an example of a regular orthogonal polyhedron because each vertex has a degree of three. The degree of vertex is useful in determining the label type of each vertex on an orthogonal polyhedron.

As stated above, the edges and faces of an orthogonal polyhedron are oriented in three orthogonal directions. Juan-Arinyo noted that the number of incident edges for any vertex in an orthogonal polyhedron is either three, four or six [57]. He also gave two possible configurations of three edges meeting at a vertex, as well as one configuration for each of four and six edges meeting at a vertex as depicted in Figure 2.8. Vertex v has three edges in Figure 2.8 (a) and (b), four edges in Figure 2.8(c), and six edges in Figure 2.8(d).

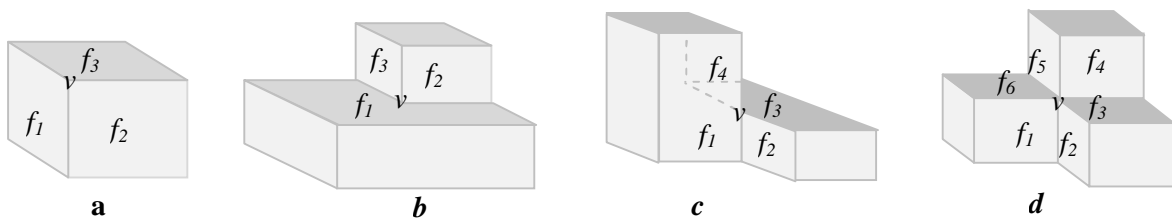


Figure 2.8: Number of Faces around a Vertex

In addition to the above four configurations, Yip and Klette found another two configurations for three edges meeting at a vertex as shown in Figure 2.9 [39], where Figure 2.9(a) has one 270° and two 90° interior dihedral angles, and Figure 2.9(b) has three 270° interior dihedral angles. Therefore, there are six vertex configurations on orthogonal polyhedra.

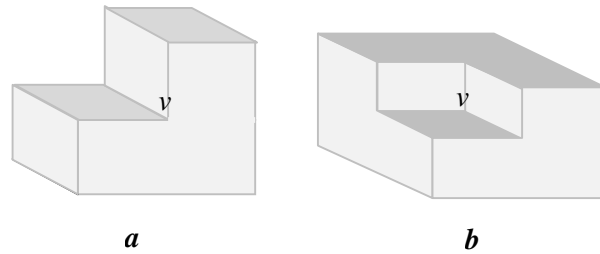


Figure 2.9: Different Dihedral Angles around a Vertex v

Orthogonal polyhedra can be grouped based on their degree of vertex. There is a kind of orthogonal polyhedra in which every vertex has degree of three and has exactly three mutually-perpendicular axis-parallel edges meeting at each vertex [58]. Figure 2.10(a) shows an example of an orthogonal polyhedron in which the vertices do not all have a degree of three (*e.g.*, vertex v has four edges). Figure 2.10(b) is an orthogonal polyhedron that has the degree of three for all its vertices.

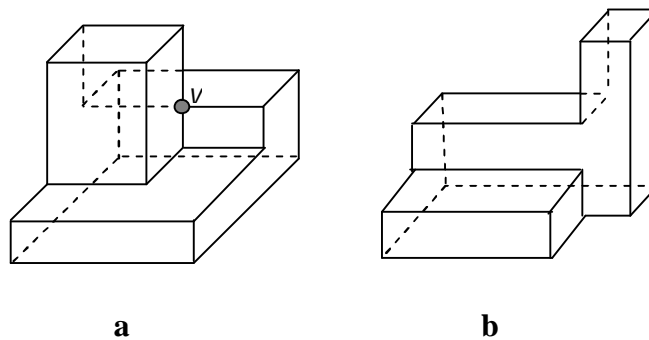
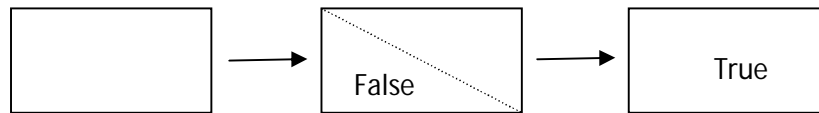


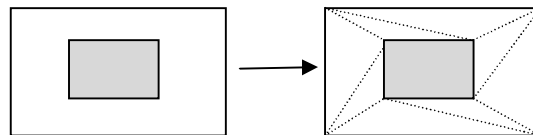
Figure 2.10: A Degree-Three and a Non-Degree-Three Orthogonal Polyhedron

An orthogonal polyhedron is called a *simple orthogonal polyhedron* if the polyhedron is both orthogonal and simple. As with a simple polyhedron, a simple orthogonal polyhedron satisfies Euler's formula, which states that the relationship among the number of vertices v , edges e and faces f satisfies the equation: $v - e + f = 2$. For the f count in the Euler formula, a flat polygon in a polyhedral surface can be counted as one face. When dealing with a polygon

with holes in a polyhedral surface, the polygon with holes requires triangulation. The number of triangles in the polygon with holes is counted as the number of faces for the f count in the Euler formula. [36]. The following are examples for the f count in the Euler formula for two different types of polygon in a polyhedral surface.



a) A flat polygon does not require triangulation, therefore $f=1$



b) A polygon with a hole requires triangulation, therefore $f=8$

Figure 2.11: The f Count in Euler Formula

The concept of orthogonal convexity is not only applicable to orthogonal polygons, but it can also be extended to orthogonal polyhedra. A simple orthogonal polyhedron is called *horizontally (vertically, frontally) convex* if its intersection with every horizontal (vertical, frontal) plane is either empty or a single orthogonally convex polygon. Meanwhile, a simple orthogonal polyhedron is called *orthogonally convex polyhedron* if it is horizontally, vertically and frontally convex. [59].

To conclude this sub-section, the following Venn diagram shows the relationship among the orthogonal pseudo-polyhedron class with the other classes in which $OCP \subset SOP \subset OP \subset OPP$ and $OPP \cap OPMS = \emptyset$

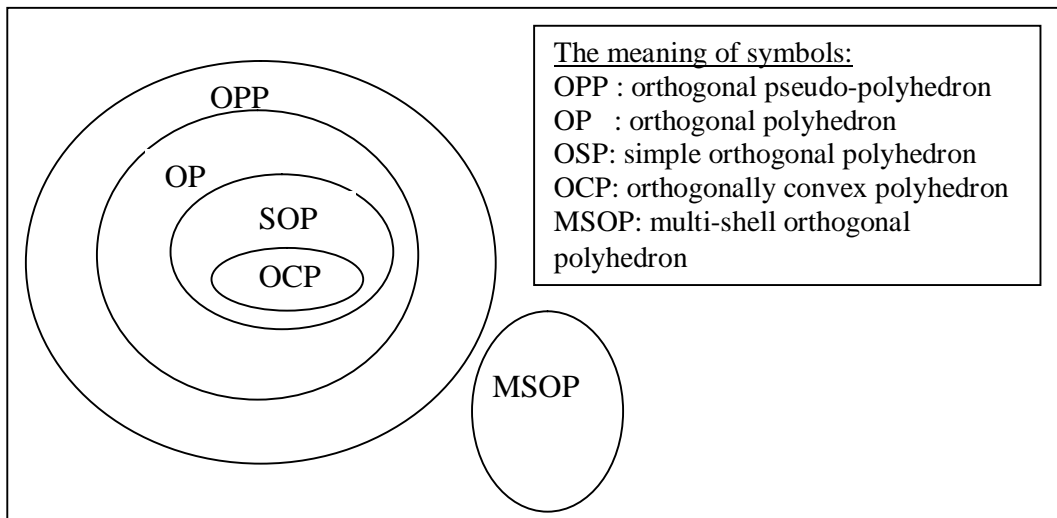


Figure 2.12: Relationship among the Orthogonal Pseudo-Polyhedron Class and other Classes

2.2.3 Decomposition of polyhedron

Polyhedron decomposition is a problem of dividing a polyhedron into a set of simpler forms of polyhedra, such as tetrahedron (*i.e.*, a pyramid based on a triangle). Sometimes, only vertices from the original polyhedron may be used as vertices of the sub-polyhedra. The problem of partitioning a polyhedron into a number of tetrahedra is called the *tetrahedralisation* of the polyhedron. Two neighbouring tetrahedra share a face, which is the triangle defined by the three shared vertices of the two polyhedra [60].

Research on tetrahedralisation began in the early twentieth century. It is now known that all convex polyhedra are tetrahedralisable, but not all non-convex polyhedra can be tetrahedralised [61]. The problem of optimal tetrahedralisation is finding a tetrahedralisation of a polyhedron with the minimum number of tetrahedra. Ruppert and Seidel showed that the three-dimensional tetrahedralisation problem is significantly more difficult than the two-dimensional triangulation problem. [62]. They also reported that one difference between the two problems lies in the size of the resulting partitions: triangulating every n -sided polygon

produces exactly $n-2$ triangles, but the number of tetrahedra in a tetrahedralisation of a given polyhedron varies.

For example, a bi-pyramid may be partitioned into two groups A and B , in which each group has either two or three tetrahedra (see Figure 2.13 below).

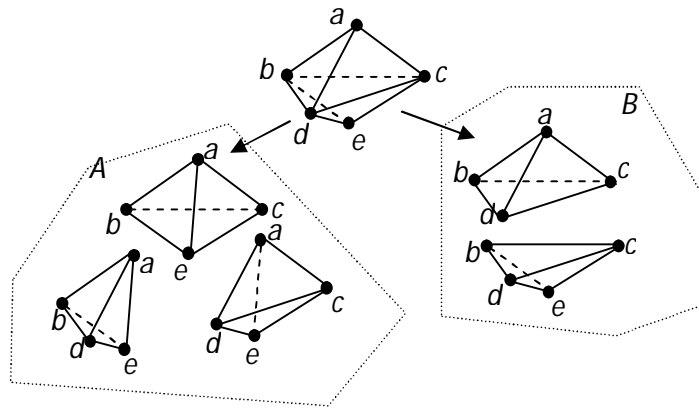


Figure. 2.13: Number of Tetrahedra in Different Tetrahedralisations

The problem of finding the minimum tetrahedralisation of a convex polyhedron is known to be NP-complete, and the number of tetrahedra from the tetrahedralisation of a convex polyhedron can be decreased if Steiner points are allowed [63]. Chen, Hsich and Wang presented a genetic algorithm for finding the minimum tetrahedralisation of a convex polyhedron. The result showed that the genetic approach obtains the optimum solution for point sets for which the optimum is known [64].

An orthogonal polyhedron can be partitioned into rectangular prisms. However, the problem of finding the minimum rectangular partition of an orthogonal polyhedron where the number of resultant rectangular prism is minimal is NP-complete [65], and Stolee reported that the scheme used for finding the minimum rectangular partition for an orthogonal polygon cannot be applied to solving the minimum rectangular partition problem for an orthogonal polyhedron. [66].

2.3 Data Representation for Polygons and Polyhedra

In computational geometry, raw data from a geometric model needs to be stored in computer memory in such a way as to make the subsequent computation more efficient. The way data are stored is called the data structure [67]. One way to represent an orthogonal polygon or orthogonal polyhedron in computer memory is by storing the coordinates of each of its vertices.

2.3.1 Data representation for an orthogonal polygon

This section will discuss how to represent an orthogonal polygon in computer memory. O'Rourke described a method for representing an orthogonal polygon [68] by using the vertices of the polygon. This method is discussed below.

In an orthogonal polygon, each edge has two end vertices, and each vertex is incident to exactly one horizontal edge and one vertical edge. For an orthogonal polygon with n vertices, the horizontal edges and vertical edges can be constructed as follows. Let v_1, v_2, \dots, v_m be the group of vertices that have the same y -coordinate and are sorted increasingly by the x -coordinates, then for this group of vertices, a set of edges can be constructed by connecting the vertices v_i and v_{i+1} , where i is an odd number. Repeat this process for every set of vertices with the same y -coordinate until all horizontal edges are obtained.

To get all vertical edges, repeat the above process by grouping the vertices that have the same x -coordinate and sorted them in increasing order by their y -coordinates. By applying this method, only one orthogonal polygon can be constructed out of any given set of vertices. For example, the vertices v_1, v_2, v_3, v_4, v_5 , and v_6 in Figure 2.14 are a group of vertices that have the same y -coordinate and are sorted increasingly by x -coordinates. The edges v_1v_2, v_3v_4, v_5v_6 are constructed by connecting the vertex v_i and v_{i+1} where $i = \{1, 3, 5\}$,

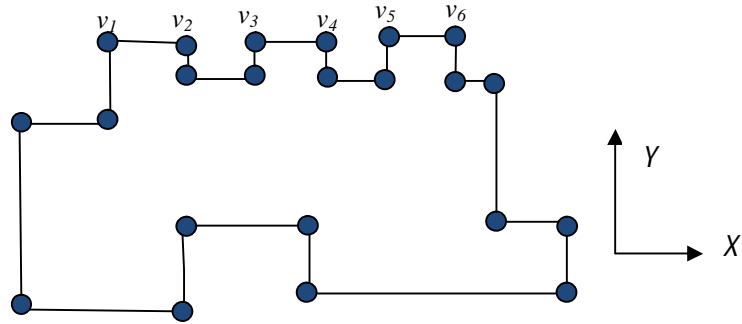


Figure 2.13: Reconstruction of an Orthogonal Polygon

2.3.2 Data representation for an orthogonal polyhedron

Generally speaking, there are two types of data structure for polyhedra, an edge-based and a vertex-based data structure. The edge-based data structure uses the edges of a polyhedron as the input to construct the polyhedron. The following methods use the edge-based data structure: winged-edge [69], half-edge [70], and quad-edge [71]. Preparata and Shamos introduced a standard representation of polyhedra by using doubly-connected edge list and vertex coordinates [37] as a variant of the winged-edge method, but this representation contains a lot of redundancy when applied to an orthogonal polyhedron. Bournez, Maler and Pnueli showed that this representation is ambiguous [72].

Aquilera and Ayala represented an orthogonal polyhedron by using only *extreme vertices* [73]. Their method requires a lower number of vertices compared with other methods that involve all the vertices. An overview of this method is given below.

As stated by Juan-Arinyo [57], the number of incident edges at any vertex on an orthogonal polyhedron can be three, four, or six. Such a vertex is called V3, V4 or V6 type of vertex, respectively. V3 means three edges meet at the vertex; the meaning of V4 and V6 are similar.

A *brink* is the longest uninterrupted line segment, built out of a sequence of collinear and contiguous edges from an orthogonal polyhedron. In an orthogonal polyhedron, every edge belongs to a brink and each brink contains at least one edge. A vertex in a brink can be V3, V4 or V6 type, but the two ending vertices of a brink is always V3 type. V4 and V6 types of vertices may only appear as interior vertices of a brink. An ending vertex of a brink in an orthogonal polyhedron is called *extreme vertex*.

The extreme vertices model (EVM), which was proposed by Aquilera and Ayala [73] represents an orthogonal polyhedron only using its extreme vertices. The extreme vertex is stored in array data structure known as the *EVM array*. Each element of the EVM array contains coordinates of a single extreme vertex. The array elements in an EVM array can be sorted lexicographically with the order $\langle a, b, c \rangle$, where x is the coordinated in Axis A, y is the coordinate in Axis B, and z is the coordinate in Axis C. The resulting array is called ABC-sorted EVM array.

For example, let $v_a = (x_a, y_a, z_a)$ and $v_b = (x_b, y_b, z_b)$ be two vertices, the YZX-sorting use the following rule:

$$\begin{aligned}
 v_a < v_b \text{ if and only if} \\
 & \text{Either } v_a < v_b \\
 & \text{Or } v_a = v_b \text{ and } z_a < z_b \\
 & \text{Or } v_a = v_b \text{ and } z_a = z_b \text{ and } x_a < x_b
 \end{aligned}$$

The resulting array is called YZX-sorted EVM array.

Depending on the order of the three Cartesian coordinates in the sorting, an EVM array can be sorted into the following six orders. They are XYZ-sorted, XZY-sorted, YXZ-sorted, YZX-sorted, ZXY-sorted and ZYX.

A *plane of vertices* of an orthogonal polyhedron is the set of extreme vertices lying on a plane perpendicular to one of the three axes. There are three directions of plane of vertices. They

are XY-plane, XZ-plane and YZ-plane of vertices. For generality, they are written as AB-plane of vertices where A and B represent two axes. An AB-plane of vertices consists of all vertices of the orthogonal polyhedron lying on the same plane that is parallel to both axis A and axis B. These vertices have the same coordinate in the C-axis (the axis other than A or B). For example, an XY-plane of vertices consists of all vertices of the orthogonal polyhedron with the same Z coordinate.

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.

Figure 2.15(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. Figure 2.15(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).

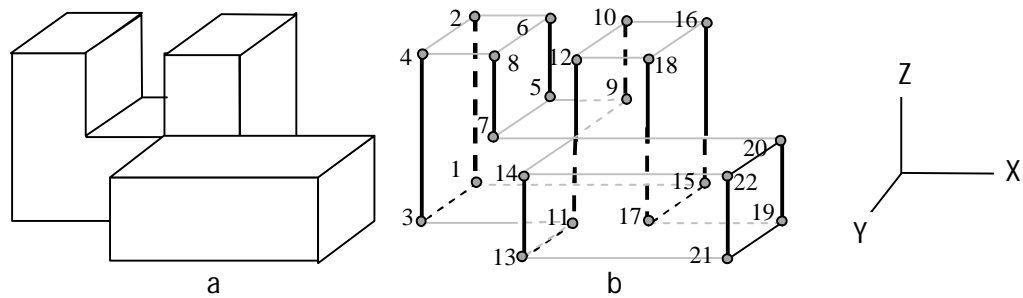


Figure 2.15: Reconstruction of Orthogonal Polyhedron

The concept of extreme vertices model in orthogonal polyhedron can be extended to orthogonal pseudo-polyhedron [56].

Representing an orthogonal polyhedron by its extreme vertices is better than representing it with all vertices. First, the number extreme vertices is smaller than the number of vertices for any orthogonal polyhedron, hence the EVM method requires less input data and therefore less memory requirement. Second, the coordinates of non-extreme vertices can be obtained from the intersection points of brinks.

2.3.3 Data structure for a polyhedron

A polyhedron can be represented by a collection of vertices, edges and facets, and this data structure is called star-edge representation [74], where a facet is a terminology for a face. Bajaj and Dey proposed the similar data structure to the star-edge representation [75]. The star-edge representation uses the following four components.

a. Vertex

A vertex is the corner of a polyhedron in which three or more faces intersect at the corner. Each vertex will be represented by two fields: the *coordinates of the vertex*, which is the position of the vertex in the three-dimensional Cartesian system; and the *adjacent edges*, which contain pointers to the edges incident to the vertex.

b. Edge

An edge is a line segment that connects two vertices and has two adjacent faces. Each edge is represented by fields: *adjacent vertices*, which contain pointers to the two end vertices, and *edge orientation*, which contains pointers to the structure that is called *Orientededges*.

c. Orientededges

The orientation of an edge on a facet f is such that a traversal of the oriented edge has facet f to its right. The orientation of an edge is recognized by several fields: *edge*; *facet*, which contains pointer to the facet on which the orientededge is incident; *orientation*, which contains information of the orientation of the edge on the facet; and *nextorientededge*, which contains pointer to the next orientededge on the oriented edge cycle of a facet as described below.

d. Facet

A facet is a polygon in a polyhedron surface. Each facet has two fields. They are the *facet equation*, which contains the equation of the plane on which the facet lies, and the *facet cycle*, which contains the oriented cycle bounding the facet. The traversal of the oriented edge cycle always has the facet to the right.

Polyhedron in Figure 2.16 consists of six vertices, nine edges, and five facets. The facet f_1 has the oriented edge cycle as e_2 , e_5 , e_6 , and e_7 . The edge e_2 has v_2 and v_3 as its ending vertices and f_1 as one of its incident facets.

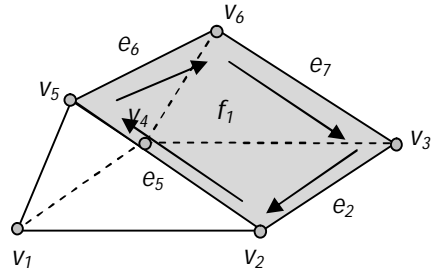


Figure 2.16: Facet f_1 is in the Right Side Oriented Edge Cycle

CHAPTER 3

VERTEX CONFIGURATIONS OF ORTHOGONAL PSEUDO-POLYHEDRA

3.1 Introduction

This chapter focused on vertex configuration in orthogonal pseudo-polyhedra (OPP). The chapter will present the following results: first, it will be shown that there are no more than 16 different vertex configurations for any orthogonal pseudo-polyhedra. Second, a method for decomposing an orthogonal pseudo-polyhedron into a set of rectangular prism is described, and it will be proved that for any OPP, there exists a set of rectangular prisms such that the OPP can be constructed by combining these rectangular prisms together. Finally, a conjecture is presented that describe a quantitative relationship between the numbers of vertices of different vertex configurations in any OPP.

The main contributions of this chapter are: a new way to represent a vertex configuration in orthogonal pseudo-polyhedra and a vertex configuration conjecture. Understanding the vertex configurations in OPP and their relationship among each other could provide insight into the structure of an OPP and be useful when designing algorithms for many 3-dimensional (3D) geometrical problems.

This chapter is organized into several sections. Section 3.2 gives definitions and terminology that are useful for discussing the topic of vertex configurations. Section 3.3 shows a proof that there are no more than 16 vertex configurations for any OPP. Section 3.4 provide an overview a technique for decomposing an OPP into a set of rectangular prisms. This section also demonstrates joining operations between various OPPs with a rectangular prism. Section

3.5 presents a formula to show the relationship among the number of each type of vertices in any OPP and gives some evidences for this formula. Finally, the last section will identify the dual vertex of each vertex configuration in multi-shell OPPs.

3.2 Vertex Configuration

Recall that an orthogonal pseudo-polyhedron (OPP) is a pseudo polyhedron in which every edge is parallel to one of the three orthogonal directions. In many practical applications, an OPP provides a simple yet effective approximation to many important geometrical objects. The use of an OPP arises frequently in practice, such as creating models of buildings that are largely orthogonal shaped. Even though it was stated in the previous chapter that orthogonal polyhedra are used to represent art galleries, in fact, not all galleries can be represented by orthogonal polyhedra, such as a gallery that has edges from four walls. Therefore, an orthogonal pseudo-polyhedron (OPP) is often required for such situation.

Like its sub-class, orthogonal polyhedron, an OPP has many applications in such areas as connected component labeling [76] and pattern analysis in digital images and VLSI layout [39]. Often they are studied with respect to partitioning problems [56] and visibility problem [77].

In an orthogonal polyhedron, any two adjacent faces form an interior dihedral angle and there are only two possible values for such a dihedral angle. Recall that either the angle is 90° , which is called the *convex dihedral angle*, or the angle is 270° , which is called *concave dihedral angle*. However, in an OPP, two adjacent faces may not always be capable of forming a dihedral angle due to the presence of a non-manifold edge or non-manifold vertex.

The *vertex configuration* of a vertex is defined by the number of adjacent edges, concave dihedral angles, and non-manifold components meeting at that vertex. Each type of vertex

configuration is represented by a unique label. For example, the vertex configuration, which has five edges, four concave dihedral angles, one non-manifold edge, and which is not non-manifold vertex, is labelled V54-10. For any manifold vertex, its label can be shortened to three digits. For instance, the aforementioned vertex can also be labelled as V54-1 instead of V54-10.

As examples, vertex v_1 in Figure 3.1(a) has four edges meet at the vertex, one concave dihedral angle, and two non-manifold edges; hence, the vertex configuration of v_1 is labelled V41-2. Vertex v_2 in Figure 3.1(b) has six edges meet at the vertex, three concave dihedral angles, and zero non-manifold edges; hence, the vertex configuration of v_2 is labelled V63-0 or V63.

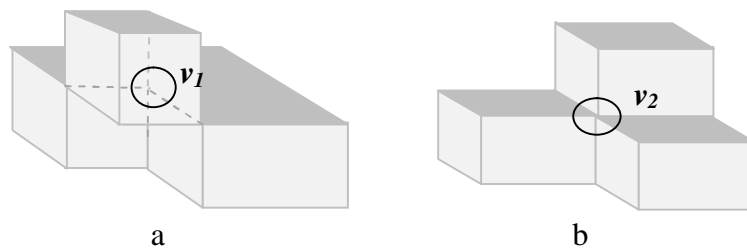


Figure 3.1: Two Different Vertex Configurations

Special for orthogonal polyhedra, Yip and Klette reported that there are only six different types of vertex configurations [39]. They also stated and proved a formula about count of these types of vertex in orthogonal polyhedra. However, very little is known about vertex configurations in OPP and their relationship with each other.

3.3 The Number of Different Vertex Configurations in any OPP

In this section it will be proved that there are no more than 16 different vertex configurations for any OPP. In order to prove the theorem, some terms and concepts need to be introduced first. A cube is the simplest OPP. Two OPPs are said to be *congruent* if both OPPs can be

oriented in such a way that they have the same shape and the same size. Imagine that a cube is divided into eight congruent cubes c_1, c_2, \dots, c_8 . The following are terminology for relationship among cubes: (i) A cube c_1 is *adjacent* to another cube c_2 , if c_1 and c_2 share one face (see Figure 3.2 (a)). (ii) c_1 is *diagonally adjacent* to c_2 if they share one edge (see Figure 3.2 (b)). (iii) Two cubes are said to be an *interstitial cubes* if they only share one vertex (see Figure 3.2 (c)). (iv) Three cubes c_1, c_2, c_3 are said to be *3-consecutive cubes* if a cube is adjacent to two other cubes and all three cubes share one common edge (see Figure 3.2 (d)). (v) Four cubes c_1, c_2, c_3, c_4 are said to be *4-consecutive cubes* if each cube is adjacent to two other cubes and all four cubes share one common edge (see Figure 3.2 (e)).

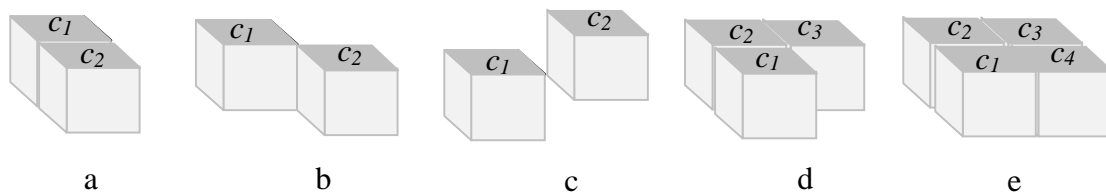


Figure 3.2: Relative Position of Cubes: Adjacent (a), Diagonally Adjacent (b), Interstitial Cubes (c), 3-Consecutive Cubes (d), and 4-Consecutive Cubes (e)

Several premises, $PR1$, $PR2$, $PR3$, $PR4$, and $PR5$ are made to describe similarities between two OPPs. The premises and their proof are listed as follows:

Premise PR1: Two adjacent cubes are congruent with any two other adjacent cubes.

Premise PR2: Two diagonally adjacent cubes are congruent with any two other diagonally adjacent cubes.

Premise PR3: Two interstitial cubes are congruent with any other two interstitial cubes.

Premise PR4: 3-consecutive cubes are congruent with any other 3-consecutive cubes.

Premise PR5: 4-consecutive cubes are congruent with any other 4-consecutive cubes.

The validity of the above five premises are obvious. For example, given any two 3-consecutive cubes, P1 and P2, no matter how they are positioned in the 3D space initially, no

matter how they are positioned in the 3D space initially, one can always re-orient one of the two, so that both look the same (and also have the same size).

In an OPP, each edge is parallel to one of the three orthogonal directions. Therefore, for each vertex in an OPP, there are at most six distinct incident edges. At the same time, there must be at least three incident edges for each vertex to be 3D. Hence the number of incident edges of any OPP vertex ranges from three to six. In this thesis, an OPP vertex with n incident edges is denoted as V_n ($n = 3, 4, 5, 6$), and from now on, these vertices are referred to as V_3 , V_4 , V_5 , and V_6 , respectively.

For a given number of edges, a vertex in an OPP vertex may have one of several possible vertex configurations depending on the way the faces are formed by edges incident to the vertex. However, we will show that the number of different vertex configurations in any OPP is no more than 16.

Imagine that a cubical frame is divided into eight congruent smaller cubical frames and an OPP can be constructed by occupying the cubical frames with, at most, eight congruent cubes that are also congruent with the smaller cubical frames. The number of ways that the cubes occupy the cubical frames is 255, which is counted by the formula: $2^8 - 1$, where 8 is the number of cubical frames that will be occupied by the cubes, 2 is the number of possibility of each frame to be occupied, and 1 is a the number of possibility for the frame having null cubes.

Each resulting OPP at the cubical frame has a number of vertices. A different vertex configuration is counted from the most shared vertex in an OPP. The total number of different vertex configurations is stated in the following theorem.

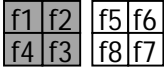
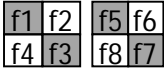
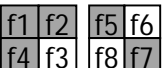
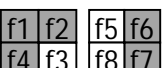
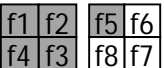
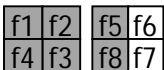
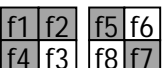
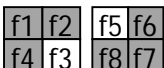
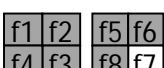
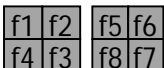
Theorem 3.1: There are no more than 16 vertex configurations in any OPP

Proof :Let B be a cubical frame that can be composed of eight smaller cubical frames of the same size, $f1, f2, \dots, f8$, and let c_1, c_2, \dots, c_8 be a set of cubes that will occupy the frames in B . Those smaller frames are grouped into a bottom layer and top layer. The members of the layer group are $f1, f2, f3$, and $f4$; meanwhile, the members of the top layer are $f5, f6, f7, f8$. An OPP is constructed by putting a number of cubes ranges from one to eight into B .

By using the above definitions and premises, then the total number of possible shapes of OPP can be detected without missing possible shapes. The possible shapes of OPP are listed in Table 3.1. Two OPPs are of similar shape and satisfy one of the above premises.

Table 3.1: Constructing OPPs Using at most Eight Cubes

N	Illustrations	Description	NSS								
1.	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>f1</td><td>f2</td><td>f5</td><td>f6</td></tr><tr><td>f4</td><td>f3</td><td>f8</td><td>f7</td></tr></table>	f1	f2	f5	f6	f4	f3	f8	f7	All frames are empty	1
f1	f2	f5	f6								
f4	f3	f8	f7								
2.	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="background-color: #cccccc;">f1</td><td>f2</td><td>f5</td><td>f6</td></tr><tr><td>f4</td><td>f3</td><td>f8</td><td>f7</td></tr></table>	f1	f2	f5	f6	f4	f3	f8	f7	A frame is occupied by a cube.	8
f1	f2	f5	f6								
f4	f3	f8	f7								
3.	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="background-color: #cccccc;">f1</td><td>f2</td><td>f5</td><td>f6</td></tr><tr><td style="background-color: #cccccc;">f4</td><td>f3</td><td>f8</td><td>f7</td></tr></table>	f1	f2	f5	f6	f4	f3	f8	f7	Two frames are occupied by two adjacent cubes.	12
f1	f2	f5	f6								
f4	f3	f8	f7								
4.	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="background-color: #cccccc;">f1</td><td>f2</td><td>f5</td><td>f6</td></tr><tr><td>f4</td><td style="background-color: #cccccc;">f3</td><td>f8</td><td>f7</td></tr></table>	f1	f2	f5	f6	f4	f3	f8	f7	Two frames are occupied by two diagonally adjacent cubes	12
f1	f2	f5	f6								
f4	f3	f8	f7								
5.	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>f1</td><td style="background-color: #cccccc;">f2</td><td>f5</td><td>f6</td></tr><tr><td>f4</td><td>f3</td><td style="background-color: #cccccc;">f8</td><td>f7</td></tr></table>	f1	f2	f5	f6	f4	f3	f8	f7	Two frames are occupied by two interstitial cubes	4
f1	f2	f5	f6								
f4	f3	f8	f7								
6.	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="background-color: #cccccc;">f1</td><td style="background-color: #cccccc;">f2</td><td>f5</td><td>f6</td></tr><tr><td style="background-color: #cccccc;">f4</td><td style="background-color: #cccccc;">f3</td><td>f8</td><td>f7</td></tr></table>	f1	f2	f5	f6	f4	f3	f8	f7	Three frames are occupied by 3-consecutive cubes.	24
f1	f2	f5	f6								
f4	f3	f8	f7								
7.	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="background-color: #cccccc;">f1</td><td style="background-color: #cccccc;">f2</td><td style="background-color: #cccccc;">f5</td><td style="background-color: #cccccc;">f6</td></tr><tr><td style="background-color: #cccccc;">f4</td><td style="background-color: #cccccc;">f3</td><td style="background-color: #cccccc;">f8</td><td style="background-color: #cccccc;">f7</td></tr></table>	f1	f2	f5	f6	f4	f3	f8	f7	Two adjacent cubes share an edge with a cube.	24
f1	f2	f5	f6								
f4	f3	f8	f7								
8.	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="background-color: #cccccc;">f1</td><td style="background-color: #cccccc;">f2</td><td style="background-color: #cccccc;">f5</td><td style="background-color: #cccccc;">f6</td></tr><tr><td style="background-color: #cccccc;">f4</td><td style="background-color: #cccccc;">f3</td><td style="background-color: #cccccc;">f8</td><td style="background-color: #cccccc;">f7</td></tr></table>	f1	f2	f5	f6	f4	f3	f8	f7	Two edges of two diagonally cubes meet with an edge of a cube.	8
f1	f2	f5	f6								
f4	f3	f8	f7								
9.	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="background-color: #cccccc;">f1</td><td style="background-color: #cccccc;">f2</td><td style="background-color: #cccccc;">f5</td><td style="background-color: #cccccc;">f6</td></tr><tr><td style="background-color: #cccccc;">f4</td><td style="background-color: #cccccc;">f3</td><td style="background-color: #cccccc;">f8</td><td style="background-color: #cccccc;">f7</td></tr></table>	f1	f2	f5	f6	f4	f3	f8	f7	The middle cube of 3-consecutive cubes shares a face with a cube.	8
f1	f2	f5	f6								
f4	f3	f8	f7								
10.	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="background-color: #cccccc;">f1</td><td style="background-color: #cccccc;">f2</td><td style="background-color: #cccccc;">f5</td><td style="background-color: #cccccc;">f6</td></tr><tr><td style="background-color: #cccccc;">f4</td><td style="background-color: #cccccc;">f3</td><td style="background-color: #cccccc;">f8</td><td style="background-color: #cccccc;">f7</td></tr></table>	f1	f2	f5	f6	f4	f3	f8	f7	An end cube of 3-consecutive cubes shares a face with a cube.	24
f1	f2	f5	f6								
f4	f3	f8	f7								
11.	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="background-color: #cccccc;">f1</td><td style="background-color: #cccccc;">f2</td><td style="background-color: #cccccc;">f5</td><td style="background-color: #cccccc;">f6</td></tr><tr><td style="background-color: #cccccc;">f4</td><td style="background-color: #cccccc;">f3</td><td style="background-color: #cccccc;">f8</td><td style="background-color: #cccccc;">f7</td></tr></table>	f1	f2	f5	f6	f4	f3	f8	f7	A shared vertex of 3-consecutive cubes shares a vertex of another cube.	24
f1	f2	f5	f6								
f4	f3	f8	f7								
12.	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="background-color: #cccccc;">f1</td><td style="background-color: #cccccc;">f2</td><td style="background-color: #cccccc;">f5</td><td style="background-color: #cccccc;">f6</td></tr><tr><td style="background-color: #cccccc;">f4</td><td style="background-color: #cccccc;">f3</td><td style="background-color: #cccccc;">f8</td><td style="background-color: #cccccc;">f7</td></tr></table>	f1	f2	f5	f6	f4	f3	f8	f7	Two diagonally adjacent cubes share vertices and edges.	2
f1	f2	f5	f6								
f4	f3	f8	f7								

13.		4-consecutive cubes occupy four frames.	6
14.		Two diagonally adjacent cubes share faces	6
15.		3-consecutive cubes share a face with diagonally adjacent cubes	8
16.		An ending face of 3-consecutive cubes shares a face with an adjacent cubes	24
17.		4-consecutive cubes share a face with a cube	24
18.		4-consecutive cubes share two face with 2-adjacent cubes.	12
19.		3-consecutive cubes share a face with 2-diagonally adjacent cubes	12
20.		3-consecutive cubes share a vertex with another 3-consecutive cubes	4
21.		3-consecutive cubes are combined with 4-consecutive cubes.	8
22.		Two 4-consecutive cubes occupy the whole frame.	1
Total number of possible shapes			255

NOTE : N→ ROW NUMBER; NSS→ NUMBER OF SIMILAR SHAPES

Table 3.1 shows how all possibilities of, at most, eight cubes occupy a cubical frame that is divided into eight cubical frames; the total number of shapes is 255, and they are distributed in 22 different shapes and sizes of OPP. The simplification is made by grouping them based on their similarity, in which two similar OPPs are grouped together, and they represent one kind of OPP. Two OPPs are called similar shapes if both of them have the same number of vertices and a similar vertex configuration. For example, the OPPs at rows 2, 3, 13, and 22 in Table 3.1 have the same number of vertices and the same vertex configuration; hence, they are grouped together as the same shape. The total possible shapes of OPPs are grouped in Table 3.2.

Table 3.2: Grouping of OPPs

Group Number	OPPs' Row in Table 3.1	Group Number	OPPs' Row in Table 3.1	Group Number	OPPs' Row in Table 3.1	Group Number	OPPs' Row in Table 3.1
1.	2,3,13,22	5.	8	9.	4,5,15	13.	10
2.	7,19	6.	12	10.	20	14.	16
3.	18	7.	11	11.	9	15.	21
4.	22	8.	17	12.	14	16.	6

To conclude, there are no more than 16 different shapes of OPP that can be constructed by arranging up to eight identical cubes in the way described above. Each OPP is counted having as one unique type of vertex configuration that is determined by the vertices that have the most shared vertex by edges in the OPP. Each the most shared vertex has a unique vertex configuration; therefore, there are no more than 16 kinds of vertex configuration in any orthogonal polyhedra. □

The 16 kinds of vertex configurations can be organized based on their degree of vertices. There are four groups of vertices namely V3, V4, V5, and V6. Each vertex configuration belongs to a group of vertices.

An OPP vertex with three edges (V3) has four possible configurations. Every vertex is two-manifold vertex and every edge is two-manifold edge. The four vertex configurations of V3 are illustrated in Figure 3.3.

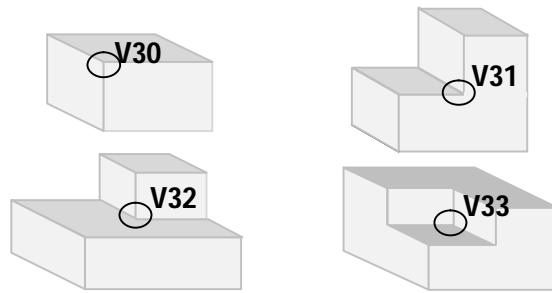


Figure 3.3: Four Possible Vertex Configuration for V3

An OPP vertex with four edges (V4) has four possible configurations, only one of which has no non-manifold edges. The four vertex configurations of V4 are illustrated in Figure 3.4.

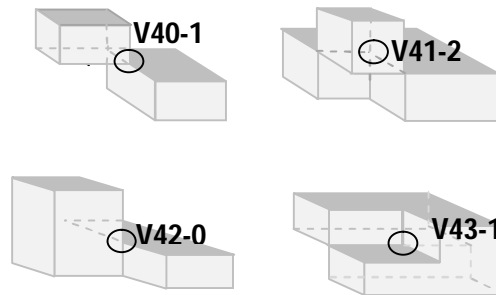


Figure 3.4: Four Possible Vertex Configurations for V4

Meanwhile, an OPP vertex with five edges (V5) has two possible vertex configurations, and both of them have non-manifold edges. The two vertex configurations for V5 are illustrated in Figure 3.5.

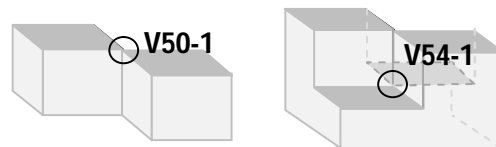


Figure 3.5: Two Possible Vertex Configurations for V5

Any OPP vertex with six edges (V6) has six possible configurations. One of them has no non-manifold edges or non-manifold vertices. Three of them have non-manifold edges but not non-manifold vertex. Two of them have a non-manifold vertex but not a non-manifold edge. All vertex configurations for V6 are shown in Figure 3.6.

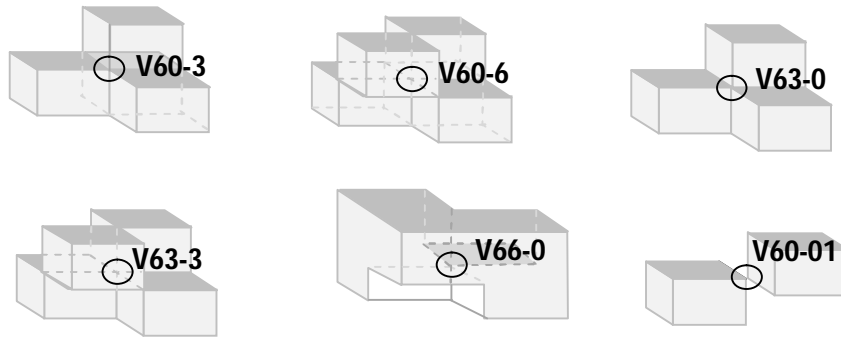


Figure 3.6: Six Possible Vertex Configurations for V6

3.4 Constructing Orthogonal Pseudo-Polyhedra

Is there any relationship among the vertex configurations of an OPP? This question may arise after identifying all the vertex configurations. The relationship can be shown with a formula, which is satisfied by different shapes of OPPs. To show a relationship among the vertex configuration of an OPP, it is initially based on a hypothesis that each OPP can be constructed from a set of rectangular prisms.

To verify the hypothesis, it is started with a statement that any OPP can be decomposed into a set of rectangular prism. This statement come from a work of Ayala and Rodriquez in which they proposed an algorithm to decompose an OPP that is represented by its extreme vertices into a particular set of disjoint boxes [76]. There are two steps to obtain the disjoint boxes from an OPP. First, split the data at every plane of vertices perpendicular to a main axis obtaining a set of slices. Second, split each slice at every plane of vertices perpendicular to another main axis obtaining a set of boxes for each slice. An OPP can be decomposed into six different set of boxes depending on the axes that is chosen to split the OPP. They are XY, YX, YZ, ZY, XZ, and ZX. The meaning of XY decomposition is OPP is split into a set of boxes by slicing on the plane of vertices that perpendicular to x -axis first, and split each slice on the plane of vertices that perpendicular to y -axis. Therefore, the number of boxes may not be unique. Figure 3.7 is an example of decomposition an OPP.

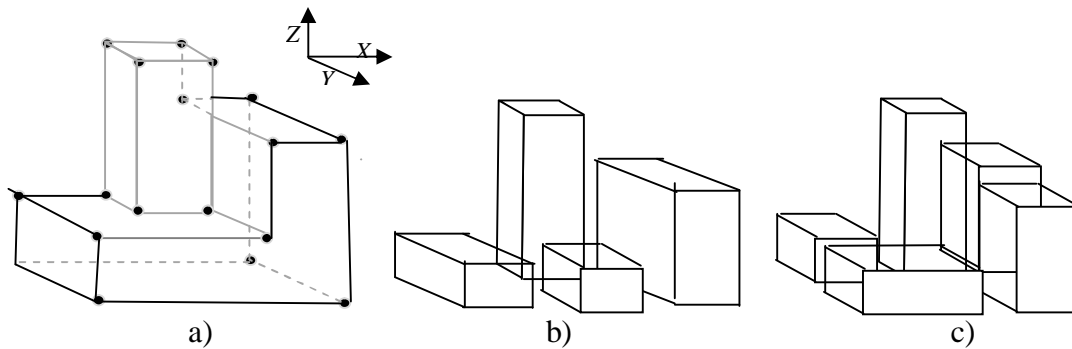


Figure 3.7: (a) An OPP. (b) XY Decomposition (4 boxes). (c) YX Decomposition (5 boxes)

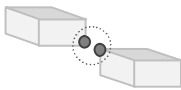
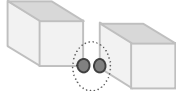
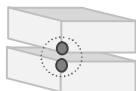
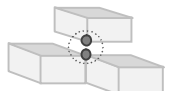
Logically, if an object can be decomposed into a set of smaller objects then the object should be able to be constructed by the set of smaller objects. This section will show how to construct an OPP from a set of rectangular prisms.

3.4.1 Joining operations

A joining operation is an operation to construct an OPP from a smaller OPP and a rectangular prism. A smaller OPP is an OPP in which its volume is less than another OPP. Therefore, each operation contains two operands, namely a smaller OPP and a rectangular prism. The location of the smaller OPP and the location of the rectangular prism are significant in joining operations.

Both of an OPP and a rectangular prism are joined at their properties. The properties of them are vertices, edges, and faces. After joining properties of an OPP and a rectangular prism, then the two joined properties will form a new format, or both of them are lost. Table 3.3 shows examples of how an OPP P and a rectangular prism R are joined.

Table 3.3: Joining Operation on an OPP

Operation Number	Description	Illustration
1.	Only a V30 vertex on P is joined with a V30 vertex on R . They gain a new V60-0 vertex.	
2.	A V30 vertex on P is joined with a V30 vertex R , in the way shown in diagram on the right. The two vertices are combined to form one V50-1.	
3.	A V30 vertex, two edges and one face on P are joined with a V30 vertex, two edges and one face on R . After joining, both V30 vertices are lost.	
5.	A V50-1 vertex on P is joined with a V30 vertex on R . They gain a new V60-3 vertex.	

To produce the complete possible results of joining two properties, a procedure is introduced. The inputs of this procedure are a set of properties of an OPP and a set of properties of a rectangular prism. The properties of the orthogonal polyhedron are 16 kinds of vertex configurations, edge, point in edge, face, and point in face. Meanwhile, the properties of the rectangular prism are identified as the V30 vertex, edge, and face.

To get the complete possible result of joining two properties, the joining operation is performed in a tree diagram. The tree contains root and children. The root of the tree is a rectangular prism, and the rectangular prism is regarded as an OPP. Some children become roots of a sub-tree, and some of them become leaves. The OPP in a root of a sub-tree contains at least two rectangular prisms and at most seven rectangular prisms. A child of a root is created by joining the OPP in a root with a rectangular prism, and the joining process is continued until the whole rectangular frames in B are occupied by rectangular prisms. Once the OPP contains eight rectangular prisms, then the OPP becomes a leaf. If all leaves are found then the tree diagram construction is finished.

As shown in Figure 3.8, each node contains an OPP. The nodes also show the part of OPPs that are joined with the properties of a rectangular prism. If the joining process creates a new vertex configuration, then the new OPP is labeled with the new created vertex configuration. Each label of the node has two parts, the joining process number and vertex configuration on the joined part. To avoid repetition, some nodes are provided with a pointer to the appropriate node. However, if the joining process does not form any kind of vertex configuration, then the new orthogonal polyhedron is labeled as follows:

(i) F-V30

The OPP is labeled as F-V30 if each vertex is a V30 vertex.

(ii) F-V31

The OPP is labeled as F-V31 if two vertices are V31 vertices, and the rest are V30 vertices.

(iii) F-V501

The OPP is labeled as F-V501 if two vertices are V50-1 vertices, and the rest are V30 vertices.

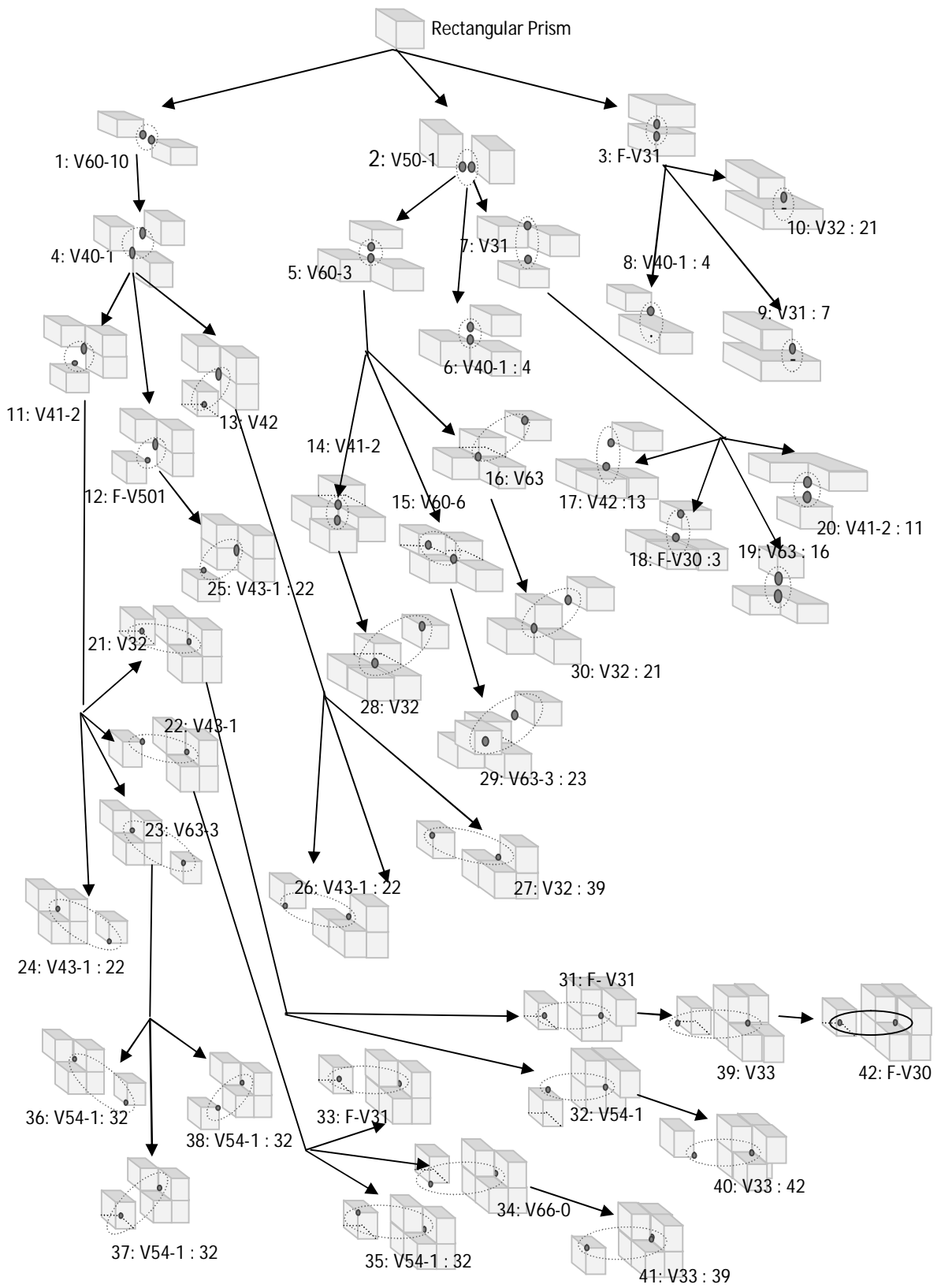


Figure 3.8: A tree Diagram of Vertex Configurations

Theorem 3.2: For any OPP, there exists a set of rectangular prisms such that the OPP can be constructed by combining these rectangular prisms together.

Proof: Ayala and Rodriguez has shown that any OPP P can be decomposed into a set of rectangular prisms [76], so we can compose reversely all rectangular prisms to construct P . The task to compose P from a set of rectangular prisms comes from the definition that two rectangular prisms are joined if they share at least one corner point and that two corner points are a share point if they have the same coordinate. Therefore, an OPP is obtained by joining a rectangular prism with a smaller OPP that has at least one shared corner points. This task is carried out repeatedly until P is achieved. \square

3.5 The Vertex Configuration Conjecture

Recall that Voss classified the orthogonal polygon boundary into the inner or outer boundary based on the relationship between concave and convex vertices in a two-dimensional digital image [41]. Yip and Klette mentioned that an orthogonal polyhedron may also have an outer boundary as well as an inner boundary [39] if the orthogonal polyhedron has a hole inside.

For a simple orthogonal polyhedron where there is only an outer boundary, Yip and Klette [39] established a formula that the relationship among the different vertex configurations in an orthogonal polyhedron can be characterized by the following formula: $(HA+HG) - (HC+HE) - 2(HD1+HD2) = 8$, where HA , HG , HC , HE , $HD1$, and $HD2$ denote the number of V_{30} , V_{33} , V_{31} , V_{32} , V_{42-0} , and V_{63-0} types of vertices in the orthogonal polyhedron respectively. They also proved that for an orthogonal polygon, the relationship between the number convex vertex (HC) and reflect vertices (HR) is: $HC - HR = 4$.

The first formula is useful for analyzing the boundaries of simple orthogonal polyhedra. The second formula can be used to analyze polygonal boundary. Yip and Klette suggested that the first formula can be used in 3D pattern analysis by providing a necessary condition for

having traced a complete 3D surface of a simple orthogonal polyhedron [39].

A digital image on 3D may have an OPP representation. After introducing the vertex configurations, it is time now to conjecture the vertex configurations relationship of an OPP.

Vertex Configuration Conjecture : Let N_{V30} , N_{V31} , N_{V32} , N_{V33} , N_{V401} , N_{V412} , N_{V420} , N_{V431} , N_{V501} , N_{V541} , N_{V600} , N_{V603} , N_{V606} , N_{V630} , N_{V633} , and N_{V660} denote the number of vertex V30, V31, V32, V33, V40-1, V41-2, V42-0, V43-1, V50 and V54-1, V60 -01, V60-3, V60-6, V63-0, V63-3, and V66-0 respectively in an OPP. The relationship among the number of each type of vertices is:

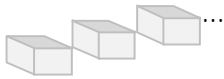
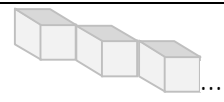

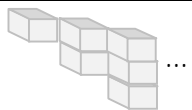
$$(N_{V30} + N_{V33} + 0N_{V412} + N_{V431} + 2N_{V541} + 6N_{V606} + 3N_{V633} + 2N_{V660}) - (N_{V31} + N_{V32} + 3N_{V401} + 2N_{V420} + 2N_{V501} + N_{V603} + 6N_{V600} + 2N_{V630}) = 8$$


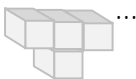

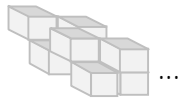
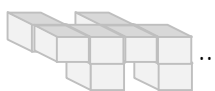
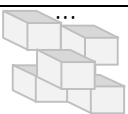

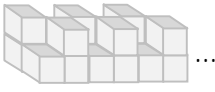
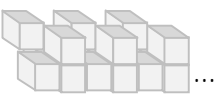

The following is a number of evidences to show that there is a good chance the conjecture is right.



1. It is proven in Theorem 3.2 that for any OPP, there exists a set of rectangular prisms such that the OPP can be constructed by combining these rectangular prisms together. The process starts with marking the first rectangular prism in the sequence as the OPP. It has exactly eight V30 vertices, and no others kind of vertices. In the next step, a next rectangular prism is added to the OPP. Assume that the OPP is congruent with the rectangular prism. After combining the OPP and the rectangular prism at their face — the whole of a face of the OPP replaces the whole of a face of the rectangular prism — then a new rectangular prism is formed. The new OPP is not congruent with the previous OPP or rectangular prism; however, the new OPP still has eight vertices, and all of them are V30-vertex. Therefore, for this case the conjecture is valid.

2. The conjecture is also true for some OPPs, as shown in Table 3.4. Each OPP only contains several types of vertex configurations, and the number of the other vertex configurations is null. The vertex configuration relationship for each OPP is obtained by determining the relationship between the number of each vertex type at n -term of the arithmetic sequence. For example, OPP at Row 1 has two kinds of vertices, the V30 vertex and the V60-1 vertex. The n -term of the arithmetic sequence for the V30 vertex is given by $S_n = 6n + 2$; meanwhile, the arithmetic sequence for the V60-1 vertex is given by $S_n = a + (n - 1)b$, where a is the first term value and b is the different value between two contiguous terms [78]. The relationship between these kinds of vertices can be written as $6n+2 = 6(n-1) + 8$, or in terms of number of each vertex, it can be written as $N_{V30} - 6N_{V60-1} = 8$. Therefore, the conjecture is valid.

Table 3.4: Relationship among the Vertex Configurations on Simpler OPPs

No	Involved Vertices	Illustrations	Variable Number Vertex	Number of Vertex Configurations in Sequence					Vertex Configuration Relationship
				S1	S2	S3	...	Sn	
1	V30,		N_{V30}	8	14	20	...	$6n+2$	$6n+2 - 6(n-1) = 8$ \leftrightarrow $N_{V30} - 6N_{V60-01} = 8$
	V60-01		N_{V60-01}	0	1	2	...	$n-1$	
2	V30		N_{V30}	8	12	16	...	$4n + 4$	$4n+4 - 2(2n-2)=8 \leftrightarrow$ $N_{V30} - 2N_{V50-1} = 8$
	V50-1		N_{V50-1}	0	2	4	...	$2n-2$	
3	V30		N_{V30}	8	8	8	...	8	$8=8 \leftrightarrow$ $N_{V30} = 8$
4	V30		N_{V30}	8	13	18	...	$5n+3$	$5n+3 - 3(n-1) - 2(n-1)$ $=8 \leftrightarrow$ $N_{V30} - 3N_{V40-1} - 2N_{V50-1} = 8$
	V40-1		N_{V40-1}	0	1	2	...	$n-1$	
	V50-1		N_{V50-1}	0	1	2	...	$n-1$	
5	V30		N_{V30}	12	15	18	...	$3n + 9$	$3n+9 - 2(n+1) - (n-1)=8$

	V50-1		N_{V50-1}	2	3	4	...	$n+1$	\leftrightarrow $N_{V30} - 2N_{V50-1} - N_{V60-3} = 8$
	V60-3		N_{V60-3}	0	1	2	...	$n-1$	
9	V30		N_{V30}	8	10	12	...	$2n+6$	$2n+6 - (2n-2) = 8$
	V31		N_{V31}	0	2	4	...	$2n-2$	$N_{V30} - N_{V31} = 8$
10	V30		N_{V30}	8	11	14	...	$3n+5$	$3n+5 - (2n-2) - (n-1) = 8 \leftrightarrow$ $N_{V30} - N_{V31} - 2N_{V32} = 8$
	V31		N_{V31}	0	2	4	...	$2n-2$	
	V32		N_{V32}	0	1	2	...	$n-1$	
11	V30		N_{V30}	13	20	27	...	$8n+5$	$8n+5 - 1 - 2(4n-2) = 8$ \leftrightarrow $N_{V30} + 0N_{V41-2} - N_{V31} - 2N_{V50-1} = 8$
	V31		N_{V31}	1	1	1	...	1	
	V50-1		N_{V50-1}	2	6	10	...	$4n-2$	
	V41-2		N_{V41-2}	1	3	5	...	$2n-1$	
13	V30		N_{V30}	8	12	16	...	$4n+4$	$4n+4 - (2n-2) - 2(n-1) = 8 \leftrightarrow$ $N_{V30} - N_{V31} - N_{V42} = 8$
	V31		N_{V31}	0	2	4	...	$2n-2$	
	V42		N_{V42}	0	1	2	...	$n-1$	
15	V30		N_{V30}	12	14	16	...	$2n+10$	$2n+10 + 6(n-1) - 2(4n-2) = 8 \leftrightarrow N_{V30} + 6N_{V60-6} - 2N_{V50-1} = 8$
	V50-1		N_{V50-1}	2	6	10	...	$4n-2$	
	V60-6		N_{V60-6}	0	1	2	...	$n-1$	
16	V30		N_{V30}	13	21	29	...	$8n+5$	$8n+5 - (4n-1) - 2(2n-1) = 8 \leftrightarrow$ $N_{V30} - N_{V31} - 2N_{V63} = 8$
	V31		N_{V31}	3	7	11	...	$4n-1$	
	V63		N_{V63}	1	3	5	...	$2n-1$	
22	V30		N_{V30}	12	18	24	...	$6n+6$	$6n+6 + (2n-1) - (4n-1) = 8 \leftrightarrow$ $2(2n-1) = 8 \leftrightarrow$ $N_{V30} + N_{V43-1} - N_{V31} - 2N_{V50-1}$
	V50-1		N_{V50-1}	1	3	5	...	$2n-1$	
	V31		N_{V31}	3	7	11	...	$4n-1$	
	V43-1		N_{V43-1}	1	3	5	...	$2n-1$	
23	V30		N_{V30}	14	19	24	...	$5n+9$	$5n+9 + (2n-1) - 3n - 2(4n-1) = 8 \leftrightarrow$ $N_{V30} + 3N_{V63-3} - N_{V31} - 2N_{V50-1} = 8$
	V31		N_{V31}	3	6	11	...	$3n$	
	V50-1		N_{V50-1}	3	7	9	...	$4n-1$	
	V63-3		N_{V63-3}	1	3	5	...	$2n-1$	
32	V30		N_{V30}	10	12	14	...	$2n+8$	$2n+8 + 2(n-1) - 2n - 2(n-1) = 8 \leftrightarrow$ $N_{V30} + 2N_{V54-1} - N_{V31} - 2N_{V50-1} = 8$
	V31		N_{V31}	2	4	6	...	$2n$	
	V50-1		N_{V50-1}	0	1	2	...	$n-1$	
	V54-1		N_{V54-1}	0	1	2	...	$n-1$	

34	V30		N_{V30}	10	12	14	...	$2n+8$	$2n+8 + 2(n-1) - (4n-2) = 8 \leftrightarrow N_{V30} + 2N_{V66-0} - N_{V31} = 8$
	V31		N_{V31}	2	6	10	...	$4n-2$	
	V66-0		N_{V66-0}	0	1	2	...	$n-1$	
39	V30		N_{V30}	10	12	14	...	$2n+8$	$2n+8 + (2n-1) - (4n-1) \leftrightarrow N_{V30} + N_{V33} - N_{V31} = 8$
	V31		N_{V31}	3	7	11	...	$4n-1$	
	V33		N_{V33}	1	3	5	...	$2n-1$	

3. There exists a consistent way to show the validity of the conjecture in the joining process of OPPs with a rectangular prism in the tree diagram in Figure 3.8.

Let $P1$ be an OPP and $P2$ be a rectangular prism in Figure 3.8, and let $R = (N_{V30} + N_{V33} + 0N_{V412} + N_{V431} + 2N_{V541} + 6N_{V606} + 3N_{V633} + 2N_{V660}) - (N_{V31} + N_{V32} + 3N_{V401} + 2N_{V420} + 2N_{V501} + N_{V603} + 6N_{V600} + 2N_{V630})$. Before a joining operation, R has a value of 16. The rest of this section will show that after the joining operation, R will have value of 8.

In a joining process, a set property of $P1$ meets correspondently a set property of $P2$. Because of the meeting, some properties will be lost, and others will change type. Therefore, the value of R will change. For example, if the V30 vertex and two of its incident edges of $P1$ meet with the V30 vertex and two incident edges of $P2$, then the two V30 vertices will be lost, and the value of R will decrease by 2. Another example is if the V30 vertex meets with a point on a line and each adjacent surface of each property coincides with each other, then the V30 vertex will be lost. Instead, they are replaced by a V31 vertex, and the value of R will decrease by 2. The difference between the R value and its value after increasing or decreasing due to the joining of two properties of $P1$ and $P2$ is symbolized with ΔR .

To calculate ΔR , the nodes in the tree diagram in Figure 3.8 are used. All kinds of ΔR may be founded based on the tree diagram. However, this thesis does not prove that there are no more properties found. The value of ΔR is calculated with the following steps:

- (1) Select a node in the tree diagram in Figure 3.8, which contains an OPP and a rectangular prism. The total value R of both the OPP and the rectangular prim is 16.
- (2) Determine the list of joined properties in the node.
- (3) ΔR of a pair of properties can be counted if ΔR of the other pairs of properties are known.
- (4) Find the relationship among the vertex configuration of the new OPP in Table 3.4
- (5) ΔR of a pair of properties is counted based on the difference between the R value of the OPP in Step (4) and with the R value in Step (1). After substituting the ΔR of each known pair of properties, then the ΔR of the unknown pair of properties is obtained.

Here is an example to calculate ΔR from the given two OPPs. Let $P1$ and $P2$ be two rectangular joined at four corners. Due to rectangular form, $P1$ and $P2$ only have V30 vertices. The value of vertex configuration on both $P1$ and $P2$ is 16 as shown in Figure 3.8. Meanwhile, if $P1$ is joined with $P2$, then they will form a new OPP in which all corners are the V30 vertex, and the value of its vertex configuration is 8. Therefore, the different value of the vertex configurations between the two original $P1$ and $P2$ with the joined $P1$ and $P2$ is 8. It means that the two joined vertices decrease the total value by 2.

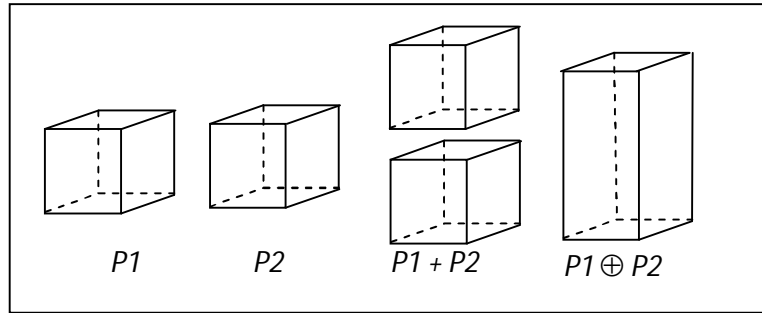


Figure 3.9: P is a Result of Joining $P1$ and $P2$

$P1$ and $P2$ are joined by operator (\oplus) such that sets of properties of $P1$ coincide with sets of properties of $P2$. The relationship that involves those kinds of set properties is shown in Table 3.5 Number 3.

By using the same way, the decreasing or increasing value of R for each set of properties as summarized in Table 3.5.

Table 3.5: Operations and their ΔR Value

Ordered Number	Involved properties	ΔR -Value		New Property
		Increase	Decrease	
1.	P1: V30; P2: V30		8	V60-01
2.	P1: (V30,1E); P2: (V30, 1E)		4	V50-1
3.	P1: (V30, 2E,1F); P2: (V30,2E,1F)		2	Nothing
4.	P1: (V60-01,2E,1F); P2: (V30,2E,1F)	2		V40-1
5.	P1: (V50-1, 2E); P2: (V30, 2E)	0	0	V60-3
6.	P1: (V50-1, 2E,1F); P2: (V30, 2E, 1F)		2	V40-1
7.	P1: (V50-1, 3E, 2F); P2: (V30, 3E, 2F)	0	0	V31
8.	P1: (Point IN E, 1E); P2: (V30, 1E)		4	V40-1
9.	P1: (Point IN E, 1E, 1F); P2: (V30, 1E, 1F)		2	
10.	P1: (Point IN F, 1F); P2: (V30, 1F)		2	V31
11.	P1: (V40-1, 2E, 1F); P2: (V30, 2E, 1F)	2		V41-2
12.	P1: (V40-1, 2E, 1F); P2: (V30, 2E, 1F)	2		Nothing
13.	P1: (V40-1, 1E, 2F); P2: (V30, 1E, 2F)	2		V42
14.	P1: (V60-3, 3E, 2F); P2: (V30, 3E, 2F)	2		V41-2
15.	P1: (V60-3, 3E); P2: (V30, 3E)	4		
16.	P1: (V60-3, 3E, 3F); P2: (V30, 3E,3F)		4	V63
17.	P1: (V31, 1E, 1F); P2: (V30, 1E, 1F)		2	V42
18.	P1: (V31, 3E, 2F); P2: (V30, 3E, 2F)	0	0	Nothing
19.	P1: (V31, 1F); P2: (V30, 1F)		2	V63
20.	P1: (V32, 2E); P2: (V30, 2E, 2F)	0	0	V41-2
21.	P1: (V41-2, 3E, 3F); P2: (V30, 3E, 3F)		2	V32
22.	P1: (V41-2, 1E, 2F); P2: (V30, 1E, 2F)	0	0	V43-1
23.	P1: (V41-2, 1E,1F); P2: (V30, 1E, 1F)	2		V63-3

24.	P1: (V41-2, 2E, 2F); P2: (V30, 2E, 2F)	0	0	V43-1
25.	P1: (Point in NE, 1E, 2F); P2: (V30, 1E, 2F)		4	V43-1
26.	P1: (V42, 2E, 1F); P2: (V30, 2E, 1F)	2		V43-1
27.	P1: (V42, 2E, 2F); P2: (V30, 2E, 2F)	0	0	V32
28.	P1: (V41-2, 3E, 3F); P2: (V30, 3E, 3F)		2	V32
29.	P1: (V60-6, 3E, 3F); P2: (V30, 3E, 3F)		2	V63-3
30.	P1: (V63, 3E, 2F); P2: (V30, 3E, 2F)	0	0	V32
31.	P1: (V32, 2E, 2F); P2: (V30, 2E, 2F)	0	0	Nothing
32.	P1: (V32, 1E, 1F); P2: (V30, 1E, 1F)	2		V54-1
33.	P1: (V43-1, 3E, 3F); P2: (V30, 3E, 3F)		2	Nothing
34.	P1: (V43-1, 1E, 2F); P2: (V30, 1E, 2F)	0	0	V66-0
35.	P1: (V43-1, 1E, 2F); P2: (V30, 1E, 2F)	0	0	V54-1
36.	P1: (V63-3, 3E, 3F); P2: (V30, 3E, 3F)		2	V54-1
37.	P1: (V63-3, 3E, 3F); P2: (V30, 3E, 3F)		2	V54-1
38.	P1: (V63-3, 3E, 3F); P2: (V30, 3E, 3F)		2	V54-1
39.	P1: (Point in ME, 1E, 2F); P2: (V30, 1E, 2F)	0	0	V33
41	P1: (V54-1, 3E, 3F); P2: (V30, 3E, 3F)	2		V33
42	P1: (V66-0, 3E, 3F); P2: (V30, 3E, 3F)		2	V33

By applying the suitable ΔR for each joined properties during joining $P1$ and $P2$ to form the new OPP, then the R value changes from 16 to 8 after joining. \square

3.6 Duality of Vertex Configurations

Recall that the polyhedral surface in a polyhedron is also called the *shell* of that polyhedron, and a *multi-shell polyhedron* is a solid shape that consists of one large polyhedron and one or more smaller polyhedra that are located completely inside the large polyhedron in which these smaller polyhedra neither intersect with the large polyhedron, nor intersect with each other, nor overlap with each other. The Euler formula for a multi-shell polyhedron is $v - e + f = 2(s - g)$, where v , e , f , s , and g is denoted as the number of vertices, edges, faces, shell, and genus [36].

Figure 3.10 is a multi-shell orthogonal polyhedron. The small polyhedron in the multi-shell orthogonal polyhedron is bounded by an inner boundary. The small polyhedron in the larger orthogonal polyhedron is also an orthogonal polyhedron. Therefore, a vertex on an inner boundary has a duality. Figure 3.10 shows that vertex A has configuration V33 as an inner

boundary of a simple orthogonal polyhedron, and vertex A is also V30 vertex as an outer boundary of an empty space inside the simple orthogonal polyhedron.

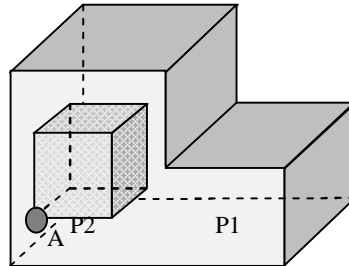


Figure 3.10: A is a Vertex on the Inner Boundary of $P1$ and Outer Boundary of $P2$

By observation, the dualities of vertex configurations of OPP are listed in Table 3.6.

Table 3.6: Vertex Configurations and Their Duality

No.	Vertex Configuration	Duality
1.	V30	V33
2.	V31	V32
3.	V32	V31
4.	V33	V30
5.	V40-1	V43-1
6.	V41-2	V41-2
7.	V42-0	V42-0
8.	V43-1	V40-1
9.	V50-1	V54-1
10.	V54-1	V50-1
11.	V60-3	V63-3
12.	V60-6	V60-6
13.	V63-0	V63-0
14.	V63-3	V60-3
15.	V66-0	V60-01
16.	V60-01	V66-0

3.7 Summary

This chapter has proven that there are no more than 16 vertex configurations for any orthogonal pseudo-polyhedra. It is also proven that for any OPP, there exists a set of rectangular prisms such that the OPP can be constructed by combining these rectangular prisms together. This chapter also conjectured a formula to show the relationship among the number of each type of vertex in any orthogonal pseudo-polyhedra. This chapter has provided some evidences to support the conjecture. The last section in this chapter has identified the dual vertex configuration of each type of vertex configuration for any OPP having inner and outer boundaries.

The next chapter covers splitting techniques for several models of geometry objects in both 2D and 3D.

CHAPTER 4

SPLITTING OPERATIONS

4.1 Introduction

This chapter focuses on developing algorithms to split orthogonal polygon and polyhedra. The chapter will develop algorithms: splitting an orthogonal polygon with polyline, splitting an orthogonal polyhedron with a polyplane, and splitting polyhedra in a bounding box. The main contribution of this chapter is to provide effective method to do splitting on orthogonal polygon, orthogonal polyhedron, and polyhedra. The procedure of splitting orthogonal polygon and an orthogonal polyhedron using polyline and polyplane, respectively can be applied to a real work activity, such splitting metal in a metal fabrication.

The task of orthogonal polyhedron decomposition is to represent an orthogonal polyhedron as the union of a number of simpler component parts. Orthogonal polyhedron decomposition has many theoretical and practical applications. For example, decomposition of an orthogonal pseudo-polyhedron into a set of rectangular prisms is a useful step for successful guard placement in the 3D-AGP.

Generally, decomposition of an object into a set of smaller objects can be achieved by a set of splitting operations. A *splitting operation* is an operation to split an object into two parts, with each part containing one or more smaller objects. The object is split by a *splitting tool* that crosses the object until it hits its boundary. For example, Ayala and Rodriguez [76] proposed a procedure to decompose an OPP into a set of rectangular prisms. In this procedure, the splitting process is applied many times, and each splitting plane contains a

single plane that divides the OPP into two parts, each consisting of one or more smaller OPPs.

Imagine that an L-shape piece of metal will be taken from a rectangular piece of metal in a metal fabrication. To get the piece, a splitting method that uses a single splitting plane for each process, such as proposed by Ayala and Rodriguez [76], can be deployed. The steps are pictured in Figure 4.1, as follows: First, a thick metal, represented by a rectangle (Figure 4.1(a)), is split in a horizontal direction from the left boundary until it hits the right boundary and the resulting pieces are split again from each piece in the vertical direction (see Figure 4.1(b)). Finally, three of four pieces (Figure 4.1(c)) are combined in their boundary to get the L-shape piece metal.

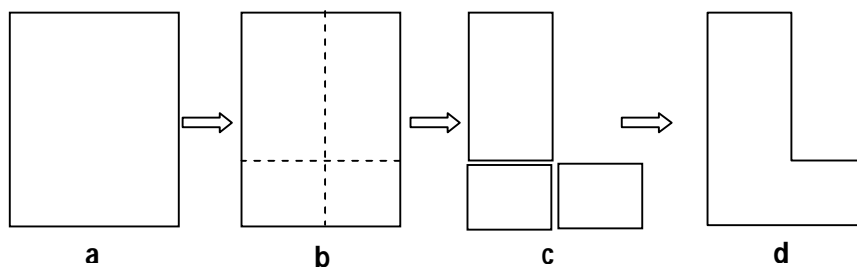


Figure 4.1: Steps to Get an L-Shape Piece

The above description shows that the splitting operation is applied twice and then followed by combining some parts to get the L-shape. This method seems inefficient. It is reasonable to ask the process in order to not involve only one splitting operation such that the combining step is no longer necessary. This question is a motivation of some of the works in this chapter.

In the practical situation illustrated in Figure 4.1, an orthogonal polyhedron needs to be split into exactly two simpler orthogonal polyhedra. Therefore, another kind of splitting technique is required. One possible splitting technique is to introduce a splitting plane that contains one

or more connecting planes. By using this kind of splitting plane, an orthogonal polyhedron can be split into two smaller orthogonal polyhedra within a single splitting operation.

A splitting tool for a polygon is a line or a set of connected lines, and a splitting tool for a polyhedron is a plane or a set of connected planes. A set of connected lines is called a *polyline*, and a set of connected planes is called *polyplane*. Each segment in a polyline or polyplane parallels to one of the axes in a Cartesian coordinate system.

In this chapter, a number of splitting algorithms are developed for different object models, orthogonal polygons and orthogonal polyhedra. Section 4.2 will develop an algorithm for splitting an orthogonal polygon with a polyline. In Section 4.3, an algorithm is developed for splitting an orthogonal polyhedron with a polyplane.

A splitting plane that contains only a single plane (not parallel to one of the three axes) can be used to split a bounding box that is a rectangular prism that contains one or more polyhedra. The splitting plane will divide the bounding box into two halves, in which each half consists of one or more polyhedra. In Section 4.4, an algorithm is provided for splitting a set of convex polyhedra in a bounding box using a splitting plane.

4.2 Splitting an Orthogonal Polygon Using Polyline

Splitting a polygon into two halves using a single splitting line is a well known-problem, and numerous methods have been proposed such as by Daniels *et al.* [79]. However, less method is known to split an orthogonal polygon using a polyline. In this section, a procedure of splitting an orthogonal polygon using a polyline is established. For that purpose, definitions and terminology are introduced first.

4.2.1 Definitions and terminology

Polyline is the term for a polygonal chain in a computer graphic. It is a connected series of line segments connecting the consecutive vertices that are treated as a single entity [80]. An *orthogonal polyline* is a polyline in which each segment of the polyline is parallel to one of the two axes. Henceforth, the term polyline is used to refer an orthogonal polyline.

In this thesis, a polyline is restricted by three conditions. First, the whole segment of a polyline must lie inside of an orthogonal polygon. Second, a segment of a polyline cannot lie on edges of an orthogonal polygon. Third, a polyline intersects the boundary of an orthogonal polygon only at two points. The point in which a polyline intersects the boundary of an orthogonal polygon is an *intersection vertex*. In Figure 4.1, a connected segment lines passes through v_{s1} , v_{s2} , and v_{s3} . The connected lines lie totally in an orthogonal polygon, and the connected lines intersect the boundary in two points; therefore, the connected line can be considered as a polyline.

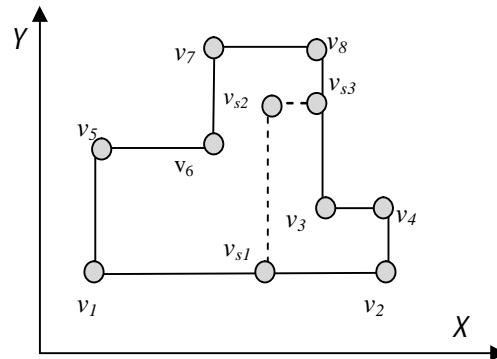


Figure 4.2: An Orthogonal Polygon with a Polyline

In Figure 4.2, v_1, \dots, v_8 are vertices of an orthogonal polygon. Each vertex has a coordinate to represent its position in the 2D Cartesian coordinate system. v_{s1} , v_{s2} , and v_{s3} are vertices of a polyline, in which v_{s1} and v_{s3} are intersection vertices obtained from intersection between the orthogonal polygon with the polyline, and they are a starting vertex and an ending vertex of the polyline, respectively.

The data representation for an orthogonal polygon has been studied by O'Rourke [68], and this data structure has been reviewed in Chapter 2. In this data structure, an orthogonal polygon is presented as in *AB-sorted vertices*. An *AB-sorted vertices* is a sequence of vertices where their coordinate are sorted first by coordinate *A*, and then by coordinate *B*. The vertices can be sorted in two different ways: *XY-sorted vertices* or *YX-sorted vertices*. Accessing vertices based on sorted-vertices is standard in computational geometry. Wu, Tian, and Xie also used sorted vertices as input for splitting arbitrary polygons [81]

Meanwhile, a polyline can be easily represented by an ordered sequence of vertices, v_1, v_2, \dots, v_n where each vertex is represented by its Cartesian coordinate. Because the proposed polyline is an orthogonal polyline, any two consecutive vertices will share one coordinate value. v_1 and v_n are vertices of a polyline that also lie on the boundary of an orthogonal polygon, and both are intersection vertices.

A polyline may contain a single line that is perpendicular to the *x*-axis or *y*-axis. Figure 4.3 shows two orthogonal polygons with different polylines. Figure 4.3(a) is an orthogonal polygon with a single line segment as a splitting line, and Figure 4.3(b) is an orthogonal polygon with a polyline as a splitting line.

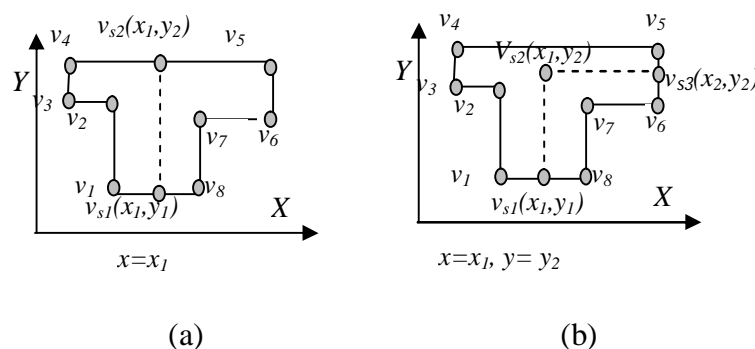


Figure 4.3: Polylines on orthogonal polygons

Each of the above figures is explained as follow:

- (i) In Figure 4.3(a), an orthogonal polygon is split by a polyline that consists of vertices v_{s1} and v_{s2} . This polyline has only one segment line. After splitting, vertices of the orthogonal polygon and the polyline will be grouped into two orthogonal polygons: $Q = \{ v_{s1}, v_1, v_2, v_3, v_4, v_{s2} \}$ and $R = \{ v_{s1}, v_8, v_7, v_6, v_5, v_{s2} \}$.
- (ii) In Figure 4.3(b), the orthogonal polygon is split by a polyline that consists of vertices v_{s1} , v_{s2} , v_{s3} . This polyline has two connected line segments. The result of splitting is $Q = \{ v_{s1}, v_1, v_2, v_3, v_4, v_5, v_{s3}, v_{s2} \}$ and $R = \{ v_{s1}, v_8, v_7, v_6, v_{s3}, v_{s2} \}$.

4.2.2 An algorithm for splitting an orthogonal polygon using a polyline

In this section, an algorithm on how to split an orthogonal polygon by polyline will be described. The inputs of the algorithm are vertices of an orthogonal polygon and vertices of a polyline. The following steps are used to split an orthogonal polygon into two smaller orthogonal polygons:

1. Combine the vertices of an orthogonal polygon and a polyline into a set of combined vertices.
2. Group the set of combined vertices into two groups of vertices in which each group represents a smaller orthogonal polygon.

Based on the above steps, two procedures are needed. The first is a procedure for combining the vertices of an orthogonal polygon and a polyline, which will be discussed in Subsection 4.2.3. The second is a procedure for grouping the combined vertices into two groups of smaller orthogonal polygons; this procedure will be discussed in Subsection 4.2.4.

4.2.3 Combining the vertices of an orthogonal polygon and a polyline

Combining vertices is a process to combine the vertices of an orthogonal polygon and a polyline into a set of combined vertices that contains two smaller orthogonal polygons. The procedure to get a set of combined vertices is based on the following observations: i) each vertex of an orthogonal polygon is a vertex of a smaller orthogonal polygon; therefore, all vertices of an orthogonal polygon are added to a set of combined vertices. ii) A polyline will become the boundary of each orthogonal polygon in the new set of vertices; therefore, each vertex of a polyline is added in such a way as to the set of combined vertices. There are two types of polyline vertices: non-ending vertices and ending vertices. A non-ending vertex belongs to two smaller orthogonal polygons; hence, a non-ending vertex is added twice to the combined vertices. Meanwhile, an ending vertex of a polyline lies on the edge or vertex of the orthogonal polygon, and the ending vertex is called as an *intersection vertex*. If an ending vertex lies on an edge, then add twice this intersection vertex to the set of combined vertices; if it lies on a vertex, then update the vertex and an adjacent vertex as in the set of combined vertices as intersection vertex (see Figure 4.4). Figure 4.4(a) shows an orthogonal polygon and a polyline before splitting. Meanwhile, Figure 4.4(b) shows the set of combined vertices after splitting.

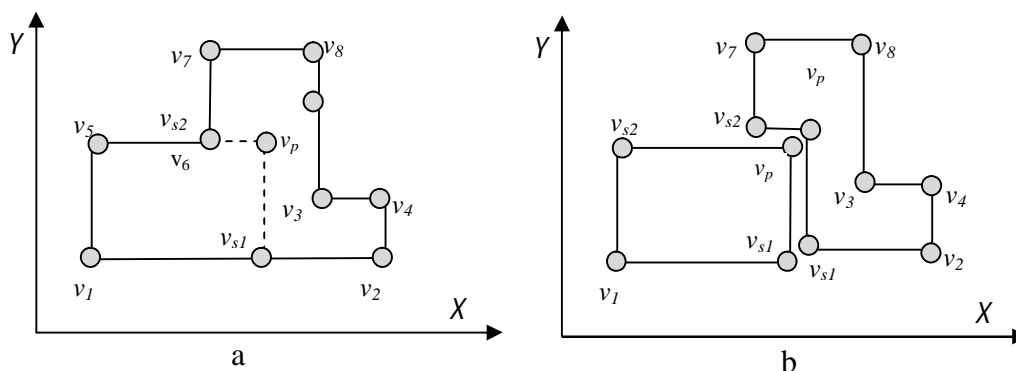


Figure 4.4: One of the Two Intersection Vertices Lies on the Vertex of an Orthogonal Polygon

Based on the observation, the source of vertices in a set of combined vertices are the vertices of the orthogonal polygon, non-ending vertices of a polyline, and intersection vertices.

Figure 4.5 is a procedure for combining, which is called `CombiningVertices`. The procedure has two kinds of input: vertices of an orthogonal polyhedron and vertices of a polyline. The output is a set of combined vertices. The procedure contains a function, `ReadPolyline`, which has a function to read one by one the vertices in a polyline, and a procedure, `ReadVertexAdjacent`, which has a purpose to determine the adjacent vertex of an intersection vertex in an orthogonal polygon.

An adjacent vertex is found as follows. Let v_s and v_r be two end vertices of a segment line in a polyline, then if $v_s v_r$ is perpendicular to x -axis, sort the vertices of the orthogonal polygon in xy -sorted to read the pair of v_s as adjacent vertex v_s' in the orthogonal polygon; or if $v_s v_r$ is perpendicular to y -axis then sort the vertices of the orthogonal polygon in yx -sorted to read the pair of v_s . After finding v_s' , update the source of vertex v_s and v_s' in the set of combined vertices as intersection vertex.

```

Procedure CombiningVertices(INPUT  $P$ : an orthogonal polygon,  $PL$ : a polyline ;
OUTPUT  $cv$ : a set of combined vertices)

var  $v_s, v_r$  : a vertex of  $PL$ ;  $v$ : a vertex of  $P$  ;  $m$ : number of vertices in  $PL$ 
 $cv = \emptyset$ ;
 $cv = cv + P$ ; // add all vertices of  $P$  into the set of combined vertices  $cv$ 
for ( $i = 1$  to  $m$ ) {
     $v_s = \text{ReadPolyline}(PL)$ 
    if  $v_s = \text{non-ending vertex}$  {
         $cv = cv + v_s + v_s$ 
    }
    else
        If ( $v_s = \text{ending vertices and lies on edge}$ ) {
             $cv = cv + v_s + v_s$ 
        }
        else //  $v_s = \text{ending vertices and lies on vertex of } P$ 
             $\text{ReadVertexAdjacent}(v_s, v_r ; v_s')$  // procedure to determine  $v_s'$ 
            update  $v_s$  and  $v_s'$  in  $cv$  as intersection vertex;
    }
}

```

Figure 4.5: Procedure `CombiningVertices` for Combining Vertices

4.2.4 Grouping vertices

The new set of vertices of the orthogonal polygon after adding all vertices of a polyline represents two orthogonal polygons, in which each orthogonal polygon has a group of vertices. However, the vertices record has not yet given any information about the group of a vertex. Therefore, a procedure is needed to group the vertices.

Grouping the vertices into two smaller orthogonal polygons is based on the following observations: (i) each vertex of the original orthogonal polygon belongs to one of the smaller orthogonal polygons; (ii) each non-ending vertex in the polyline belongs to the two orthogonal polygons; (iii) An intersection vertex belongs to a smaller orthogonal polygon.

An orthogonal polygon has one boundary. This means that there is a path that passes through all the vertices and edges of the orthogonal polygon. The boundary of a smaller orthogonal polygon consists of a part of boundary of the original orthogonal polygon and a polyline. A polyline starts with an intersection vertex and finishes at another intersection vertex with different coordinate. Each smaller polygon has the same polyline; hence, the remaining path for a smaller orthogonal polygon is a path that starts with an intersection vertex and then continues to walk until meet the other intersection vertex.

Figure 4.6 shows the algorithm to group vertices into two orthogonal polygons. The procedure starts by running a sub-procedure `ReadNonEndingVertices`, which reads all non-ending vertices in the set of combined vertices cv . All different non-ending vertices v_{ne} become the vertices of a smaller orthogonal polygon Q , while the other group of smaller orthogonal polygon R contains the set of combined vertices minus v_{ne} . See Figure 4.6 for detail.

```

Procedure GroupingVertices (INPUT  $cv$ : a set of combined vertices,  $cv_{xy}$ :  $cv$  in  $xy$ -sorted,  $cv_{yx}$ :  $cv$  in  $yx$ -sorted; OUTPUT  $Q, R$ : two orthogonal polygons)

var
   $v_{ne}$       : non-ending vertices in  $cv$ 
   $v_{s1}, v_{s2}$  : intersection vertices
   $v_i, v_t$    : vertices of NP
   $direction$ : Boolean variable
   $PL$         : vertices of a polyline

ReadNonEndingVertices( $cv; v_{ne}$ ) // a procedure to read the set of non vertices in  $cv$ 
 $Q = v_{ne}; R = cv - v_{ne}$ 
 $direction = FALSE;$ 

 $v_i = ReadPairVertex(cv_{xy}, v_{s1})$  // read a pair of  $v_{s1}$  from  $cv_{xy}$ ;
If ( $v_i \in PL$ ) {
   $v_i = v_{s1};$ 
Else
   $v_t = v_i; Q = Q + v_i; R = R - v_i$ 
}
While  $v_t \neq v_{s2}$  {
   $direction = Not\ direction$ 
  If ( $direction = TRUE$ ) {
     $v_t = ReadPairVertex(cv_{yx}, v_i);$ 
  Else
     $v_t = ReadPairVertex(cv_{xy}, v_i);$ 
  }
   $v_i = v_t; Q = Q + v_i; R = R - v_i$ 
}
}

```

Figure 4.6: Algorithm for Grouping Vertices

4.2.5 Implementation of the algorithm

In the implementation of the algorithm, the vertices of an orthogonal polygon are inputted randomly and saved in a file.

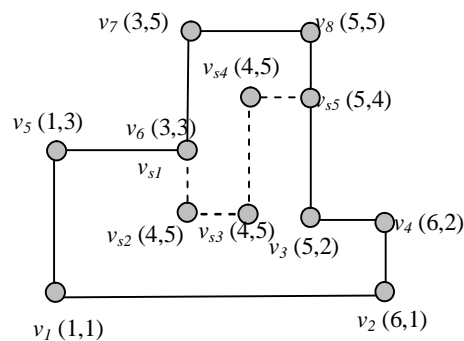


Figure 4.7: The Vertices of Orthogonal Polygon and a Polyline

(1) Add all vertices of the orthogonal polygon and non-ending vertices of the polyline into a set of combined vertices cv . Therefore $cv = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_{s2}, v_{s3}, v_{s4}\}$.

(2) The set of intersection vertices is $v_s = \{v_{s1}, v_7, v_{s5}, v_{s5}\}$. Each vertex in v_s is added to cv , and if the vertex already exist in cv then replace it by vertex in v_s .

(3) Group the vertices into two groups by using the procedure `GroupingVertices`. The results are $Q = \{v_{s1}, v_5, v_1, v_2, v_3, v_4, v_{s5}, v_{s4}, v_{s3}, v_{s2}\}$ and $R = \{v_8, v_{s5}, v_{s4}, v_{s3}, v_{s2}, , v_7\}$.

4.2.6 Time complexity analysis and discussion

The time cost is calculated for each of the following activities. Firstly, the time complexity for combining an orthogonal polygon and a polyline can be constructed in $O(n)$ where n is total number of vertices of an orthogonal polygon and a polyline. The last part of the algorithm, grouping vertices into two orthogonal polygons, cost a time complexity in $O(n)$. The sorting, which has $O(n \log n)$ in an average case, is not included to determine the complexity of these algorithms, because sorting is considered as inputs of the algorithm.

Meanwhile, splitting an orthogonal polygon with a line also takes a linear time, and the splitting still takes a linear time for k number of splitting lines. However, this method is not effective because there should be operations to combine some part of orthogonal polygon.

4.3 Splitting an Orthogonal Polyhedron Using a Polyplane

Orthogonal polyhedra are the 3D analogue of 2D orthogonal polygons. They are used in computational geometry as a well-known model to represent many real 3-dimensional objects.

Many different operations can be defined for an orthogonal polyhedron, for example: partitioning [82], splitting [56], and Boolean operations on arbitrary orthogonal polyhedra of any dimension [72], etc.

Splitting is one of operations in an orthogonal polyhedron, and a splitting has the purpose of splitting an orthogonal polyhedron into two halves. One of the splitting techniques has been introduced by Ayala by using a splitting plane that contains a single plane [56]. This splitting technique splits an orthogonal polyhedron into two halves, and each half consists of one or more orthogonal polyhedra.

Suppose a smaller orthogonal polyhedron will be taken out from a large orthogonal polyhedron. Of course, the Ayala technique can be used to get the smaller orthogonal polyhedron by applying this operation many times until the smaller orthogonal polyhedron is achieved. However, this technique seems ineffective, since it may be applied many times, and compound operation is probably needed to get the smaller orthogonal polyhedron. In the present section, a new method is established in which one splitting operation is sufficient to get the smaller orthogonal polyhedron from a large orthogonal polyhedron. The splitting operation uses a polyplane instead of single plane. This splitting method split an orthogonal polyhedron into two halves, and each half only contains one orthogonal polyhedron.

4.3.1 Definitions and terminology

Two orthogonal polygons are said to be *connected* if edges with the same length from each orthogonal polygon meet. An edge that belongs to two orthogonal polygons is called a *shared edge*; meanwhile, the ending point of a shared edge is called a *shared vertex*. Recall that a *degree of vertex* is the number of edges that meets at a vertex. Because each orthogonal polygon is parallel to one of three planes in Cartesian space, then the dihedral angle between two connected orthogonal polygons at a shared edge is either 90° or 270° . An *orthogonal polyplane* is a connected orthogonal polygon, but it is not a closed polygonal surface. The present study is restricted to an orthogonal polyplane that has the same length edges for each

shared edges. Henceforth, the term *polyplane* is used as shorthand for the orthogonal polyplane defined above.

As mentioned above, for any two orthogonal polygons with a shared vertex, the dihedral angle at the vertex is either 90° or 270° . Thus, the shared vertex has degree of three.

As an example, Figure 4.8 is a polyplane that contain three orthogonal polygons p_1 , p_2 , and p_3 . e_1 is a shared edge, shared by p_1 and p_2 . v_1 is a shared vertices that is shared by three edges e_1 , e_2 , and e_3 . Meanwhile, Figure 4.8(b) is not a polyplane. This is because the orthogonal polygon does not meet at the same length edges.

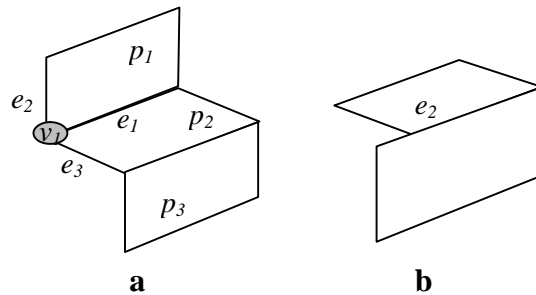


Figure 4.8: Valid (a) and not Valid (b) Instances of Polyplane

Figure 4.9 shows an orthogonal polyhedron having a polyplane. Numbers 1, 2 ... 28 in the figure are labels for vertices of the orthogonal polyhedron. Each vertex has a coordinate that represent its position in the 3D coordinate system. The gray planes represent a polyplane having three contiguous planes.

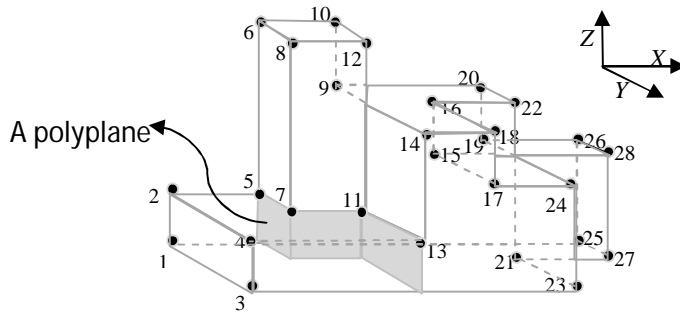


Figure 4.9: An Orthogonal Polyhedron with a Polyplane

Splitting an orthogonal polyhedron with a polyplane is defined as dividing the orthogonal polyhedron into two smaller orthogonal polyhedra by splitting the orthogonal polyhedron along the polyplane. For the splitting operation, conditions include:

1. A polyplane lies entirely in an orthogonal polyhedron.
2. A vertex of a polyplane intersects the boundary of an orthogonal polyhedron at a vertex, an edge, or a face of the orthogonal polyhedron. If a vertex of a polyplane intersects at a face of an orthogonal polyhedron, then the vertex has a degree of three.
3. Any shared edge of a polyplane cannot coincide with any edge or face of an orthogonal polyhedron.

From the above conditions, it can be identified that there are two kinds of vertex in a polyplane: *coalition vertex* that is a vertex of a polyplane that lies on a vertex of an orthogonal polyhedron, and *non-coalition vertex* that is a vertex of a polyplane that lies on edge, on a surface, or in the interior of an orthogonal polyhedron. Figure 4.10 shows some valid polyplanes and a not valid polyplane in orthogonal polyhedra.

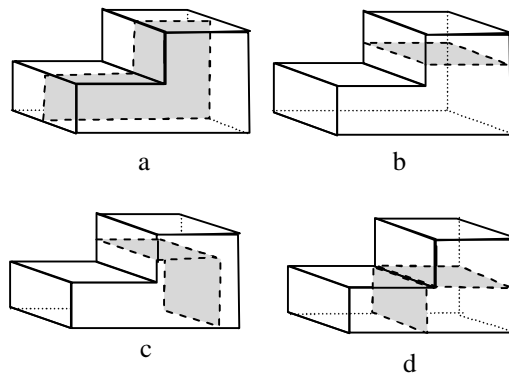


Figure 4.10: Valid (a,b,c) and not Valid (d) Polyplanes in Orthogonal Polyhedra

(i). The polyplane in Figure 4.10(a) is a valid polyplane that consists of an orthogonal polygon, and all the vertices of polyplane intersect (coincide) with the edges of the orthogonal polyhedron.

(ii) Figure 4.10(b) has one rectangle as a splitting polyplane that intersects with four edges of the orthogonal polyhedron and creates four intersection vertices.

(iii) The polyplane in Figure 4.10(c) is a valid polyplane that has two contiguous rectangles that intersect four edges, and the polyplane has two shared vertices.

(iv) The splitting plane in Figure 4.10(d) is not a valid polyplane, because a shared edge in the polyplane coincides with an edge of the orthogonal polyhedron; hence, it does not satisfy the rule of a polyplane intersecting an orthogonal polyhedron.

An orthogonal polyhedron is represented by its extreme vertices as discussed in Chapter 2. Recall that extreme vertices are the ending vertices of brinks in an orthogonal polyhedron, and a *brink* is defined as the longest uninterrupted line segment, built out of a sequence of collinear and contiguous two-manifold edges of an orthogonal polyhedron.

Meanwhile, a polyplane is represented by an open connected of planes, p_1, p_2, \dots, p_n , in which each plane is an orthogonal polygon. Representing an orthogonal polygon has been described in Chapter 2.

4.3.2 An algorithm of splitting an orthogonal polyhedron using a polyplane

This section will present an algorithm for splitting an orthogonal polyhedron by using a polyplane. The inputs of the algorithm are the extreme vertices of an orthogonal polyhedron and the vertices of a polyplane. The algorithm works in two main steps:

1. Combine the vertices of an orthogonal polyhedron and a polyplane into a set of combined vertices.
2. Group the set of combined vertices into two groups in which each group represents an orthogonal polyhedron.

Based on the above steps, there are two main procedures for splitting an orthogonal polyhedron using a polyplane. The first is a procedure for combining the vertices of an orthogonal polyhedron, which will be discussed in Subsection 4.3.3. The second is a procedure for grouping the combined vertices into two groups that represent two orthogonal polyhedra; this procedure will be discussed in Subsection 4.3.4.

4.3.3 Combining vertices

Combining vertices is a process to combine the vertices of an orthogonal polyhedron and a polyplane into a set of combined vertices. Properties of each vertex in a set of combined vertices are a vertex name, the vertex coordinate, and source of the vertex. The procedure to get a set of combined vertices rests on the following observations: i) each vertex of an orthogonal polyhedron is a vertex of one of the two smaller orthogonal polyhedra. Therefore, all the vertices of an orthogonal polyhedron are added to a set of combined vertices. ii) A polyplane will become the boundary of each orthogonal polyhedron after splitting; therefore, each vertex of a polyplane is added twice in such a way to the set of combined vertices. iii) A non-coalition vertex lies inside of an orthogonal polyhedron and it becomes a vertex in the

boundary of smaller orthogonal polyhedra. Hence, the vertex is added twice directly to the combined vertices. Meanwhile, if a coalition vertex exists, then update the vertex and its adjacent vertex as the same vertex in the combined vertices as the coalition vertex.

Based on the observation, the source of vertices in a set of combined vertices are orthogonal polyhedron vertices, coalition vertices, and non-coalition vertices.

Figure 4.11(a) shows an orthogonal polyhedron and a polyplane before splitting. Meanwhile, Figure 4.11(b) shows the set of combined vertices after splitting.

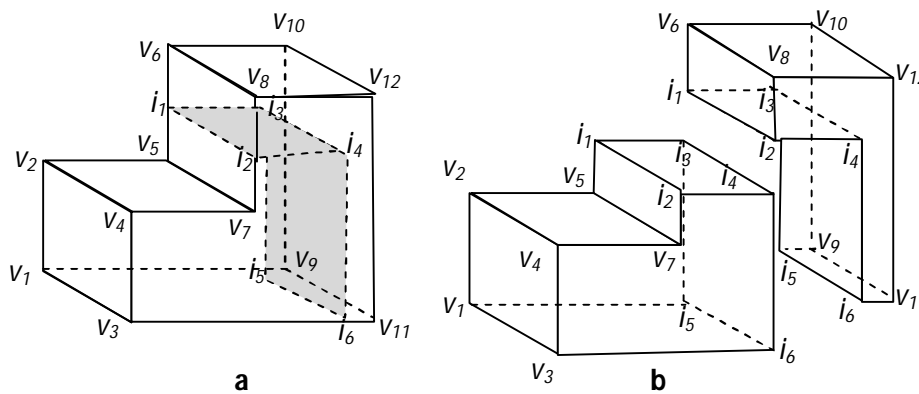


Figure 4.11: Splitting an Orthogonal Polyhedron with a Polyplane

The procedure `CVOPolyhedronPolyplane`, which is shown in Figure 4.12, is used to combine an orthogonal polyhedron and a polyplane. The inputs of this procedure are vertices of the orthogonal polyhedron and the vertices of the polyplane, and the output is a set of combined vertices. This procedure contains the function `ReadVertexPolyplane()`, which has the task of reading vertices in a polyplane.

```

Procedure CVOPolyhedronPolyplane(INPUT  $P$ : the extreme vertices of an
orthogonal polyhedron,  $PL$ : a polyplane; OUTPUT  $cv$ : a set of combined vertices)

var    $vp_i$    :  $i^{th}$  polyplane vertex
       $v$        : a vertex in  $cv$ 
       $m$        : number of vertices in a polyplane
 $cv = \emptyset$ 
 $cv = cv + P$ 

For ( $i=1$  to  $m$ ){ // a polyplane has  $m$  vertices
   $vp_i = \text{ReadVertexPolyplane}(PL)$  // function to read a vertex of PL
  If  $vp_i$  is a coalition vertex
     $cv = cv + vp_i + vp_i$ 
  else //  $vp_i$  is non-coalition vertex and  $v p_i \in cv$ 
     $\text{ReadVertexAdjacent}(vp_i, cv, v, v')$  // function to read  $vp_i$  in  $cv$ 
    replace  $v$  and its an adjacent vertex  $v'$  in  $cv$  with  $vp_i$ ;
}

```

Figure 4.12: Procedure CVOPolyhedronPolyplane for Combining Vertices

4.3.4 Grouping vertices

After combining the vertices, the next task is to separate the set of combined vertices into two groups of vertices in which each group represents an orthogonal polyhedron. To group the vertices into two orthogonal polyhedra rests on the following observations: (i) each vertex having source orthogonal polyhedron vertices in the set of combined vertices belongs to one of the two smaller orthogonal polyhedra. (ii) Each vertex having source non-coalition vertex in the set of combined vertices becomes a vertex in the boundary of each smaller orthogonal polyhedron, so each the same vertices is distributed to each smaller orthogonal polyhedron. Therefore, each shared vertex is added directly to each smaller orthogonal polyhedron without a separating process. (iii) Each vertex having source coalition vertex belongs to one of the two smaller orthogonal polyhedra.

A smaller orthogonal polyhedron has a polygonal surface, and the vertices on the polygonal surface come from the three sources of vertex in a set of combined vertices. Because the surface of an orthogonal polyhedron is a set of connected polygons, a *closed walk* can be

made from a vertex and back to the vertex after visiting all vertices in a smaller orthogonal polyhedron. There are two non-coalition vertices that have the same position, and each of them belongs to different orthogonal polyhedra. Thus, the non-coalition vertex is obviously a vertex of each smaller orthogonal polyhedron, and it is not necessary to be explored in a closed walk. Some non-coalition vertices and coalition vertices are adjacent to orthogonal polyhedron vertices, and some of them are surrounded by other non-coalition and coalition vertices. Therefore, the closed walk moves backwards when meeting these kinds of vertices.

The closed walk needs to visit vertices of the vertices having source orthogonal polyhedron, and move backwards when meets the coalition and non-coalition vertices. A closed walk starts from a starting vertex and terminates at the vertex. A coalition vertex and a non-coalition vertex that lies on edge of the original orthogonal polyhedron are also known as a *separating vertex*. A starting vertex is a vertex that is selected from any separating vertex. All the vertices in a closed walk and non-coalition are the complete vertices of a smaller orthogonal polyhedron.

The procedure for grouping vertices is based on by the following assumptions: i) each vertex having source orthogonal polyhedron vertices is visited three times, due to the vertices being extreme vertices related to three other vertices. ii) After visiting a separating vertex, the walk backwards to the previous visited vertex. iii) A vertex is visited in priority order, pair in a brink, pair on the same plane, and pair on the next plane.

Figure 4.13 is the procedure `GroupingVertices3D` for grouping the vertices in a set of combined vertices into two groups of orthogonal polyhedra. Inputs of this procedure are the set of combined vertices vc . Meanwhile, the outputs are Q and R , respectively an orthogonal polyhedron.

A closed walk starts with any separating vertex v_{sI} that is achieved using a function `ReadSeparatingVertex()`. The next step is determining the pair of v_{sI} , v_i , from the sorted of a set of combined vertices using a function `ReadPairVertex()`. Let v_t be the pair vertex of v_i , then repeat the function `ReadPairVertex()` until v_t is equal with the starting vertex v_{sI} . Once it exists, the two groups of vertices are achieved, and each group of vertices represents a small orthogonal polyhedron.

```

Procedure GroupingVertices3D(INPUT: cv combined vertices, cvxyz: cv in xyz-sorted, cvyxz: cv in yxz-
sorted, cvzxy: cv in zxy-sorted; Q,R: orthogonal polyhedra)

var    vsI           : a separating vertex
       vi, vt, vnc    : vertex in combined vertices cv
       dir,i         : integer
       staorthopoly  : a Boolean variable { TRUE if the source vertex is orthogonal polyhedron }
       visited       : a Boolean variable { TRUE a vertex is visited }

staorthopoly = false; visited= false; i=0
file(0) = cvxyz; file(1) = cvyxz; file(2) = cvzxy;

vnc = ReadNonCoalitionVertex(cv) // a function to read non-coalition vertices from cv
Q = vnc; R = cv - vnc;

// making a closed walk
vsI = ReadSeparatingVertex(cv) // a function to read a separating vertex
if (vsI ≠ null) {
    while (staorthopoly = false) {
        vi = ReadPairVertex (file(i), vsI) // read a pair of vsI from file(i)
        if (vi ∈ orthogonal polyhedron vertex){
            dir = direction(vsI,vi) // a function to determine the direction the brink (vsI,vi)
            staorthopoly = true
            if (vsI ∉ non-coalition vertices){
                Q = Q + vsI; R = R - vsI;
            }
        }
        else
            i=i+1;
    }
    While (vi ≠ vsI){
        dir = ChangeDirection(dir) //change direction of brink in the walk
        vt = ReadPairVertex(file(dir),vi)
        if (visited = false and vt ∉ non-coalition vertices){
            Q = Q + vt; R = R-vt; vi = vt;
        }
    }
}

```

Figure 4.13: Procedure GroupingVertices3D for Grouping the Combined Vertices

4.3.5 Implementation of the orthogonal polyhedron splitting algorithm

The following example illustrates the implementation of the algorithm for splitting an orthogonal polyhedron with a polyplane. Figure 4.14 is an orthogonal polyhedron having twelve vertices, and a polyplane splits the orthogonal polyhedron into two smaller orthogonal polyhedra.

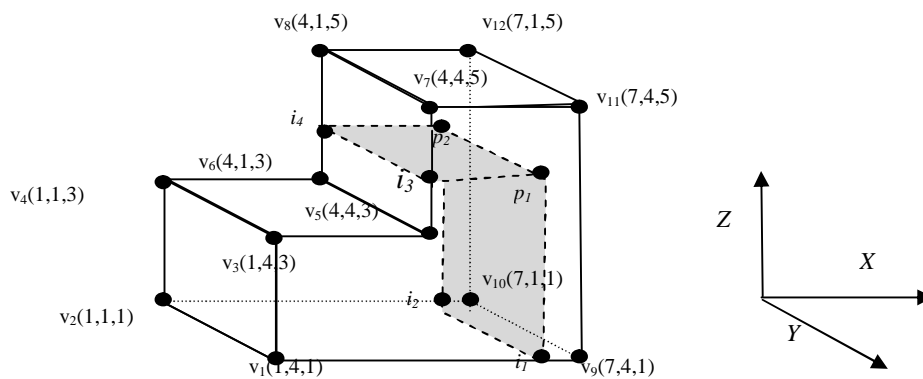


Figure 4.14: The Vertices of an Orthogonal Polyhedron after Splitting

(1) Add all vertices of the orthogonal polygon and non-ending vertices of the polyline into a set of combined vertices cv by using the procedure `CVOPolyhedronPolyplane`.

Therefore $cv = \{v_2, v_4, v_1, v_3, v_6, i_4, i_4, v_8, v_5, i_3, i_3, v_7, i_2, i_2, p_2, p_2, i_1, i_1, p_1, p_1, v_{10}, v_{12}, v_9, v_{11}\}$.

(2) The set of non-coalition vertices $v_{nc} = \{i_3, i_4, p_1, p_2, i_1, i_2\}$, and the set of coalition vertices is empty.

(3) Apply the procedure `GroupingVertices` to group the vertices into two group vertices. Each of them represents an orthogonal polyhedron. i_4 is a separating vertex, and let it be a starting vertex in the closed walk. v_8 is a pair of i_4 in xyz -sorted, because $i_4 v_8$ is a brink that is parallel to the Z -axis. The resulting closed walk is $i_4, v_8, v_7, i_3, v_{11}, v_9, v_{10}, v_{12}, i_2$, and i_1 as shown in Figure 4.15.

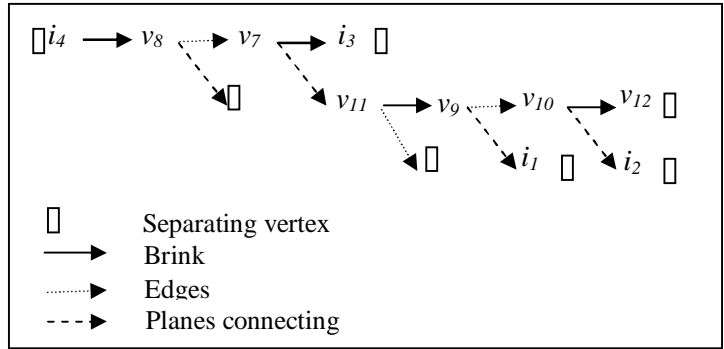


Figure 4.15: A Walk for Grouping Vertices

The closed walk vertices without non-coalition are $v_8, v_7, v_{11}, v_9, v_{10}$, and v_{12} . After combining the non-coalition vertices and the closed walk vertices without non-coalition vertices, a smaller orthogonal polyhedron vertices is $Q = \{i_4, v_8, i_3, v_7, i_2, p_2, i_1, p_1, v_{10}, v_{12}, v_9, v_{11}\}$, and they are presented in XYZ-sorted order. The remaining of vertices are allocated to R .

4.3.6 The Time complexity analysis and discussion

Let n be the number of vertices of a given orthogonal polyhedron and a polyplane, then the time cost for splitting the orthogonal polyhedron using the polyplane is calculated as the following activities:

- ABC-sorted vertices in the set of combined vertices has the time complexity $O(n \log n)$ in which the ABC-sorted applies Quick-sort method [83].
- The time complexity for combining vertices of an orthogonal polyhedron and a polyplane is $O(n)$.
- Grouping the set of combined vertices into two smaller orthogonal polyhedra needs: i) $O(n)$ to put each non-coalition vertex to each orthogonal polyhedron, ii) $O(n)$ time to read the first separating vertex, iii) $O(n)$ for making a closed walk from the first separating vertex and back to the first separating vertex, in which each vertex in the walk is visited three times.

From the above list, sorting the set of combined vertices in ABC-sorted is the most time consuming operation. Therefore, the time complexity for splitting them using a polyplane is $O(n \log n)$.

4.4 Splitting Polyhedra in a Bounding Box

A complex shape of a polyhedron is often a reason for dividing the polyhedron into simpler shapes to better facilitate representing and performing a variety of operations on the shape. There could be a variety of different operations to do this task. For example: decomposition [84], or splitting [85]. Decomposition is an operation to divide a polyhedron into a set of simpler polyhedra. One of the possible problems in decomposition operation is how to decompose a given polyhedron into a minimum number of tetrahedra. Splitting is another kind of operation to split a polyhedron into two partitions.

Instead of dividing a complex-shape polyhedron, dividing a polyhedron based on a view point often happens in the real life, but not much study has been focused on it. The purpose of this kind of operation is to separate a visible and an invisible area of a polyhedron from a view point outside of the polyhedron. It is a quite interesting operation, because a set of polyhedra bounded by a bounding box can be split simultaneously into two groups: visible and not visible polyhedra.

Given a set of convex polyhedra P_i in a bounding box B and a view point v that sees partially B in an orthogonal direction, what is the efficient procedure to split P_i in B from v ? The algorithm has two main steps; compute the splitting plane derived from a view vertex, and if the splitting plane intersects with a polyhedron then split each of the affected polyhedra into two polyhedra.

4.4.1 Definitions and terminology

A bounding box is an orthogonal prism, and it is composed by one or more polyhedra. Thus, if there is only a polyhedron in a bounding box then the polyhedron is an orthogonal polyhedron (a polyhedron in which each edge is parallel to any orthogonal direction). The bounding box definition implies that all properties of polyhedra lie entirely in the bounding box.

The boundary of bounding box consists of six facets that can be divided into a solid facet and an arbitrary facet. A facet is called a solid facet if any two points at each side of the facet cannot see each other. Meanwhile, a facet is called arbitrary facet if any two points at each side of the facet can see each other. In this thesis, it is assumed that, for any bounding box, at least one facet is a solid facet; meanwhile, at most five facets are arbitrary facets. A point v cannot see another point p inside a bounding box if a solid facet intersects the segment line pv , and v can see p if they intersect with an arbitrary facet.

A view point is a vertex outside of a bounding box that has the task of seeing an area in the bounding box. Due to the existence of at least one solid facet on the bounding box then a view point sees the bounding box partially. A bounding box is visible from one view point if at least a point in the interior of an arbitrary facet is visible from the view vertex. A view point v , as shown in Figure 4.16 will create an arbitrary plane in a bounding box if v sees at least four different points on one of the bounding box facets, and v is not in the same plane horizontally or vertically with the visible vertices.

An orthogonally view point is a view point in which the segment line to the closet point in bounding box is parallel to one of the three orthogonal directions. A closet point is a bounding box vertex that has the smallest distance to a view point. If a view point sees a bounding box partially then the bounding box will be separated into regions in terms of v ,

namely visible area and invisible area. The visible area is separated by a cutting plane from the invisible area, and a cutting plane is also known as a splitting plane. The planes that contain s_0, s_1 , and q in Figure 4.16 are examples of splitting planes.

The splitting plane S splits a bounding box B into two partitions, and it intersects four edges of B . There are three possibilities of a view point to cut a polyhedron from an orthogonal view point as shown in Figure 4.16. The possibility is dependent on the existence and position of a gate point that is a point in segment line that lies between the view vertex and a closet vertex in B . If a closet vertex is visible from a view point, then gate point does not exist; otherwise, a gate point exists. Point g in Figure 4.16(b) is a gate point. If a gate point does not exist, then a splitting plane starts from the closet vertex to v ; otherwise, a splitting plane starts from the intersection of line vg with an edge of the closet facet to v .

Figure 4.16 also gives the three possible ways of a splitting plane to cut a bounding box. Figure 4.16(a) is a bounding box having a splitting plane that derived from the view point v . The splitting plane in the bounding box in Figure 4.16(b) has two planes that derived from view point v . v sees the bottom partially because the line vq is blocked by another object.

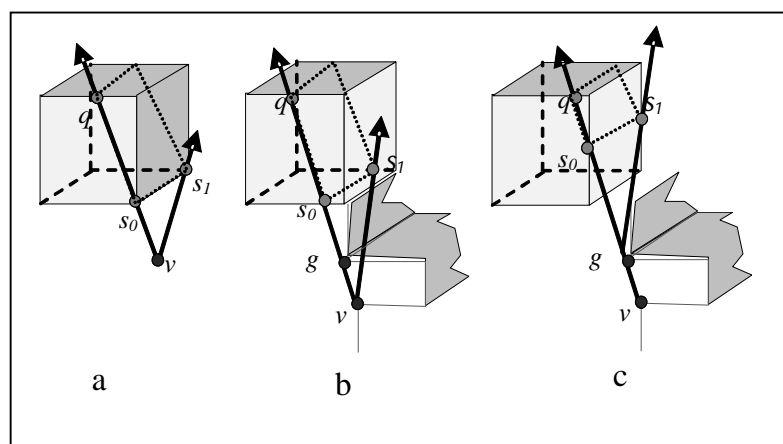


Figure 4.16 One Edge of B is Shared by S in (a);
There is no Edge of B Shared by S in (b) or (c)

A non-convex polyhedron can be decomposed into convex polyhedra. For a polyhedron P with n edges and r notches (features causing non-convexity in polyhedra), the algorithm produces a worst case optimal $O(r^2)$ polyhedra in $O(nr^2 + r^{7/2})$ [86].

Bajaj and Pascucci proposed a locality-based algorithm for splitting a complex polyhedron with a hyperplane h . The algorithm is divided into three phases: (i) in the first phase, primary numerical computations are performed to classify vertex positions with respect to h ; (ii) in the second phase, symbolic manipulations return the topological structure of the result; (iii) in the final phase, secondary numerical computations are used to detail the geometric structure of the result. [87].

4.4.2 The algorithm

The algorithm to split a polyhedron P in a bounding box B by a splitting plane S that is derived from a view point v contains two main steps:

1. Compute points that form the splitting plane equation.
2. Split each polyhedron that is intersected by a splitting plane into two polyhedra.

In the subsequent sections, some theories to support this algorithm are reviewed such as data structure for a polyhedron, primitive operations, and the theory of intersection line and plane. Finally, procedures are developed for performing some tasks, such as: forming a splitting plane from a view point, calculating the intersection between a plane and line segments, and splitting a polyhedron.

Let P be a simple polyhedron having n vertices: $\{v_1, v_2, \dots, v_n\}$, m edges: $\{e_1, e_2, \dots, e_m\}$, and q facets: $\{f_1, f_2, \dots, f_q\}$. P is represented by a collection of vertices, edges and facets. The list of vertices, edges, and facets of P are stored similarly to the star-edge representation of polyhedra as discussed in Chapter 2. To create some main procedures, it is better to have an

insight about some preliminary procedures that are known as primitive operations. Figure 4.17 shows the definitions of some primitive operations.

```

FUNCTION InitPolyhedron() RETURN polyhedron
{Returns an empty polyhedron}
PROCEDURE ReadFacet(INPUT P: polyhedron; OUTPUT f : facet)
{Reads next facet from a polyhedron P}
PROCEDURE ReadEdge(INPUT f: facet; OUTPUT e : edge)
{Reads next edge (pair of vertices) from a facet f}
PROCEDURE PutEdge(INPUT vb,ve: ending vertices of an edge; OUTPUT P: polyhedron)
{Appends to a polyhedron P an edge having end vertices vb and ve}
PROCEDURE IsVisible(INPUT p,v: point in B and view point; OUTPUT status: Boolean
variable whether p is block by v.)
FUNCTION RitTo() RETURN group of polyhedron
{Return Q if the last read edge belong to Q, or R for the other case}
PROCEDURE ReadBoxPlane(INPUT B: bounding box; OUTPUT v1,v2,v3: vertices of B)
{Read three vertices from each plane of B}

```

Figure 4.17: Primitive Procedures and Functions for Splitting in a Bounding Box

4.4.3 Intersection line and plane

A splitting plane is formed from three points, and has the general equation $Ax + By + Cz + D = 0$ [88]. Given three points — $s_0(x_1, y_1, z_1)$, $s_1(x_2, y_2, z_2)$ and $q(x_3, y_3, z_3)$ — then the coefficients of the splitting plane equation are formulated as Equations (4.1), (4.2), (4.3), and (4.4) as follows:

$$A = y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2) \quad (4.1)$$

$$B = z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(y_1 - y_2) \quad (4.2)$$

$$C = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \quad (4.3)$$

$$-D = x_1(y_2 * z_3 - y_3 * z_2) + x_2(y_3 * z_1 - y_1 * z_3) + x_3(y_1 * z_2 - y_2 * z_1) \quad (4.4)$$

An edge of an orthogonal polyhedron has two ending points. Let $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$ be the edge-ending points. So, by using a parameter u , a line equation for the edge P_1P_2 is $P = P_1 + u(P_2 - P_1)$ where P_1 and P_2 are vectors in R^3 [88]. The intersection points should lie on

the line and the splitting plane as well. Hence, to get the intersection point, substitute P to the plane equation to get the following formula:

$$\mathbf{u} = \frac{A(x_1) + B(y_1) + C(z_1) + D}{A(x_1 - x_2) + B(y_1 - y_2) + C(z_1 - z_2)} \quad (4.5)$$

The coordinate of intersection points are:

$$x = x_1 + \mathbf{u}(x_2 - x_1); y = y_1 + \mathbf{u}(y_2 - y_1); z = z_1 + \mathbf{u}(z_2 - z_1) \quad (4.6)$$

These concepts are very useful to find the intersection point in some relevant tasks in this thesis, such as an intersection between a line segment from a view point and a plane on the bounding box, and intersection between an edge of a polyhedron and a splitting plane. Constants A , B , C and D are processed by a primitive operation *DetermineConstantofPlane* procedure as follows:

PROCEDURE DetermineConstantofPlane(INPUT v1,v2,v3; OUTPUT A,B,C,D)

Meanwhile, the constant \mathbf{u} is processed by a procedure that defines as:

PROCEDURE DetermineConstantofLine(INPUT v,so; OUTPUT \mathbf{u})

4.4.4 Computing a splitting plane

Instead of splitting a bounding box, splitting plane simultaneously splits an affected polyhedron into two polyhedra. A splitting plane S is formed by three points that lie on a bounding box, and three points are necessary to make a plane equation. Two points of S lie on a segment line that is derived from a view point. If the closet point p_0 of B is visible from v then make a segment line from v through p_0 until it intersects with another edge of B . But if the closet point p_0 is not visible from v then make a segment line from v through a gate point

g until it intersects with two edges of B at points p_0 and p_1 , respectively. See Figure 4.16 to have a precise understanding about the splitting plane and the supporting points.

The task to determine the closet vertex of bounding box to the view point is quite simple. Let $B_i(x_b, y_b, z_b)$ be a vertex in B , and let $V(x_v, y_v, z_v)$ be a view point, then the distance between V and B is determined by the following formula:

$$D = \sqrt{(x_v - x_b)^2 + (y_v - y_b)^2 + (z_v - z_b)^2} \quad (4.7)$$

A procedure to get the smallest D value is written as follows:

PROCEDURE theClosestVertex(INPUT Vi: vertices B; OUTPUT D: a vertex)

Thus, the splitting plane procedure is described as follows:

- Make a segment line s from the view point v to the shortest visible vertex s_0 of a bounding box B , and extend s until it hits the next edge of B at q . If the vertex s_0 is not visible from v , make the segment line through the gate point g until it intersects the first edge at s_0 and the next edge at q .
- Assign a point s_1 such that the edge s_0s_1 is perpendicular to segment line vs_0 , and establish the plane equation $Ax + By + Cz + D = 0$ through s_0, s_1, q .

In detail, the procedure to compute the splitting plane is described in Figure 4.18.

```

PROCEDURE: ComputingSplittingPlane (INPUT:  $v$  coordinate of a view point, eight coordinate of the
bounding box  $B$ ,  $g$  coordinate of gate point; OUTPUT: list of splitting plane's coordinates
VAR
     $s_0, s_1, q, v_1, v_2, v_3$  : points
    staVis, staIntsc : Boolean variable
     $A, B, C, D, u$  : constants
ENDVAR
theClosestVertex(view point  $v$ , list of  $B$  coordinate,  $s_0$ )
{read the visible vertices, determine the closet vertex  $s_0$ }
IsVisible( $v, s_0$ , staVis) { $s_0$  is visible from  $v$  if  $vs_0$  does not intersect with another boundary}
If staVis =FALSE THEN GatePoint( $v, g, s_0$ ) ENDIF
ReadBoxPlane( $B, v_1, v_2, v_3$ )
WHILE BoxPlane NOT EOF DO
    DetermineConstantaofPlane( $v_1, v_2, v_3, A, B, C, -D$ )
    DetermineConstantaofLine( $v, s_0, u$ )
    IntersectionPlaneLine( $v, s_0, u, q$ , staIntsc)
    IF staIntsc =TRUE THEN EXIT WHILE
    ReadBoxPlane( $B, v_1, v_2, v_3$ )
ENDWHILE
FindOtherPoint( $s_0, q, s_1$ )

```

Figure 4.18: Algorithm ComputingSplittingPlane for Computing a Splitting Plane

4.4.5 Calculating the intersection points on an edge of a polyhedron

An intersection point on a polyhedron is a point at which a splitting plane intersects with an edge of a polyhedron. A set of intersection points forms a new facet that will split the polyhedron into two polyhedra. To get the intersection point, determine a point that lies both on the edge of the polyhedron and the splitting plane. The main steps are:

- Input points that will form a splitting plane.
- Input an edge of a facet of the polyhedron
- Calculate constants A, B, C, D and u to get the intersection value and intersection *status*

The corresponding Intersection algorithm can be stated as the following figure.


```

PROCEDURE: Intersection
Input   s0,s1,q: Splitting Plane points, eii: edge
Output vm : vertex, stalntsc: Boolean variable
Var
    A,B,C,D      : constants
    p1,p2       : points
ENDVAR
Readpoint(eii,p1,p2)
DetermineConstantaofPlane(s0,s1,q,A,B,C,-D)
DetermineConstantaofLine(s0,s1,u)
Intersect(p1,p2,u,vm, stalntsc)

```

Figure 4.19: Algorithm IntersectionPoint for Determining Intersection Points

4.4.6 Splitting a polyhedron into two polyhedra

This section presents a procedure for splitting of a polyhedron *P* against a splitting plane *S*. It computes two resulting polyhedra *Q* and *R*, respectively.

Splitting *P* along *S* is carried out by splitting facets which are intersected by *S*. Suppose *f_i* is such a facet which is to be split at *v₁ⁱ, v₂ⁱ, ..., v_kⁱ* that lie on the edges *e₁ⁱ, e₂ⁱ, ..., e_kⁱ* respectively where *k* is the number of intersection point in facet *i*.

The splitting process is started by reading the first facet *f₁* of *P*, and then followed by reading an edge *e₁¹* on *f₁*. *e₁¹*, which contains two end points *v_b* and *v_e* is evaluated by procedure *Intersection* that has inputs *v_b* and *v_e* and the splitting plane *S*. The output of this procedure is Boolean variable *status* and intersection point *v_m*. If *status* = TRUE, then put *v_b* and *v_m* in *Q* where *v_m* is a new vertex that lies between *v_b* and *v_e*, and put *v_m* and *v_e* in *R*. If *status* = FALSE, then there is no new point, and put *v_b* and *v_e* in *Q*.

Edges on a facet have a direction, because the facet has a cycle of edges. It means that the second vertex (*v_e*) on a previous edge becomes the first vertex (*v_b*) on the next edge. Hence

for the next edges, the group of polyhedra is determined by the group of v_b in the previous edges. A function $RltTo()$ is used to determine the group of edges.

Figure 4.20 shows two facets f_1 and f_2 in the same polyhedron. f_1 has a cycle of edges e_1, e_2, e_3 and e_4 . e_1 has two end points v_b^1 and v_e^1 . v_b^1 is written instead of v_b just to clarify that the vertex is the beginning point of e^1 ; however, for a general case, it is shorten as v_b only. f_1 does not intersect with a splitting plane, so all vertices, edges and face are grouped as Q . Meanwhile f_2 intersects with a splitting plane at p_1 and p_2 , then edges that have at least one ending point at Q are grouped as Q ; otherwise, they are grouped as R .

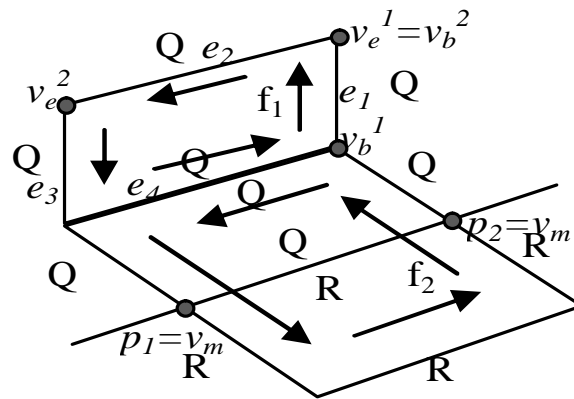


Figure 4.20: Grouping Polyhedra

According to the above explanation, the corresponding splitting polyhedron procedure can be stated as Figure 4.21.

```

PROCEDURE SplittingPolyhedron (INPUT P: polyhedron, S: splitting plane; OUTPUT Q, R :
polyhedra
VAR Vb, Ve, Vm: point of polyhedron   ENDVAR
Q:= InitPolyhedron(); R:=InitPolyhedron();
ReadFacet(P,fi)
WHILE fi NOT EOF DO
ReadEdge(fi,ej1,vb,ve)
  WHILE ej1 NOT EOF DO
    Intersect(S,fi, ej1,vb,ve,Vi,statusIntersect)
    IF statusIntersect = FALSE
      RltTo()
      PutEdge(Ve,Ve,Q,R)
    ELSE
      RltTo()
      PutEdge(Vb,Ve,Vm,Q,R)
    ENDIF
    ReadEdge(fi,ej1,vb,ve)
  ENDWHILE
ReadFacet(P,fi)
ENDWHILE

```

Figure 4.21: Algorithm `SplittingPolyhedron` for splitting polyhedron

If a bounding box has p polyhedra, then apply *splittingpolyhedron* procedure p times.

4.4.7 Implementation of the algorithm

The example below would explain the implementation of the algorithm. Let B be a bounding box, P_1 and P_2 are polyhedra in B , and v be a view vertex. See Figure 4.22(a) to describe B , P , and Figure 4.22(b) to describe v .

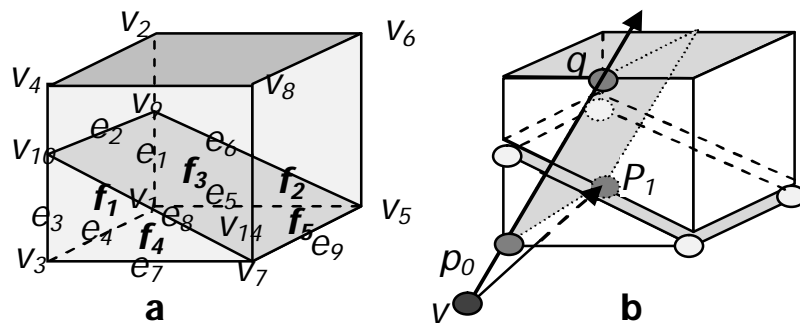


Figure 4.22: Illustration of Splitting Polyhedron in a Bounding Box

1) There are two polyhedra in B . Polyhedron P_1 contains list of vertex, edge E , orientation O , and facet F as follows:

$V = \{v_1, (0,0,0), (e_1, e_4, e_5), v_9, (0,0,2), (e_1, e_2, e_6), v_3, (0,4,0), (e_3, e_4, e_7), v_{10}, (0,4,2), (e_2, e_3, e_8), v_5, (4,0,0), (e_5, e_6, e_9), v_7, (4,4,0), (e_7, e_8, e_9)\}$,
 $E = \{e_1, (v_1, v_9, o_1), e_2, (v_9, v_{10}, o_2), e_3, (v_3, v_{12}), e_4, (v_1, v_3), e_5, (v_1, v_5), e_6, (v_5, v_{10}), e_7, (v_3, v_7), e_8, (v_7, v_{11})\}$,
 $O = \{o_1, e_1, f_1, e_2, v_1, v_9\}, \{o_2, e_2, f_1, e_3, v_9, v_{10}\}$, etc},
 $F = \{f_1, (e_1, e_2, e_3, e_4), f_2, (e_1, e_5, e_6), f_3, (e_8, e_2, e_6, e_9), (f_4, (e_3, e_8, e_7), f_5, (e_4, e_7, e_9, e_5)\}$;

Polyhedron $P2$ contain $V = \{v_{10}, v_2, v_{12}, v_4, v_{13}, v_6, v_{14}, v_8\}$.

2) Compute a splitting plane from the view point v that has coordinates $(-0.5, 4, -1)$ by using *ComputingSplittingPlane* procedure. The closet point to v is v_3 , so relabel v_3 as s_0 , and the line equation trough v and s_0 is $L = (-0.5, 4, -1) + u(0.5, 0, 1)$. The line vs_0 intersects the edge of bounding box B on the plane $z=4$ at coordinate $q(2, 4, 4)$.

3) Find the third point on edge or point of B . It is quite simple to get this point. Let s_1 be the third point then segment line s_0s_1 must be perpendicular to s_0q . This means s_1 shares a plane with s_0 , but not with q . The next step is making a plane through the points s_0, q , and s_1 (a corner of B not having the same plane with s_0 and q). By using Formula (1), the splitting plane equation S is $2x - z = 0$.

4) Apply procedure *SplittingPolyhedron* to do splitting. Check all facets and determine whether the facet f_i is intersected by the S or not. For example: check edges on the facet f_1 that has e_1, e_2, e_3, e_4 edges in cycle order. Use *intersection* procedure to get the *status* of intersection. If *status* is TRUE then the program will proceed to calculate intersection points p_1 and p_2 . The return value of function *RltTo* is Q for the first facet; hence, the list of edges, (e_1, e_2, e_3, e_4) belong to Q.

5) From the above algorithm, the next facet nf depends on a sharing edge with the previous facet pf . The group of the first edge in nf is the same as that of the last edges in pf . If the last edge in pf is visited then visit the last two edges. This method guarantees that each edge belongs to the correct polyhedron.

6) Thus, the result of splitting polyhedra is shown in the Figure 4.23. Polyhedron is partitioned into polyhedra Q and R , see Figure 4.23(b). There is a new facet f_{12} , and it is a property of Q and R .

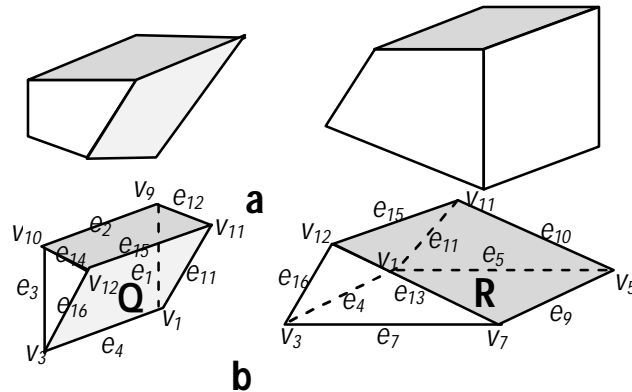


Figure 4.23: The Resulting of Splitting Polyhedra

4.4.8 The time complexity analysis and discussion

The time cost is calculated for each of the following activities:

The points that form a splitting plane from a view point are computed as the following steps. First, calculate the distance between each vertex in B and v ; second, determine the closet point to v . It takes $O(1)$, because there are six vertices of B . Second, find the intersection points between a plane of B and line that formed by v or g . This step has to proceed with an iteration at the most six times, and the cost is $O(1)$.

The time complexity of splitting polyhedron procedure is determined by the number of facets and edges on the polyhedron. Let f be number of facets and m be the maximum edges in any facet, then the time complexity for splitting polyhedron is $O(fm)$. Overall, the time complexity of the algorithm for splitting polyhedron in bounding box from a view vertex is $O(1) + O(fm) = O(fm)$.

The time complexity for splitting polyhedron and orthogonal polyhedron is linear, as shown in [87] and [56], respectively. From the above algorithm, fm is the number of inputted facets. Hence, the time complexity is linear as well. There is nothing improved in terms of time complexity; however, the proposed algorithm is suitable for separating a visible and an invisible area of a polyhedron from an outside view point of the polyhedron, and splitting several polyhedra simultaneously in bounding a box area.

4.5 Summary

This chapter has developed an algorithm for splitting an orthogonal polygon with a polyline, in which the algorithm works in two main steps: combining the vertices of an orthogonal polygon and a polyline, and grouping vertices into two smaller orthogonal polygons.

This chapter has also developed an algorithm for splitting an orthogonal polyhedron with a polyplane. The algorithm involves two main steps, combining the vertices of an orthogonal polyhedron and a polyplane, and grouping vertices into two smaller orthogonal polyhedra.

Finally, this chapter has developed an algorithm for splitting polyhedra in a bounding box, in which the splitting plane passes through a view point that partially sees the bounding box in an orthogonal direction. The developed algorithm has two main steps: compute the splitting plane derived from a view vertex, and if the splitting plane intersects with a polyhedron, then split each of the effected polyhedra into two polyhedra.

The next chapter will develop an algorithm for guard placement for an orthogonal pseudo-polyhedron.

CHAPTER 5

PLACEMENT OF FIXED-POINT GUARDS IN AN ORTHOGONAL PSEUDO-POLYHEDRON

In this chapter, procedures are developed for point guard placement to monitor the interior of an orthogonal pseudo-polyhedron (OPP).

As mentioned in Chapter 1, in the Art Gallery problem, given a polygonal gallery and the goal is to guard the gallery's interior with a number of guards that must be placed strategically on edges, or on corners, or any point inside of the gallery. A gallery is in a 3-dimensional space, but its floor outline usually has enough information to place the guards. Therefore, the art gallery is modelled as a polygon.

Earlier work on the art gallery problem assumed that the floor outline of any building provides the sufficient information for monitoring the building. However, such a floor outline does not always give us adequate information about the complex spatial structure of the building. In many applications, the knowledge of spatial structure of the building is essential in deciding how the building should be monitored. Therefore, it is necessary to take into account the spatial information on the environment to determine the guard placement. We call this version of art gallery problem *the 3-Dimensional Art Gallery Problem*.

An art gallery is modelled by an OPP, because this shape arises frequently in practice and deserves special attention due to the fact that most real life buildings and art galleries are largely orthogonal shaped.

Work in 3D-AGP is less extensive. Bose *et al.* [33] and Urrutia [6] considered mobile guards along the edges to monitor the exterior of a polyhedron. Recently, Souvaine *et al.* [34]

introduced face guards: guards that roam over an entire interior face of a simple polyhedron, and they also established bounds for the number of face guards that are necessary and sufficient to observe the interior of a simple polyhedron and a simple orthogonal polyhedron.

In contrast to work of Souvaine *et al.*, this thesis consider guarding the interior of OPP using fixed point guard, not using moving guards. A reason of using fixed guards is related to a practical application in which most of cameras in buildings (*e.g.*, art galleries, banks, and supermarket) work in fixed point. In other words, using moving guards for applications are not adequate, but fixed guard are fine. In this work, a procedure is developed for calculating the guard placement in which the guards are placed in any point in an OPP.

Partitioning is a useful first step for successful guard placement. However, compared to partitioning a polygon, partitioning a polyhedron is a lot more complex, *e.g.*, not all non-convex polyhedra can be tetrahedralized [61], and the number of tetrahedra in a tetrahedralization of a given polyhedron is not unique [62]. Therefore, it is important to find a partition scheme for orthogonal polyhedra in order to solve the 3D-AGP. One possible scheme is by decomposing a given OPP into a number of rectangular prisms instead of tetrahedralization.

Once an OPP is decomposed into a set of rectangular prisms, then a guard can be deployed to monitor a rectangular prism. Of course, each guard also monitors several rectangular prisms. To get a smaller number of guards, the 3D-AGP can be solved by transforming the problem into the Minimum Set Cover (MSC) problem. The MSC problem is defined as follow: given a universe U of elements and a collection S of (non-empty) subsets of U , and the goal is to find the smallest of a subset $S' \subseteq S$ which covers U [89].

In this chapter, a method is developed for point guard placement in an OPP. The key to this method is the mapping of the 3D-AGP into a Minimum Set Cover (MSC) problem. The method has three main steps: (1) decompose a given orthogonal polyhedron into a set of rectangular prisms, (2) construct a visibility subset for each corner point, and (3) map the 3D-AGP into a MSC problem. To implement this method, a number of definitions and terminologies are required, and they will be introduced in Section 5.1.

A number of procedures are developed for support guard placement method, namely procedure for decomposing an OPP and procedure for construction visibility subset. These procedures will be discussed in Section 5.4 and 5.5. Section 5.6 shows how to convert the 3D-AGP into MSC problem.

After discussing the time complexity of algorithm for guard placement in Section 5.8, a new algorithm for guard placement is developed in Section 5.9. This new algorithm has a purpose to reduce the number of guards that is achieved by the previous guard placement's algorithm.

5.1 Terminology and Related Research

The complex shape of a polyhedron is often the reason for dividing the polyhedron into simpler shapes to make it easy to performing a variety of operations on the polyhedron. One such operation is OPP partitioning, which is the process of decomposing an OPP into a set of rectangular prisms that do not intersect each other except on their boundaries (see Chapter 3). Each partition is called a piece of the original OPP. The symbol Π is used to represent the collection of all orthogonal prisms created from the partitioning of the OPP.

For each rectangular prism, there are eight corner points. Each corner point either corresponds to a vertex of the original OPP, or to an interior point of the original OPP. The former is called a vertex and the latter is called a partition point. Figure 5.1 depicts an OPP

which is partitioned into nine rectangular prisms. After the partitioning, v_1 is a vertex, and u_1 is a partition point. Both v_1 and u_1 are also corner points of the same rectangular prism. In addition, ρ_1 and ρ_2 are two pieces in the partition.

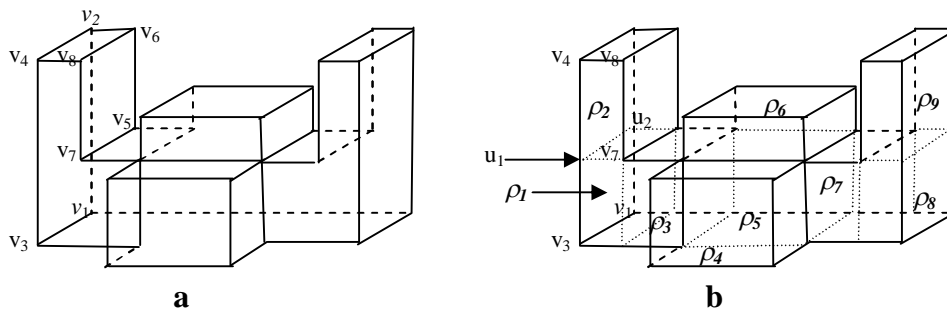


Figure 5.1: (a) an OPP, (b) Partitioning of the OPP

A guard can be placed at a certain point in an OPP to monitor the interior of the OPP. Each guard is capable of monitoring some parts of the OPP. To facilitate discussion, the following terms, some of which were introduced in [23], are defined.

Definition 5.1. Two points x and y in an OPP are said to be visible from each other if and only if the segment xy does not intersect the boundary of the OPP.

Definition 5.2. Let c be a point of an OPP, the visibility region of c , denoted $Vr(c)$, is the set of points of P that are visible from c .

Definition 5.3. A piece ρ of an OPP 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 .

In this thesis, point guards are used instead of vertex guards because Seidel has proved that there exist orthogonal polyhedra which cannot be fully monitored even if a guard is placed at each and every vertex of the orthogonal polyhedra [4].

One can expect that placing a guard at each and every vertex of a polyhedron would cover the entire interior of the polyhedron. This would only be obvious if every polyhedron is tetrahedralizable. For then, every tetrahedron would have a guard in a corner and the guards in these tetrahedra would cover the whole interior of the polyhedron. In the absence of tetrahedralization, however, it would be less clear whether the interior is still completely covered by these vertex guards.

Seidel gave an example of an orthogonal polyhedron in which guards placed at every vertex do not fully cover the interior, and he then provided an upper bound of the minimum number of guards for monitoring that special type of orthogonal polyhedra and stated that $\Omega(n^{3/2})$ guards are necessary, where n is the number of vertices in the orthogonal polyhedron [4].

If each rectangle prism is allocated one point guard, the whole OPP will be fully monitored by these point guards. However, placing a guard inside every rectangular prism seems an overkill. A guard placed on a partition point, for example, will monitor at least two rectangle prisms. This indicates that there is a method to reduce the number of such points for placing guard so that all the rectangular prisms are totally covered.

Recall that in Minimum Set Cover problem, it is given a universe U of elements and a collection S of (non-empty) subsets of U . The task is to find the minimum cardinality of a subset $S' \subseteq S$ which covers U [89].

The solution to the MSC problem, i.e., the subset $S' \subseteq S$ with the minimum cardinality that covers U gives us valid guard placements, and the cardinality of S' can be interpreted as the number of guards for guarding an orthogonal polyhedron.

The Minimum Set Cover problem has important application in areas such as rail way and air line scheduling [90], logical analysis of data [91], and species differentiation [92]. Although

MSC problem is proven to be NP-hard [89], there are several practical solutions for that problem such as linear programming approach, greedy algorithm and backtrack algorithm.

A trivial solution to the MSC problem required $O(m2^n)$ time where m is the number of elements in U and n is the number of subsets in S . Another solution of MSC problem is by disposing the problem into the Dominant Set Problem [89]. It is out of the scope of this thesis to try to give a comprehensive list of all effective algorithms for MSC including both heuristic approach and exact algorithm. Interested readers are referred to the survey by Caprara *et al* [93] and Buezas [92].

5.2 The Point Guards Placement Algorithm

In this section, an algorithm on how to place guards in a given orthogonal polyhedron is described. The following steps are used to generate a guards' placement:

- Step 1. Partition the orthogonal polyhedron into a set of rectangular prisms.
- Step 2. Construct a visibility subset of each corner point of each rectangular prism.
- Step 3. Map the 3D-AGP into the MSC problem and find either an exact solution, or an approximation solution to the MSC problem. The solution, whether exact or approximate, would provide a valid guard placement for the 3D-AGP.

The guard placement algorithm relies on the following procedures. The first one is to partition a given OPP into a set of rectangular prisms. This procedure will be discussed in Section 5.3. The second procedure is to compute the visibility subsets for the corner points from these rectangular prisms. This will be discussed in Section 5.4. The last one is to reduce the number of guards by using the MSC problem as a means to calculate the minimum corner points required to cover the entire all pieces, hence the entire original OPP is covered. The conversion to the MSC problem will be described in Section 5.6. Section 5.7 provides a brief

introduction to the solution of the MSC problem. The input to the algorithm, which is also the input to the partitioning procedure, is in the form of extreme vertices, as described in Chapter 2.

5.3 Partitioning of an Orthogonal Pseudo-Polyhedron

The purpose of partitioning an OPP is to decompose the OPP into a set of rectangular prisms $\Pi = \{\rho_i \mid \rho_i \text{ is rectangular prism where } i=1, \dots, m\}$, and m is the number of rectangular prisms. Figure 5.1 is an illustration of partitioning result of an OPP.

As mentioned in Chapter 3 that Ayala and Rodriquez have proposed a technique of partitioning an OPP into a set of rectangular prisms [76]. In their partitioning technique, the number of rectangular prisms of partitioning an OPP in different directions is not unique, in which the number of rectangular prisms depends on the way to cut the OPP. Therefore, if their technique is applied to get the number of rectangular to solve the 3D-AGP, then the number of guards is not unique as well. Hence, it is better to develop a new technique of partitioning such that this technique always gives a unique number of rectangular prisms for partitioning an OPP.

In this new technique, decomposing an OPP into a set of rectangular prisms rests on the following observations: i) each OPP has at least two planes of vertices that parallel to one of the three planes (i.e., XY-plane, XZ-plane, and YZ-plane). An OPP is a rectangular prism if the number of a plane of vertices that parallel to each plane of the three planes is exactly two; otherwise, the OPP is not a rectangular prism. ii) An OPP can be decomposed into a set of smaller objects by using a number of *splitting planes*, which split an OPP into two halves in which each half may contain one or more OPPs. A splitting plane is perpendicular to one of the three orthogonal directions and is represented by a plane equation. A *splitting plane equation* is a plane equation that passes through a plane of vertices. iii) Vertices in a plane of

vertices have the same coordinate X , Y or Z .

The procedure of decomposition is called `DecomposeOPP`, and this procedure assumes: i) An OPP is represented by its extreme vertices that are sorted depending on the splitting plane's perpendicularity. If an OPP is split by a splitting plane that is parallel to the x -axis, then the OPP is YZX -sorted, and if an OPP is split by a splitting plane that is parallel to the y -axis then the OPP is ZXY -sorted, and if an OPP is split by a splitting plane that is parallel to the z -axis then the OPP is XYZ -sorted. To sort several OPPs, sort vertices based on each group OPP before sorting in XYZ -sorted or another sorted (ii) A splitting plane passed through vertices of planes of vertices that contain at least one non-V30 vertex (reflection vertex); therefore, the number of splitting plane along each axis depends on the number of planes of vertices.

The procedure `DecomposeOPP` has two kinds of inputs and one output. The inputs are OPP and splitting plane equation, and the output is a set of rectangular prisms. The splitting planes equations are derived from by the coordinate of each plane of vertices. For example, if the vertices of a plane of vertices have the same coordinate X , say coordinate $X = x_l$, then the splitting plane equation is $x = x_l$, where x_l is also called as the splitting plane value. To make a convenient splitting process, the splitting is processed in XYZ -processed that has a meaning: the OPP is split first along the x -axis, then along y -axis, and then along z -axis. The detail procedure is given Figure 5.2.

```

Procedure RectangPrism (INPUT P: OPP, SP: splitting plane equations; OUTPUT RP: a set of
rectangular prisms)

var    $s_i$        : splitting plane value
       $dir$        : the direction of splitting plane equation movement
              (1 =  $x$ -direction, 2 =  $y$ -direction, 3 =  $z$ -direction)
      Q,R,Q'    : OPP
       $k_{dir}$     : number of splitting plane equations at  $dir$  direction

for ( $dir = 1$  to 3){
  Q' =  $\emptyset$ ;
  if ( $dir = 1$ ) { Sort P according its group of OPP and XYZ-order }
  if ( $dir = 2$ ) { Sort P according its group of OPP and YZX-order }
  if ( $dir = 3$ ) { Sort P according its group of OPP and ZXY-order }

  for ( $i = 1$  to  $k_{dir}$ ){
    ReadSplittingPlane( $i, s_i$ ) // read  $s_i$ , the  $i^{th}$  splitting plane
                                equation in direction  $dir$ 

    SplittingOPP( $P, s_i, Q, R$ );
     $P = R$ ;
     $Q' = Q' + Q$ ;
  }
   $P = Q'$ ;
}
RP = P;

```

Figure 5.2: Algorithm `rectangPrism` for Decomposing P into Rectangular Prisms

To split an OPP into two halves, cut the OPP using a plane of vertices until it hits the boundary of the OPP, in which a splitting plane is perpendicular to the x -axis, then the procedure `SplittingOPP` is applied (see Figure 5.3).

The procedure `SplittingOPP` has two kinds of inputs, namely a splitting plane equation and the brinks of P that perpendicular to the splitting plane. This algorithm is considered to those brinks that are parallel to the x -axis, and they appear as consecutive couples of vertices in a YZX -sorted model. So, if the splitting plane equation is $x = s_x$ then the splitting plane will cut all brinks that parallel to the x -axis.

To split an OPP at a splitting plane, the splitting plane value is compared with the coordinate two end points of a brink, and then the vertices of P will be grouped into two groups, Q and R respectively. The splitting continues until the last splitting equation on X -direction is applied.

If a splitting plane is perpendicular to other axis then a suitable *ABC*-sorted must be applied to the model prior to this process. The procedure `ReadBrink(P: OPP in YZX-sorted model; vb,ve: a pair of vertices)` reads as a next brink of the *YZX*-sorted model. The procedure `Intersect(end points of brinks,splitting plane equation;intersection vertex)` obtains v_s as follows: Let the splitting plane equation $s = s_x$, and $v_b = (v_{bx},y,z)$ and $v_e = (v_{ex},y,z)$ be the beginning and ending vertices coordinate of a brink, then the coordinate of $v_s = (s_x,y,z)$.

```

Procedure SplittingOPP(INPUT P: OPP in YZX-sorted, s=sx : splitting plane; OUTPUT Q,R:
Orthogonal pseudo-polyhedron)
// partition P by a splitting plane

VAR:  vb, ve           : end points of a brink
      vbx, vex       : x-coordinate of the end points
      vsx            : point in the middle of a brink
      k              : number of brinks

Q = ∅; R = ∅
For (i=1 to k){
  ReadBrink(P,vb,ve);
  IF vbx < sx  && vex <= sx THEN Q = Q + (vb,ve) ENDIF
  IF vbx >= sx && vex > sx THEN R = R + (vb,ve) ENDIF
  IF vbx < sx  && vex > sx THEN
    Intersect(vb,ve,sx,vs)
    Q = Q + (vb,vs)
    R = R + (vs,ve)
  ENDIF
  ReadBrink(P,vb,ve);
}

```

Figure 5.3: Algorithm for Splitting an OPP into Two Halves

Similar splitting processes are also applied to *Y*-axis and *Z*-axis, but keep the parts resulted from the previous process together until all cut (*x*-cut, *y*-cut, or *z*-cut) are completed.

As an illustration, Figure 5.4 is the OPP in Figure 5.1 after partitioning. Each corner point in the OPP is labelled.

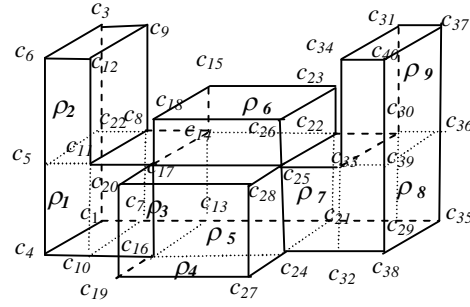


Figure 5.4: Corner Points on an Orthogonal Pseudo-Polyhedron

5.4 Computing Visibility Subsets on an Orthogonal Pseudo-Polyhedron

The purpose of computing the visibility subsets is to construct a collection non empty sets $S = \{ S_j / j=1, \dots, k \}$, where $S_j = \{ \rho \mid \rho \in \Pi \text{ and } \rho \subseteq \text{Vr}(c_j) \}$ is the visibility subset for corner point c_j .

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 OPP, a *Type II face* is completely made up of the interior points of the original OPP except possibly at the edges of the face. If an edge of a face is also an edge on the original OPP, the edge is said to be *Type I edge*. Otherwise the edge would consist of only interior points of the original OPP 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.

The following algorithm assumes the availability of k corner points c_i ($i=1, 2, \dots, k$) from m rectangular prisms rp_j ($j=1, 2, \dots, m$) which are resulted from the partition of an OPP. 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 ($i=1, 2, \dots, k$). 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 . See Figure 5.5 for detail.

```

for (  $i = 1$  to  $k$  ) {
   $S_i = \emptyset$ ;
  for (  $j = 1$  to  $m$  ) {
    If ( $c_i$  is a corner point of  $rp_j$  ){
       $S_i = S_i + \{ rp_j \}$ ;
      continue;
    }
     $blocked = false$ ;
    for (  $l = 1$  to  $m$  ) {
      if (  $l \neq j$  )
         $blocked = IsViewBlocked (c_i, rp_j, rp_l)$ ;
      if (  $blocked$  )
        break;
    }
    if ( not  $blocked$  )
       $S_i = S_i + \{ rp_j \}$ ;
  }
}

```

Figure 5.5: Algorithm for Constructing Visibility Subsets

The function *IsViewBlocked* 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.

```

function IsViewBlocked ( $c_i$ ,  $rp_j$ ,  $rp_1$ )
{
    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  $rp_1$  ) {
            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;
}

```

Figure 5.6: Function *IsViewBlocked* for Blocking Determination

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 the visible face at the base and the corner point at its apex. If another rectangular prism rp_1 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 OPP, hence it would be “transparent”.

Rectangular prism rp_1 blocks the view from c_i to rp_j if and only if rp_1 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_1 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_1 . The line connecting c_i and s would intersect with one or more points of rp_1 . 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_1 blocks the view from c_i to rp_j , then rp_1 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_1 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_1 . This means that rp_1 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.

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 noticed 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 intersects with the edge.

As an example of the visibility procedure result is presented in Table 5.1 that shows the visibility region of each corner point of partitioned OPP in Figure 5.4. Each visibility subset of a corner c_i is kept in subset S_i .

Table 5.1: Corner Point and their Visibility Regions

Subset	Element of Subset	Subset	Element of Subset	Subset	Element of Subset
S_1	$\rho_1, \rho_2, \rho_3, \rho_5, \rho_7, \rho_8$	S_{15}	ρ_5, ρ_6	S_{29}	$\rho_1, \rho_3, \rho_5, \rho_7, \rho_8, \rho_9$
S_2	$\rho_1, \rho_2, \rho_3, \rho_5, \rho_7, \rho_8$	S_{16}	$\rho_1, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7, \rho_8$	S_{30}	$\rho_1, \rho_3, \rho_5, \rho_7, \rho_8, \rho_9$
S_3	ρ_1, ρ_2	S_{17}	$\rho_1, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7, \rho_8,$	S_{31}	ρ_8, ρ_9
S_4	$\rho_1, \rho_2, \rho_3, \rho_5, \rho_7, \rho_8$	S_{18}	ρ_5, ρ_6	S_{32}	$\rho_1, \rho_3, \rho_5, \rho_7, \rho_8, \rho_9$
S_5	$\rho_1, \rho_2, \rho_3, \rho_5, \rho_7, \rho_8$	S_{19}	ρ_4, ρ_5	S_{33}	$\rho_1, \rho_3, \rho_5, \rho_7, \rho_8, \rho_9$
S_6	ρ_1, ρ_2	S_{20}	ρ_4, ρ_5	S_{34}	ρ_8, ρ_9
S_7	$\rho_1, \rho_2, \rho_3, \rho_5, \rho_7, \rho_8$	S_{21}	$\rho_1, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7, \rho_8$	S_{35}	$\rho_1, \rho_3, \rho_5, \rho_7, \rho_8, \rho_9$
S_8	$\rho_1, \rho_2, \rho_3, \rho_5, \rho_7, \rho_8$	S_{22}	$\rho_1, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7, \rho_8$	S_{36}	$\rho_1, \rho_3, \rho_5, \rho_7, \rho_8, \rho_9$
S_9	ρ_1, ρ_2	S_{23}	ρ_5, ρ_6	S_{37}	ρ_8, ρ_9
S_{10}	$\rho_1, \rho_2, \rho_3, \rho_5, \rho_7, \rho_8$	S_{24}	$\rho_1, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7, \rho_8$	S_{38}	$\rho_1, \rho_3, \rho_5, \rho_7, \rho_8, \rho_9$
S_{11}	$\rho_1, \rho_2, \rho_3, \rho_5, \rho_7, \rho_8$	S_{25}	$\rho_1, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7, \rho_8,$	S_{39}	$\rho_1, \rho_3, \rho_5, \rho_7, \rho_8, \rho_9$
S_{12}	ρ_1, ρ_2	S_{26}	ρ_5, ρ_6	S_{40}	ρ_8, ρ_9
S_{13}	$\rho_1, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7, \rho_8$	S_{27}	ρ_4, ρ_5		
S_{14}	$\rho_1, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7, \rho_8$	S_{28}	ρ_4, ρ_5		

5.5 Mapping 3D-AGP into MSC Problem

Given an OPP, assume that it is partitioned into a set of m rectangular prism $\Pi = \{\rho_i \mid \rho_i \text{ is rectangular prism and } i=1, \dots, m\}$. The partition results in corner points from these m rectangular prisms. Section 5.5 described a procedure to construct k visibility subset S_j ($j=1, 2, \dots, k$), each of which contains those rectangular prisms that are completely visible from a given corner point. A trivial guard placement can be obtained by placing one guard at each corner point. This is because each guard would cover a number of rectangular prisms and the k guards would cover all rectangular prisms hence the entire OPP, *i.e.*, $\cup_{j=1 \text{ to } k} S_j = \Pi = P$.

However the number of guards in the above trivial placement scheme is far too excessive. Many rectangular prisms are visible from multiple corner points. Furthermore some visibility subsets may contain all rectangular prisms from another visibility subset. Hence the number

of guards can be greatly reduced if a minimum number of visibility subsets can be found that contain all rectangular prisms.

As each corner point c_j has an associated S_j , the above task is equivalent to the Minimum Set Cover (MSC) problem. Hence 3D-AGP problem can be mapped to the Minimum Set Cover problem by imposing $U = \Pi$ and $S = \{S_1, S_2, \dots, S_k\}$. The solution to the MSC problem, i.e., the subset $S' \subseteq S$ with the minimum cardinality that covers Π gives us a valid guards placement, and the cardinality of S' can be interpreted as the number of guards in an OPP.

Section 5.6 will show how the Integer Linear Programming gives a very close solution for the MSC problem.

5.6 Solving the MSC Problem — An Example

As mentioned above, the 3D-AGP can be converted to the MSC problem: given a universe $U = \{\rho_i \mid \rho_i \text{ is rectangular prism and } i=1, \dots, m\}$, and a non-empty set $S = \{S_j \mid j=1, \dots, k\}$ where $S_j = \{\rho \mid \rho \in U \text{ and } \rho \text{ is totally visible from a corner point } c_j\}$, find a subset S' of S with the minimum cardinality that covers all elements in U .

There are a number of methods for solving the MSC problem, such as linear programming, heuristic algorithms, and exact algorithms [93].

Linear programming is regarded a very important technique for the optimization of a linear objective function, subject to linear equality and inequality constraints. This method is used in many areas, one of it being business and economics due to the fact that problems like maximizing outcome can be straightforwardly stated and efficiently solved.

Linear programming is the problem of optimizing (minimizing or maximizing) a linear function subject to linear inequality constraint [94]. In their canonical form, linear programs are expressed as:

$$\text{Maximize / Minimize } c^T x$$

$$\text{Subject to } Ax \leq b,$$

where x is a vector of variables whose value must be determined, and c and b are vector of known coefficients. The expression $c^T x$ is to be maximized / minimized within the limit defined by $Ax \leq b$. Linear programming problems can be solved using different very well known methods such as Simplex, Ellipsoid, and Interior Point.

If each value of a vector whose value must be determined is an integer number then the above model is called integer linear programming. This is the problem of optimizing (minimizing or maximizing) a linear function subject to linear inequality constraint in which the possible value of each variable is restricted to be an integer number [94].

Another approach to solve the MSC problem is by using heuristic approach. Heuristic approach is used for algorithms which find solutions among all possible ones, but they do not guarantee that the best will be found, therefore they may be considered as approximately and not accurate algorithms [93]. A greedy algorithm for the MSC problem is an example of heuristic approach [94]. This algorithm always selects a set which cover the maximum number of yet uncovered elements, and it is a log M -approximation algorithm, where M is the number of sets. The greedy approach does not guarantee that, upon termination, a minimum cover will be found. However, this algorithm can be used in the minimization process to establish a first upper bound for the size of the minimum cover.

The last approach in this discussion is the exact approach. The most effective exact approach to MSC are branch and bound algorithm [93]. The main reason for this success is the fact that it is apparently very difficult to get significantly stronger lower bound by alternative methods which are computationally more expensive.

5.6.1 Integer Linear Programming formulation of the MSC problem

As an example of solving the MSC problem, this section will discuss the application of integer linear programming approach. The problem of MSC is formulated in integer linear programming model as follows:

Given a Boolean matrix A having size $m \times k$. Let $M = \{1, \dots, m\}$ and $K = \{1, \dots, k\}$, then column $j \in K$ is said to cover a row $i \in M$ if $A_{ij} = 1$. The MSC problem calls for a minimum subset $S \subset K$ of columns such that each row $i \in M$ is covered by at least one column $j \in S$.

The integer linear programming is defined as [95] :

$$\begin{aligned} & \text{Minimize} && \sum_{j=1}^n c_j x_j \\ & \text{Subject to} && \sum_{j=1}^n a_{ij} x_j \geq b_i \quad i = 1, \dots, m \\ & && x_j \in \{0,1\} \quad j = 1, \dots, k \end{aligned}$$

The integer linear programming can be solved by the dual simplex algorithm [95].

All subsets in Table 5.1 can be processed using the dual simplex algorithm by inputting the component of vectors a , b and c as shown in Appendix 1. The component vector a consists of two numbers only, namely 0 and 1. 1 is used if a piece ρ is an element of subset S_i , and 0 for the other case. Meanwhile, the number of component in vector b is equal with the number of pieces in Π , and all components are 1.

After processing these inputs by using the dual simplex algorithm as shown in the Appendix 1 then a result is achieved as follows: the variables x_1 , x_{13} and x_{29} have value 1, and the other variables have value 0. The objective value at the optimal point: $z = 3$. It implies that the number of guard is 3, and their positions are at c_1 , c_{13} and c_{29} as shown in the following Figure.

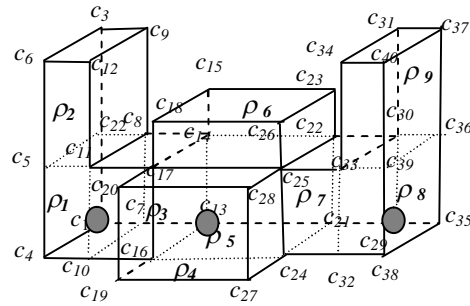


Figure 5.7: The Guards Position after Deploying Integer Linear Programming

5.7 Time Complexity of the Guard Placement Algorithm

The time cost of the guard placement algorithm is calculated as follows. Firstly, the time complexity for decomposing an orthogonal polyhedron into a set of rectangular prisms is determined by the number of splitting planes, and the whole partition can be complete in polynomial time in n , the number of vertices in the orthogonal polyhedron. Secondly, 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 . However, the final step requires the solution of an MSC problem, which is NP-hard. Therefore, no matter how fast one can perform partitioning and construct visibility subsets, the overall time complexity for the guard placement algorithm is still exponential in n .

Although finding the exact solution to the MSC problem will take exponential time, there are heuristic and approximation algorithms for the MSC problem [92]. These algorithms can produce solutions that are close to the true minimum in reasonable amount of time. As

discussed in Section 5.6 that even if a solution to the MSC is approximate, it still provides a valid guards' placement.

5.8 Reducing the Number of Guards

It is still possible to reduce the number of guards for covering an OPP that are determined by guard placement algorithm. This possibility comes from a fact that some pieces could be covered by two or more guards cooperatively despite of covered totally by a guard. Therefore, the guard that covers totally a piece can be removed. For example: the piece ρ_4 is totally visible from c_{13} , but ρ_4 is also partially visible from corner points c_1 and c_{29} . If both of corner points cover all the area of ρ_4 , then c_{13} can be deleted as a guard to achieve a smaller number of guards.

In this section, a new guard placement procedure is proposed to reduce the number of guard. Basically, the new guard placement algorithm is quite similar with the previous guard placement algorithm. The only different is the new algorithm used the result of the previous as inputs and a rectangular prism may be monitored cooperatively by several guards.

The input of new guard placement algorithm is the result of the previous guard placement algorithm that is a collection of corner points that can cover all pieces in an OPP. Meanwhile, the output is the smallest number of corner points for covering an OPP in which number of corner points implies the number of guards.

The procedure of the new algorithm consists of several activities namely: (i) identify pieces that are partially visible from at least two resulting guards, (ii) partitioning each piece in rectangular prism which is called as a bounding box into polyhedra, (iii) determine the minimum set of corner points for covering all pieces.

The following definitions are introduced to support the proposed the new guard placement algorithm:

Definition 5.4. Rectangular prism partitioning is splitting a rectangular prism or polyhedron in a bounding box into visible area and non-visible area from a view point. Both resulting areas are polyhedra.

If a view point does not see the whole surface of an orthogonal prism ρ , then ρ is said partially visible from c . In another word, ρ is *visible by section* from c . It might happen several view-points see cooperatively the whole surface of ρ . This fact leads us to the following definition.

Definition 5.7. A piece ρ is visible by sections if it is covered by several visibility sections cooperatively.

Let C' be a set of corner points in an OPP in which all guards are lain as a result of the previous guard placement algorithm, then new guard placement algorithm has main steps as follows:

- Step 1. Determine D_ρ that is partially visible pieces from at least two corner points of element C' .
- Step 2. Decompose each element of D_ρ into polyhedra by using cutting planes that are derived from all element of C' . Put each resulted polyhedron and undecomposed orthogonal prisms into a new set of pieces $\Gamma = \{\rho_i \mid \rho_i \text{ is rectangular prism or polyhedra, and } i = 1, \dots, m\}$
- Step 3. Construct $S' = \{S_j' \mid j=1, \dots, k\}$, where $S_j' = \{\rho \mid \rho \in \Gamma \text{ and } \rho \subseteq \text{Vr}(c_j)\}$. m and k are the number of pieces and corner points of a partitioned OPP respectively.
- Step 4. Determine the minimum subset $S_j'' \subset S_j'$ that cover Γ

The first step of algorithm is determining a subset D_ρ that partially visible from at least two corner points in C' . It is necessary to say that each element D_ρ must be partially visible from at least two corner points, because at least two corner points may see a piece cooperatively, and it cannot do by a single corner point. A function `CountedVisibility` is created to retrieve D_ρ during counting the visibility subsets in a partitioned OPP.

The second step of algorithm is decomposing each piece of D_ρ into a number of polyhedra. Each element of D_ρ is regarded as a bounding box which has a form as rectangular prism. The purpose of this step is to decompose the original piece into several polyhedra. To achieve a set of polyhedra in a bounding box, a number of splitting operations may be carried out, and it depends on the number of cutting plane. The number of cutting planes is at most equal with the number of dominant corner points. The algorithm for `Splitting Polyhedra in a Bounding Box from a view Point` in Chapter 4 is recalled to do the task of partitioning. All polyhedra and the rectangular prisms become elements of set of partition Γ .

The third step is determining pieces that are totally visible from each corner point $c \in C'$, and store it in S_j' .

Finally, the last step is determining the minimum subset $S_j'' \subset S_j'$ that covers Γ . This step is similar with the last step of the previous guard placement algorithm by transferring the problem into MSC problem. The inputs are $U = \Gamma$ and $S = S_j'$, and the result is the minimum set S_j'' that covers all pieces in Γ . The cardinal number of element of S_j'' implies the number of guards to cover the interior of P . By using the dual simplex method, the MSC problem as mapping of 3D-AGP is solved.

As implementation, the new guard placement algorithm will be applied to reduce the number of guards for example in Section 5.7. Initially, based on the result of the previous guard

placement algorithm, $\Gamma = \{\rho_1, \rho_{21}, \rho_{22}, \rho_{23}, \rho_3, \rho_{41}, \rho_{42}, \rho_{43}, \rho_5, \rho_{61}, \rho_{62}, \rho_{63}, \rho_7, \rho_8, \rho_{91}, \rho_{92}, \rho_{93}\}$ and $S' = \{S_1, S_{13}, S_{29}\}$ as shown in Figure 5.8.

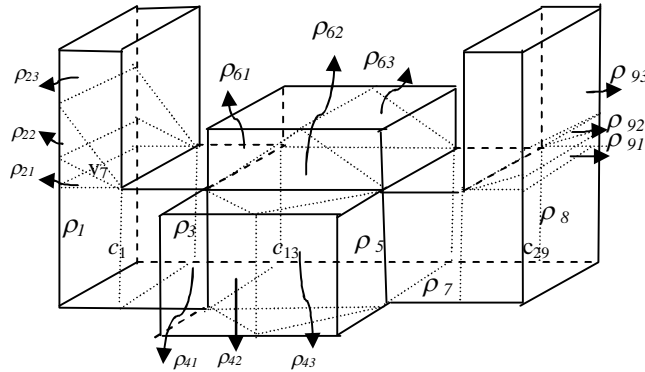


Figure 5.8: Partitioning in Pieces that are Visible by Sections

In this figure, a corner point c_1 see a set of pieces $S_1' = \{\rho_1, \rho_{21}, \rho_{22}, \rho_{23}, \rho_3, \rho_{42}, \rho_{43}, \rho_5, \rho_{62}, \rho_{63}, \rho_7, \rho_8, \rho_{91}\}$. Each element of S' has the visibility region listed in Table 5.2, which shows the visibility region of each corner point that is kept in subset S_i . To simplify the notation, each element is relabeled, and they start from $i = 1$ to m , where m is the total number of pieces in Γ .

Table 5.2: Corner Points and their Visibility Region

Subset	Relabelled of Subset	Element of Subset	Relabelled of element
S_1'	S_1'	$\rho_1, \rho_{21}, \rho_{22}, \rho_{23}, \rho_3, \rho_{42}, \rho_{43}, \rho_5, \rho_{62}, \rho_{63}, \rho_7, \rho_8, \rho_{91}$	$\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_7, \rho_8, \rho_9, \rho_{11}, \rho_{12}, \rho_{13}, \rho_{14}, \rho_{15}$
S_{13}'	S_2'	$\rho_1, \rho_{21}, \rho_{22}, \rho_3, \rho_{41}, \rho_{42}, \rho_{43}, \rho_5, \rho_{61}, \rho_{62}, \rho_{63}, \rho_7, \rho_8, \rho_{91}, \rho_{92}$	$\rho_1, \rho_2, \rho_3, \rho_5, \rho_6, \rho_7, \rho_8, \rho_9, \rho_{10}, \rho_{11}, \rho_{12}, \rho_{13}, \rho_{14}, \rho_{15}, \rho_{16}$
S_{29}'	S_3'	$\rho_1, \rho_{21}, \rho_3, \rho_{41}, \rho_{42}, \rho_5, \rho_{61}, \rho_{62}, \rho_7, \rho_8, \rho_{91}, \rho_{92}, \rho_{93}$	$\rho_1, \rho_2, \rho_5, \rho_6, \rho_7, \rho_9, \rho_{10}, \rho_{11}, \rho_{13}, \rho_{14}, \rho_{15}, \rho_{16}, \rho_{17}$

After processing these inputs by using the dual simplex algorithm as shown in the Appendix 2 then a result is achieved as follows: the variables x_1 and x_3 have value 1, and variable x_2

have value 0. The objective value at the optimal point: $z = 2$. It implies that the number of guard is 2, and their positions are at c_1 and c_{29} .

5.9 Summary

This chapter has discussed the procedure of the point guard placement for monitoring the interior of an orthogonal pseudo-polyhedron. The main steps are: partitioning a given OPP into a set of rectangular prisms, counting visibility subsets of each corner point, and transfer the 3D-AGP into the MSC problems.

The procedure for partitioning OPP and counting visibility subsets can be constructed in polynomial time in n . However, the final step requires the solution of an MSC problem, which is NP-hard. Therefore, in overall, the point guard placement for solving the 3D-AGP is NP-hard.

The contributions of this chapter are:

- The procedure of point guard placement in an OPP is proposed.
- A new technique of partitioning an OPP into set of rectangular prism has been presented.
- The procedure for counting the visibility subset of each corner point in a partitioned OPP is also proposed.
- A method has been proposed to reduce the number of point guards for monitoring the interior of an OPP.

The upper bound number of guards for monitoring the interior of an orthogonal polyhedron is dealt with in the next chapter.

CHAPTER 6

AN UPPER BOUND ON THE NUMBER OF FIXED-POINT GUARDS FOR ORTHOGONAL POLYHEDRA

6.1 Introduction

In this chapter, the dominant pieces around various types of vertex configurations in any orthogonal polyhedra are identified. A technique is also proposed to reduce the number of data inputs for the minimum set cover (MSC) problem. The main contribution of this chapter is in the establishment of an upper bound of fixed-point guards for any orthogonal polyhedra.

As discussed in Chapter 5, rectangular prisms produced by partitioning on an orthogonal pseudo-polyhedron were used as input data for two procedures: calculating the visibility subsets and the solution to the MSC problem. The rectangular prisms, which are also called pieces in this chapter can be grouped as dominating pieces and dominated pieces. The grouping of the pieces is based on the observation that is described below:

Let an orthogonal polyhedron be partitioned into three pieces, as depicted in Figure 6.1. The piece ρ_1 is totally visible from the corner points in the pieces ρ_1 and ρ_2 , the piece ρ_2 is totally visible from the corner points in the pieces ρ_1 , ρ_2 and ρ_3 . The piece ρ_3 is totally visible from the corner points in the pieces ρ_3 and ρ_2 . All corner points that can see ρ_1 can also see ρ_2 . However, not all corner points that can see ρ_2 can see ρ_1 . For example, the corner points in ρ_3 can see ρ_2 , but not all corner points in ρ_3 can see ρ_1 . In this situation, ρ_1 is said to be dominant over ρ_2 . Using the same observation for pieces ρ_2 and ρ_3 , ρ_3 can be seen to be dominant over ρ_2 .

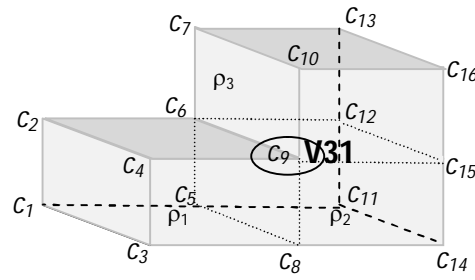


Figure 6.1: Dominant Pieces Shared by the V31 Vertex

Based on these observations, the set of all dominant pieces of an orthogonal polyhedron can represent all the pieces in that orthogonal polyhedron. The number of dominant pieces in any orthogonal polyhedron is used to determine the upper bound number of fixed-point guards for monitoring the interior of an orthogonal polyhedron.

6.2 Determining the Dominant Pieces

Recall that the piece ρ of an orthogonal polyhedron is said to be totally visible from a point, c , if every point of ρ is visible from c , and ρ is said to be partially visible from c if some, but not all, points of ρ are visible from c . Note that if a piece ρ is totally visible from a point c , then the point c is visible from every point of the piece ρ . Hence in this case, the point c is also said to be totally visible from the piece ρ . The concept of a *dominant piece* is defined below.

Definition 6.1: Let $G\rho_i$ and $G\rho_j$ be two set of corner points, which are derived from a rectangular-prism decomposition of an orthogonal polyhedron, that are totally visible from the piece ρ_i and ρ_j , respectively. Then, the piece ρ_i is said to be dominant over a piece, ρ_j , if $G\rho_i \subset G\rho_j$. If $G\rho_i = G\rho_j$ then ρ_i is said to be equivalent to ρ_j and vice versa.

As an example, the dominant pieces in Figure 6.1 are obtained by Definition 6.1 as follows.

$$G\rho_1 = \{c_1, c_2, c_3, c_4, c_5, c_6, c_8, c_9, c_{11}, c_{12}, c_{14}, c_{15}\}, G\rho_2 = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10},$$

$c_{11}, c_{12}, c_{13}, c_{14}, c_{15}, c_{16}$ }, and $G\rho_3 = c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}, c_{16}$. $G\rho_1 \subset G\rho_2$ and $G\rho_3 \subset G\rho_1$; therefore, ρ_1 and ρ_3 are the dominant pieces over ρ_2 .

A vertex on an orthogonal polyhedron is either be a reflex vertex or a convex vertex. An *arbitrary vertex* is a corner point of any pieces in a partitioned orthogonal polyhedron that is neither a reflex vertex nor a convex vertex. The pieces that are totally visible from any vertices can be grouped as two groups of pieces: around pieces and remote pieces. An *around piece*, ρ , is a piece that is visible from at least one vertex, v , and v is a corner of ρ . A *remote piece*, ρ , is a piece that is visible from at least one vertex but all corner points of ρ are arbitrary vertices. The partitioned orthogonal polyhedron in Figure 6.2 has eight vertices: $c_1, c_2, c_3, c_4, c_{13}, c_{14}, c_{15}$ and c_{16} . The pieces ρ_1 and ρ_3 are around pieces, and ρ_2 is a remote piece. Subsection 6.2.1 will identify dominant pieces at around pieces and remote pieces.

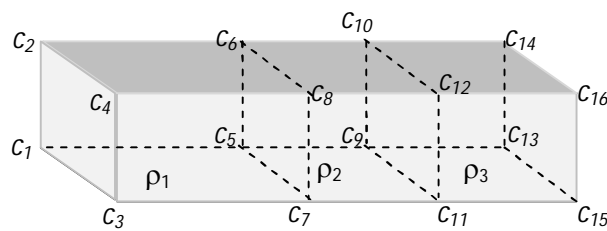


Figure 6.2: Around Pieces and Remote Pieces

6.2.1 Around pieces of various types of vertex

Vertices are differentiated by their vertex configurations. Chapter 3 defined six kinds of vertex configurations on orthogonal polyhedra: V30, V31, V32, V33, V42, and V63. The first digit of a vertex label represents the number of edges meeting at the vertex, and the second digit represents the number of concave dihedral angles at the vertex.

Figure 6.3 depicts all six vertex configurations. Each orthogonal polyhedron is decomposed into a set of rectangular prisms (pieces) by cutting them at the reflex vertices. Some vertex is

shared by between three, four, five, and seven different pieces. For example, the V32 vertex is shared by five pieces, so there are five around pieces of the V32 vertex. The following list shows the number of around pieces for each type of vertex in orthogonal polyhedra.

- A V30 vertex is shared by one piece.
- A V31 vertex is shared by three pieces.
- A V32 vertex is shared by five pieces.
- A V33 vertex is shared by seven pieces.
- A V42 vertex is shared by four pieces.
- A V63 vertex is shared by four pieces.

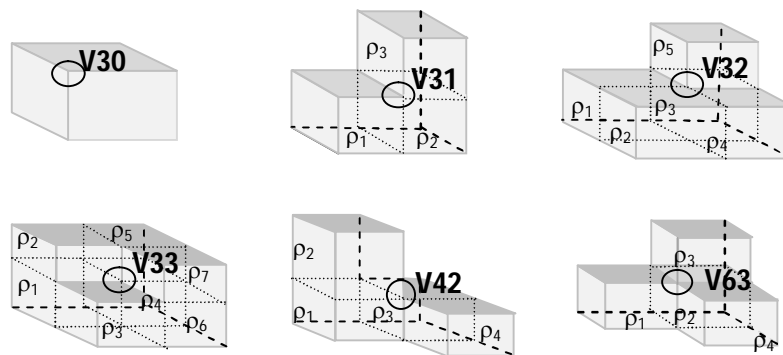


Figure 6.3: Around Pieces of Vertices of Orthogonal Polyhedra

There are two kinds of around pieces of a vertex: dominant pieces and dominated (non-dominant) pieces. A dominant piece is less totally visible than a dominated piece from a number of corner points. There are a number of around pieces for a V31 vertex, V32 vertex, V33 vertex, V42 vertex, and V63 vertex. A V30 vertex is not considered to have a dominant piece because there is only one around piece of a V30 vertex. By using Definition 6.1, the dominant pieces of each orthogonal polyhedron in Figure 6.3 can be identified. The results are described as follows:

1. The number of dominant pieces sharing each V31 vertex is two.

There are three around pieces sharing the same V31 vertex, namely ρ_1 , ρ_2 , and ρ_3 . ρ_1 is dominant over ρ_2 , and ρ_3 is also dominant over ρ_2 . But ρ_1 and ρ_3 are not equivalent; hence, the number of dominant pieces sharing each V31 vertex is two.

2. The number of dominant pieces sharing each V32 vertex is two.

There are five around pieces sharing the V32 vertex, namely ρ_1 , ρ_2 , ρ_3 , ρ_4 and ρ_5 . The pieces ρ_1 , ρ_2 and ρ_4 are equivalent, and they are dominant over ρ_3 . ρ_5 is also dominant over ρ_3 . Therefore, the number of dominant pieces sharing the V32 vertex is two.

3. The number of dominant pieces sharing each V33 vertex is three.

There are seven around pieces sharing the same V33 vertex, namely ρ_1, \dots, ρ_7 . The number of dominant around pieces sharing the V33 vertex is three, and they are ρ_2 , ρ_3 and ρ_7

4. The number of dominant pieces sharing each V42 vertex is two.

There are four around pieces sharing the same V42 vertex, namely ρ_1, \dots, ρ_4 . The number of dominant pieces sharing the same V42 vertex is two, and they are ρ_2 and ρ_4

5. The number of dominant pieces sharing each V63 vertex is three. They are ρ_1 , ρ_3 and ρ_4 .

6.2.2 Remote pieces

As defined in Section 6.2, a remote piece is visible from at least one vertex of an orthogonal polyhedron and all corner points of the remote piece are arbitrary point. A remote piece lies at least between two around pieces.

Lemma 6.1: A remote piece is dominated by an around piece.

Proof: As a remote piece lies at least between two around pieces, then a remote piece has at least two arbitrary boundaries that is a kind of boundary having arbitrary point as corner points. It is obvious that the number of vertices see a remote piece is more than the number of vertices see an around piece. By definition 6.1, an around piece is less visible than a remoter piece; therefore, an around piece is dominant over a remote piece. \square

6.3 Reducing the Number of Input Data for the MSC Problem

The number of pieces input in the MSC problem can be reduced by excluding the dominated pieces. Based on the definition of a dominant piece, one can conclude that if guards monitor the dominant pieces, then the guards also monitor the dominated pieces. Therefore, the dominated pieces can be neglected as inputs in the MSC problem. Thus, a corner point that only sees dominated pieces can also be removed as an input in the MSC problem.

Given an orthogonal polyhedron P , assume that P is partitioned into a set of an m rectangular prism $\Pi = \{\rho_i \mid \rho_i \text{ is a rectangular prism and } i=1, \dots, m\}$. The k visibility subset S_j ($j=1, 2, \dots, k$), each of which contains those rectangular prisms that are totally visible from a given corner point, can be transformed into a collection of subsets $G\rho_i$ ($i = 1, 2, \dots, m$), each of which contains those corner points that are visible from any point on a rectangular prism, ρ . To identify the dominant pieces, the rule in Definition 6.1 is applied. All the dominated pieces and the corner points that only see the dominated pieces are removed as input in the MSC problem.

After removing the dominated pieces, there is a set of an m dominant rectangular prism $\Pi = \{\rho_i \mid \rho_i \text{ is a dominant rectangular prism and } i=1, \dots, m\}$, and k visibility subset S_j ($j=1, 2, \dots, k$), where $S_j = \{\rho \mid \rho \in \Pi \text{ and } \rho \subseteq \text{Vr}(c_j)\}$ is the visibility subset for corner point c_j , and c_j are corner points that are only visible from dominant pieces.

6.4 Upper Bound of Point Guards for an Orthogonal Polyhedron

After the partitioning, a guard can be deployed at each piece of an orthogonal polyhedron such that the whole interior of the orthogonal polyhedron is totally covered. However, considering the total number of pieces as an upper bound number of guards on the orthogonal polyhedron seems overkill.

Seidel stated that not every orthogonal polyhedron can be covered by all its vertices [4]. This idea can provide a starting point to calculate the upper bound number of guards. It implies that there could be a point in some orthogonal polyhedra that is not visible from any vertex. A piece ρ is totally visible from a vertex v if all points in ρ are visible from v , and a piece ρ is a partially visible piece if there is a point in the piece that is not visible from any vertex. Thus, there are two groups of pieces in terms of visibility from any vertex: a group of totally visible pieces and a group of partially visible pieces.

The number of guards is not more than the number of dominant pieces. Hence, to obtain an upper bound number of guards, the following steps are applied: The upper bound of dominant pieces is first determined (Subsection 6.4.1), and followed by counting the number of partially visible pieces (Subsection 6.4.2). They are then combined into a single number, and the formula is represented by the n parameter, where n is the number of vertices of the orthogonal polyhedron (Subsection 6.4.3).

6.4.1 Counting the number of dominant visible pieces

Section 6.2 discussed how to obtain the dominant pieces from a partitioned orthogonal polyhedron that is visible from at least one vertex of the orthogonal polyhedron. The number of dominant pieces can be considered as the upper bound of the guard to cover the visible pieces from any vertices inside an orthogonal polyhedron.

When determining the upper bound number of guards for monitoring the visible pieces from any vertex in an orthogonal polyhedron, it should be noted that each reflex vertex is always adjacent to at least one adjacent vertex. Two adjacent vertices share the same dominant piece; therefore, it is sufficient to count one reflex vertex only.

Observation: Each reflex vertex has an adjacent reflex vertex.

Based on the above description, the highest number of dominant around pieces of any vertex is three, and each reflex vertex has an adjacent reflex vertex; therefore, there are at most $3R/2$ number of dominant visible pieces, where R is the number of reflex vertices. If the value of $R = 0$, then the orthogonal polyhedron is a rectangular prism; hence, the number of guards is one. The next step is calculating the number of partially visible pieces.

6.4.2 Counting the number of partially visible pieces

A dominant piece will not be visible if it is placed among six dents, in which each dent has at least four reflex vertices. Figure 6.4 presents an illustration of a partially visible piece ρ which is bounded by six dents d_1, d_2, d_3, d_4, d_5 and d_6 . The position of d_5 is under d_6 .

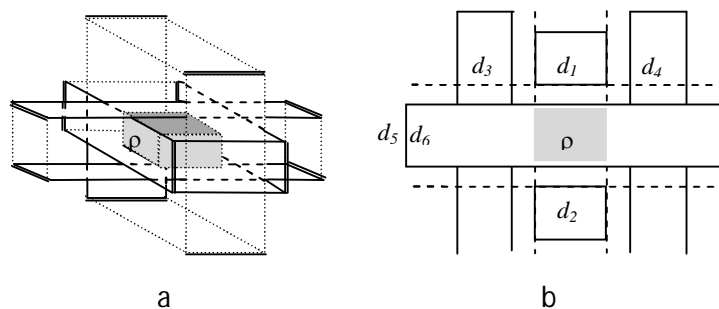


Figure 6.4: Illustration of a Partially Visible Piece ρ Position in 3D (a) and 2D (b)

The problem of finding an upper-bound number of partially visible pieces is defined as follows:

Definition 6.2 : Let D be the number of dents. Then, a function $f(D)$ is the maximum number of partially visible pieces that can be created by D dents.

The function $f(D)$ is calculated based on the following:

1. Each partially visible piece is bounded by six dents

This fact implies that each dent may bound one or more partially visible pieces. If each dent bounds only one partially visible piece then the total number of dents with m partially visible pieces is given by the formula: $D = 6m$. However, if one or more dents are shared by a number of partially visible pieces then the formula is $D < 6m$. Therefore, to obtain the minimum number of dents from m partially visible pieces then a partitioned orthogonal polyhedron must have shared dents as many as possible.

Lemma 6.1 : If the number of dents of m partially visible pieces is less than $6m$, then there is at least one shared dent.

Proof: Let $D^k = \{ D_x^{lk}, D_x^{rk}, D_y^{lk}, D_y^{rk}, D_z^{lk}, D_z^{rk} \}$ be a set of dents around a piece ρ_k where D_x^{lk} and D_x^{rk} are dents in the left and the right of ρ_k , respectively, and each element is not empty. Otherwise ρ_k is visible piece from a vertex. Two partially visible pieces ρ_i and ρ_j share a dent if one element of D^i is the same as that with D^j . If all elements of D^i are different from the element of D^j then $D^i + D^j = 12$, and if, at least one, element of D^i is the same with element of D^j then $D^i + D^j < 12$.

2. Two partially visible pieces may share at most three dents.

There are two positions of a shared dent with two partially visible pieces: between the two partially visible pieces and in the same side of the two partially visible pieces (see Figure 6.5 as an illustration)

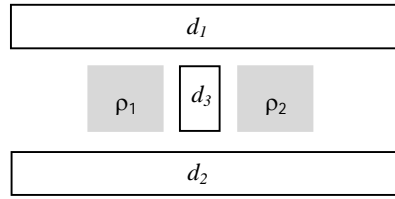


Figure 6.5: The Partially Visible Pieces ρ_1 and ρ_2 Share Three Dents d_1 , d_2 and d_3

This fact implies that if each partially visible piece shares three dents then the total number of dents bound to m partially visible pieces is minimum.

Definition 6.3: A partially visible piece ρ_1 is adjacent to a partially visible piece ρ_2 if ρ_1 shares three dents with ρ_2 .

Theorem 6.1: To get a maximum number of adjacent pieces a from m partially visible pieces then all partially visible pieces are arranged in a cube form.

Proof: A piece may be adjacent to several numbers of pieces, perhaps from one to six. If the partially visible pieces' layout is a line form (Figure 6.6(a)), then the total number of group adjacent pieces is $m-1$, and each partially visible piece belongs to at least one group of adjacent pieces and at most two groups of adjacent pieces. Meanwhile, if the partially visible pieces' layout is a plane form (Figure 6.6(b)), then each partially visible piece belongs to at least one group of adjacent pieces and at most four groups of adjacent pieces. Finally, if the partially visible pieces' layout is a cube form (Figure 6.6(c)), then each partially visible piece belongs to at least one group of adjacent pieces and at most six groups of adjacent pieces. \square

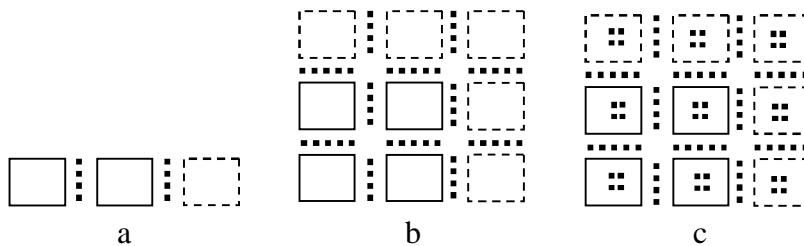


Figure 6.6 : Illustration of Adjacent Pieces among Several Formats

The maximum number of adjacent pieces suggests the minimum number of dents.

Let L , W , and H be the length, the width, and the high, respectively, of a cube form that contains m number of partially visible pieces. m is achieved by multiplication L , W and H ; therefore $L = W = H = m^{1/3}$.

The number of adjacent pieces a can be calculated in several ways. Two different methods are described below:

1. Method 1

Slice the cube of m into $m^{1/3}$ layers in any direction. Let a_1 , a_2 , and $a_m^{1/3}$ be the number of adjacent pieces in each layers. Then

$$\begin{aligned} a_1 &= 2(m^{1/3} - 1) + 2(m^{1/3} - 1)^2 \\ a_2 &= 2(m^{1/3} - 1) + 2(m^{1/3} - 1)^2 \\ &\dots \\ a_m^{1/3} &= 2(m^{1/3} - 1) + 2(m^{1/3} - 1)^2 \end{aligned}$$

The number of adjacent pieces a is equal to the number of pieces in each layer plus the number of adjacent pieces between the layers. Therefore,

$$\begin{aligned} a &= m^{1/3} (2(m^{1/3} - 1) + 2(m^{1/3} - 1)^2) + (m^{1/3} - 1)m^{2/3} \text{ OR} \\ a &= 3m - 3m^{2/3} \end{aligned}$$

2. Method 2

Separating the cube of m into two parts such that one of the partitions is a smaller cube, yields the set-up shown in Figure 6.7.

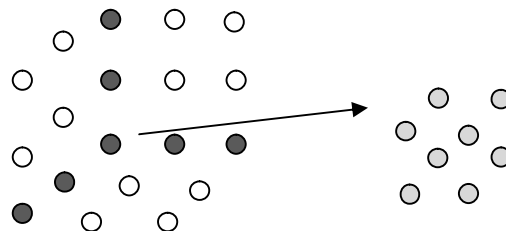


Figure 6.7: Cube of m where $m = 27$

(i) The number of adjacent pieces along the x , y and z axes is represented by a solid black circle: $a_{axis} = 3m^{1/3} - 2$

- (ii) The number of adjacent pieces in the three planes is represented by a white circle: $a_{plane} = 3(m^{1/3} - 1)^2 + 3(m^{1/3}-1)^2$
- (iii) The number of adjacent pieces in the remained cube is represented by a grey circle: $a_{remainder} = (m^{1/3}-1)^3$

So, the total number of adjacent pieces is a compound of the a_{axis} , the a_{plane} and the $a_{remainder}$ as shown in the following formula:

$$a = 3m^{1/3} - 2 + 3(m^{1/3} - 1)^2 + 3(m^{1/3}-1)^2 + 3(m^{1/3}-1)^3$$

$$a = 3m - 3m^{2/3}$$

As mentioned above, the number of dents is equal with each having six partially visible pieces minus the number of deductions due to each adjacent piece sharing a dent. Therefore, the relationship between m and D is derived as follows:

$D = 6m -$ number of deductions due to the shared dent of each adjacent piece.

$$D = 6m - (3 \cdot 3(m^{1/3} - 1) + 3 \cdot 3(m^{1/3} - 1)^2 + 2 \cdot 3(m^{1/3}-1)^2 + 3(m^{1/3}-1)^3 + 2(m^{1/3}-1)^3 + (m^{1/3}-1)^3).$$

$$D = 3m^{1/3} + 3m^{2/3}.$$

Let $x = m^{1/3}$, then $3x^2 + 3x - D = 0$.

The *abc* formula is applied to obtain the value of x . Therefore, $x = -3 + ((9 + 12D)^{1/2})/6$.

$$x \approx (D/3)^{1/2} \text{ or } m^{1/3} = (D/3)^{1/2}.$$

$$m = (D/3)^{3/2} \text{ Or } f(D) = (D/3)^{3/2}.$$

As each dent has eight reflex vertices, $8D \leq R$ or $D \leq R/8$

$$\text{Therefore, } m = (R/24)^{3/2}.$$

6.4.3 Relationship between the number of all vertices and reflex vertices

The relationship between the number of reflex vertices R and the number of vertices n on an orthogonal polyhedron can be simply derived as follows:

Let P be an orthogonal polyhedron, and let P_{V30} , P_{V31} , P_{V32} , P_{V33} , P_{V42} , and P_{V63} denote the number of vertices having V30, V31, V32, V33, V42, and V63 configurations respectively,

let n be the number of vertices, and let R be the number of reflex vertices where $R = P_{V31} + P_{V32} + P_{V33} + P_{V42} + P_{V63}$. Then, the relationship between n and R is:

$$n = P_{V30} + R \quad (6.1).$$

As every orthogonal polyhedron has at least eight P_{V30} vertices, then

$$n \geq 8 + R \quad \text{or} \quad R \leq n - 8 \quad (6.2).$$

As mentioned above, the upper bound number of guards g for covering an orthogonal polyhedron is the upper bound number of dominant pieces that are totally visible from vertices added to the upper-bound number of partially visible pieces from vertices, and g is written as the following formula:

$$g \leq 3R/2 + (R/24)^{3/2} \quad (6.3).$$

Finally, substituting the equation (6.2) with (6.3) yields:

$$g \leq 3(n-8)/2 + (n-8)/24)^{3/2} \approx n^{3/2}$$

6.5 Summary

This chapter has discussed a method to obtain an upper bound of point guards for monitoring the interior of an orthogonal polyhedron. The upper bound of fixed-point guards was obtained by determining the upper bound of dominant visible pieces from any vertex of an orthogonal polyhedron and the upper bound of partially visible pieces from any vertex of an orthogonal polyhedron. The upper bound number of fixed-point guards is the sum of the dominant pieces and partially visible pieces, in which the upper bound of the fixed-point guards for any orthogonal polyhedron having n vertices is $\Omega(n^{3/2})$.

CHAPTER 7

CONCLUSION AND FUTURE RESEARCH

7.1 Conclusion

This thesis aims to provide a solution to the 3-dimensional art gallery problem, which is the natural extension to the classical art gallery problem. There has been extensive research on the classical art gallery problem, but relatively less research has been done on the 3-dimensional version of the problem. Part of the reason for this lack progress in the 3-dimensional version of the problem is the difficulty in the tetrahedralisation of polyhedra. However, the 3-dimensional version of the problem is important in many real-world applications. This is obvious when one considers the extensive use of surveillance cameras in supermarkets, banks, and many public places. In this type of applications one has to take the spatial structure of the buildings, not just the floor outline of the building, into consideration in order to provide adequate monitoring.

Current solutions to the 3-dimensional art gallery problem use mobile guards that are required to move either around faces and along the given edges in order to provide full coverage of the interior of a simple polyhedron. While this type of guards may be useful in some applications, such as deployment of human guards in an art gallery, they are not adequate in many other applications where fixed-point guards are required. This thesis presented our attempt to the 3-dimensional art gallery problem. The approach adopted in this thesis differs from the existing work in two ways: it uses fixed-point guards instead of mobile guards and it uses orthogonal pseudo-polyhedron, rather than simple orthogonal polyhedron which is a small subset of orthogonal pseudo-polyhedron, as the model for buildings.

This thesis has presented a new algorithm for determining and placement of fixed-point guards for any orthogonal pseudo-polyhedron. The algorithm involves the following steps:

1. Partition an orthogonal pseudo-polyhedron into a set of rectangular prisms.
2. Construct a visibility subset for each corner point of each rectangular prism. Each corner point can be a candidate position to station a point guard.
3. Computing the minimum number of visibility subsets that include all rectangular prisms from the above step. The computation in this step is equivalent to the Minimum Set Cover (MSC) problem, including both exact algorithm and approximated algorithms.
4. After the reduction of the number of visibility subsets in Step 3, there are still rooms to further reduce the number of guards for covering an OPP. This possibility comes from the fact that some rectangular prism could be covered by two or more guards cooperatively even though it is already covered totally by an allocated guard. In this case this allocated guard may be removed without affecting the coverage of that rectangular prism.

The result of the last two steps is the reduced number of visibility subsets, each of which corresponds to one corner point of a rectangular prism. By placing one fixed-point guard in each of these corner points, either every rectangular prism would be visible from at least one such guard, or each point in reach rectangular prism is visible from at least one such guard. Hence the entire interior of the original orthogonal pseudo-polyhedron is covered by this set of guards. This algorithm is the first algorithm known so far for covering any orthogonal pseudo-polyhedron with fixed-point guards.

The thesis has also shown that no more than $O(n^{3/2})$ fixed-point guards are required to provide full coverage of the interior of any orthogonal polyhedron. This is the lowest known upper bound for the number of fixed-point guards for any orthogonal polyhedron.

The above results are superior to those of other studies on the 3D-AGP for several reasons. First, the proposed method uses fixed-point guards. This means our method is suitable to many situations where mobile guards are not adequate. Second, our algorithm not only determines the number of guards but also provide guard placement. In contrast, for example, Souvaine *et al.* [34] provided only an upper and lower bound of the required number of face guards without any procedure for their placement. Third, Souvaine *et al.* work in [34] is only applicable to a simple orthogonal polyhedron, *i.e.*, an orthogonal polyhedron with genus 0; in contrast, our method is applicable to any orthogonal pseudo-polyhedron in which is the superset of simple orthogonal polyhedron. Finally, this study shows that the upper bound for the number of fixed-point guards required for covering any orthogonal polyhedron having n vertices is $\Omega(n^{3/2})$; meanwhile, Seidel proposed the same upper bound, but it was only applicable to one special case of simple orthogonal polyhedra [4].

The key to the 3-dimensional art gallery problem is in the handling of pseudo-polyhedron, or orthogonal pseudo-polyhedron which is the focus of this thesis. To this end, the thesis proposed a new way, which is called vertex configuration, to characterise different types of vertex in an orthogonal pseudo-polyhedron. It has shown that there are no more than 16 different vertex configurations in any orthogonal pseudo-polyhedron. We believe this result is useful in the study of orthogonal pseudo-polyhedron and it can be used as a tool in the analysis of orthogonal pseudo-polyhedron. Furthermore, we have proposed the following conjecture, known as Vertex Configuration Conjecture, which characterises the quantitative relationship between different vertex configurations in any orthogonal pseudo-polyhedron:

$$(N_{V30} + N_{V33} + 0N_{V412} + N_{V431} + 2N_{V541} + 4N_{V606} + 3N_{V633} + 2N_{V660}) \\ - (N_{V31} + N_{V32} + 3N_{V401} + 2N_{V420} + 2N_{V501} + N_{V603} + 6N_{V600} + 2N_{V630}) = 8$$

Where N_{V30} , N_{V31} , N_{V32} , N_{V33} , N_{V401} , N_{V412} , N_{V420} , N_{V431} , N_{V501} , N_{V541} , N_{V600} , N_{V603} , N_{V606} , N_{V630} , N_{V633} , and N_{V660} denote the number of vertex V30, V31, V32, V33, V40-1, V41-2, V42-0, V43-1, V50 and V54-1, V60 -01, V60-3, V60-6, V63-0, V63-3, and V66-0 are the number of vertices of the 16 different vertex configurations.

Some evidences supporting this conjecture has been provided in the thesis.

In addition, the thesis has developed a procedure for splitting an orthogonal polygon using as polyline, a procedure for splitting an orthogonal polyhedron using a polyplane. These procedures may be useful in some application such as metal fabrication.

7.2 Future Research

The following is a list of open problems for future research:

Open Problem 1: The proof of the Vertex Configuration Conjecture

Although some evidences supporting the validity of the conjecture are provided in this thesis, we have not been able to prove it. If the conjecture is proven to be true, it could be a very useful tool for studying orthogonal pseudo-polyhedron.

Open Problem 2: What is the upper-bound for our fixed-point guard placement algorithm?

In this thesis, a new algorithm has been presented for determining the number of fixed-point guards to cover any orthogonal pseudo-polyhedron. The algorithm involves two steps of

optimisation in order to reduce the number of guards. However, we have not been able to establish a non-trivial upper bound for the number of fixed-point guards from this algorithm. It is interesting to know what is the upper bound of this algorithm. Such kind of upper bounds would also be useful in the measurement of the quality of the algorithms.

For orthogonal polyhedron we have established that $\Omega(n^{3/2})$ fixed-point guards required for monitoring the orthogonal polyhedron. It would be interesting to compare this upper bound with that for orthogonal pseudo-polyhedron.

REFERENCES

- [1] V. Chavatal, "A combinatorial theorem in plane geometry," *Combinatorial Theory Series B*, vol. 18, pp. 39-41, 1975.
- [2] S. Nahar and S. Sahni, "Fast algorithm for polygon decomposition," *IEEE Transaction on Computer-Aided Design*, vol. 7, pp. 473-483, 1988.
- [3] M. d. Berg, M. v. Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry*, Second ed.: Springer, 2000.
- [4] J. O'Rourke, *Art Gallery Theorems and Algorithms*: Oxford University Press 1987, 1987.
- [5] T. C. Shermer, "Recent results in art galleries," in *Proceedings of IEEE*, 1992, pp. 1384 -1399.
- [6] J. Urrutia, *Art gallery and Illumination Problems*. Amsterdam: Elsevier Publisher, 2000.
- [7] S. Fisk, "A short proof of Chavatal's watchman theorem," *Combinatorial Theory Series B*, vol. 24, p. 374, 1978.
- [8] D. Avis and G. T. Toussaint, "An efficient algorithm for decomposing a polygon into star-shaped polygons," *Pattern Recognition*, vol. 13, pp. 395-398, 1981.
- [9] M. Garey, D. Johnson, F. Preparata, and R. Tarjan, "Triangulating a simple polygon," *Information Processing Letters*, vol. 7, pp. 175-179, 1978.
- [10] Chazelle, "Triangulating a simple polygon in linear time," *Discrete and Computational Geometry*, vol. 6, pp. 485-524, 1991.
- [11] A. A. Kooeshesh and B. E. Moret, "Three-coloring the vertices of a triangulated simple polygon," *Pattern Recognition*, vol. 25, p. 443, 1992.
- [12] J. Khan, M. Klawe, and D. Kleitman, "Traditional galleries require fewer watchmen," *SIAM Journal on Algebraic and Discrete Methods*, vol. 4, pp. 194-206, 1983.
- [13] J. O'Rourke, "An alternate proof of the rectilinear art gallery theorem," *Journal of Geometry*, vol. 211, pp. 118-130, 1983.
- [14] H. Edelsbrunner, J. O'Rourke, and E. Welzl, "Stationing guards in rectilinear art galleries," *Computer Vision, Graphics, Image processing*, vol. 27, pp. 167-176, 1984.
- [15] J. R. Sack, "Rectilinear computational geometry," Ph. D, School of Computer Science, McGill University, 1984.
- [16] I. Bjorling-Sachs and D. L. Souvaine, "An efficient algorithm for guard placement in polygons with holes," *Discrete and Computational Geometry*, vol. 13, pp. 77-109, 1995.
- [17] D. T. Lee and A. K. Lin, "Computational complexity of art gallery problems," *IEEE Transaction on Information Theory Archive*, vol. 32, pp. 276-282, 1986.
- [18] D. Schuchardt and H. D. Hecker, "Two NP-hard art-gallery problems for orthogonal polygons," *Mathematical Logic Quarterly*, vol. 41, pp. 261-267, 1995.
- [19] J. O'Rourke, "Galleries need fewer mobile guards: a variation on Chavatal's theorem," *Geometriae Dedicata*, vol. 14, pp. 273-283, 1983.
- [20] I. Bjorling-Sachs, "Edge guards in rectilinear polygons," *computational Geometry: Theory and Applications*, vol. 11, pp. 111-123, 1998.
- [21] E. Gyori, F. Hoffmann, K. Kriegel, and T. Shermer, "Generalized guarding and partitioning for rectilinear polygons," *Computational Geometry: Theory and Applications*, vol. 6, pp. 21-44, 1996.
- [22] S. K. Ghosh, "Approximation algorithms for art gallery problems," in *Proceeding of Canadian Information Processing Society Congress*, 1987, pp. 429-434.
- [23] A. P. Tomas, A. L. Bajuelos, and F. Marques, "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.
- [24] Y. Amit, J. S. B. Mitchell, and E. Packer, "Locating guards for visibility coverage of polygons," 2007.

- [25] A.L.Bajuelos, A. M. Martins, S. Canales, and G. Hernandez, "Metaheuristic Approaches for the Minimum Vertex Guard Problem," in *Third International Conference on Advanced Engineering Computing and Applications in Sciences*, 2009, pp. 77-82.
- [26] I. Saleh, "K-vertex guarding simple polygon," *Computational Geometry*, vol. 42, p. 10, 2008.
- [27] C. Fragoudakis, E. Markou, and S. Zachos, "How to place efficiently guards and paintings in an art gallery," in *10th Panhellenic conference on Informatic*, 2005, pp. 145-154.
- [28] A.L.Bajuelos, S. Canales, G. Hernandez, and A. M. Martins, "Estimating the maximum hidden vertex set in polygons," in *International conference on Computational Sciences and Its Applications ICSA*, 2008, pp. 421-432.
- [29] S. Rana, "Use of GIS for planning visual surveillance installations," in *ESRI Homeland Security GIS Summit*, 2005.
- [30] M. I. Karavelas, "Guarding curvilinear art galleries with edge or mobile guards," in *ACM Symposium on Solid and Physical Modeling*, 2008.
- [31] D.Eppstein, M.T.Goodrich, and N.Sitchinava, "Guard placement for Wireless Localization," 2006.
- [32] C. D. Toth, "Art gallery problem with guards whose range vision is 180o," *Computational Geometry*, vol. 17, pp. 121-134, 2000.
- [33] P. Bose, T. Shermer, G. Toussaint, and B. Zhu, "Guarding polyhedral terrains," *Computational Geometry*, vol. 7, pp. 173-185, 1997.
- [34] D. L. Souvaine, R. Veroy, and A. Winslow, "Face guards for art galleries," presented at the XIV Spanish Meeting on Computational Geometry, Spanish, 2011.
- [35] J. O'Rourke, *Computational Geometry in C*, Second ed. Cambridge: Cambridge University Press, 1998.
- [36] F. Yamaguchi, *Computer-Aided Geometric Design*: Springer, 2002.
- [37] F. P. Preparata and M. I. Shamos, *Computational Geometry an Introduction*. New York: Springer-Verlag, 1985.
- [38] B. Genc, "Reconstruction of orthogonal polyhedra," PhD, University of Waterloo, Ontario, Canada, 2008.
- [39] B. Yip and R. Klette, "Angle Counts for Isothetic Polygons and Polyhedra," *Pattern Recognition Letter*, vol. 24, pp. 1275-1278, 2003.
- [40] J. M. Keil, "Polygon decomposition," University of Saskatchewan, Saskatoon, Canada 1996.
- [41] K. Voss, *Discrete Images, Objects, and Functions in Z^n* . Berlin: Springer, 1993.
- [42] G. T. Toussaint, "Pattern recognition and geometrical complexity," in *Fifth International Conference on Pattern Recognition* 1985, pp. 1324-1347.
- [43] T. Asano, T. Azano, and H. Imai, "Partitioning a polygonal region into trapezoids," *Journal of the ACM*, vol. 33, pp. 290-312, 1986.
- [44] J. O'Rourke and G. Tewari, "Partitioning orthogonal polygons into fat rectangles in polynomials time," in *Proceeding of 14th Canadian Conference Computational Geometry*, 2002, pp. 97-100.
- [45] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge: The MIT Press, 1990.
- [46] L. P. Ku and H. W. Leong, "Optimum partitioning problem for rectilinear VLSI layout," National University of Singapore, Singapore 1995.
- [47] V. H. Nguyen, "Optimum partitioning of rectilinear layouts," in *IEEE proceeding Computing Digital Technology*, 1996.
- [48] W. T. Liou, J. J. Tan, and R. C. T. Lee, "Minimum rectangular partition problem for simple rectilinear polygons," *IEEE Transaction on Computer-Aided Design*, vol. 9, 1990.
- [49] M. A. Lopez and D. P. Mehta, "Efficient decomposition of polygons into L-shapes with application to VLSI layouts," *ACM Transaction on Design Automation of Electronic System*, vol. 1, 1996.
- [50] H. S. M. Coxeter, *Regular polytopes*. New York: Dover Publications, 1973.

- [51] J. R. Rossignac and A. A. G. Requicha, "Constructive Non-Regularized Geometry," *Computer - Aided Design*, vol. 23, pp. 21-32, 1991.
- [52] 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.
- [53] J. D. Foley, A. v. Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice in C*, 2nd edition ed.: Addison-Wesley, 1996.
- [54] T. Biedl and B. Genc, "Reconstructing orthogonal polyhedra from putative vertex sets," *technical reports*, 2007.
- [55] M. J. Wenninger, *Polyhedron Models*: Cambridge University Press, 1974.
- [56] 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.
- [57] R. Juan-Arinyo, "Domain extension of isothetic polyhedra with minimal CSG representation," *Computer Graphics Forum*, vol. 5, pp. 281-293, 1995.
- [58] D. Eppstein and E. Mumford, "Steinitz theorems for orthogonal polyhedra," in *SoCG '10 Proceedings of the 2010 annual symposium on Computational geometry*, 2010, pp. 429-438.
- [59] T. Biedl and B. Genc, "When can a graph form an orthogonal polyhedron?," presented at the CCCG 2004, Montreal, 2004.
- [60] B. Yang and C. A. Wang, "Minimal tetrahedralizations of a class of polyhedra," *Journal of Combinatorial Optimazation*, vol. 8, pp. 241-265, 2004.
- [61] N. J. Lennes, "Theorem on the simple finite polygon and polyhedron," *American Journal of Mathematics*, vol. 33, pp. 37-62, 1911.
- [62] J. Ruppert and R. Seidel, "On the difficulty of tetrahedralizing 3-dimensional non-convex polyhedra," in *Proceedings of the Fifth Annual Symposium on Computational Geometry*, 1989, pp. 380 – 392.
- [63] A. Below, U. Brehm, J. A. D. Loera, and J. Richter-Gebert, "Minimal simplicial dissections and triangulations of convex 3-polytopes," *Discrete and Computational Geometry*, vol. 24, pp. 35-48, 2000.
- [64] K. Chen, I. Hsich, and C. A. Wang, "A genetic algorithm for minimum tetrahedralization of a convex polyhedron," in *Proceedings of the 15th Canadian Conference on Computational Geometry*, 2003, pp. 115-119.
- [65] V. J. Dielissen and A. Kaldewij, "Rectangular partition is polynomial in two dimensions but NP-complete in three," *Information Processing Letters*, vol. 38, pp. 1-6, 1991.
- [66] D. Stolee. (2008). *Minimum Rectilinear Partitioning*. Available: www.math.unl.edu/~s-dstolee1/Presentations/Sto08-MinRectPart.pdf
- [67] M. Sprankle and J. Hubbard, *Problem solving and programming concept*, eighth ed. New Jersey: Person Prentice Hall, 2009.
- [68] J. O'Rourke, "Uniqueness of orthogonal connect-the-dots," *Machine Intelligence and Pattern Recognition: Computational Morphology*, pp. 97–104, 1988.
- [69] B. G. Baumgart, "A polyhedron representation for computer vision," in *National computer conference and exposition Anaheim, California 1975*, pp. 589-596
- [70] K. Weiler, "Edge-Based Data Structure for Solid Modlling in Curve-Surface Environments," *IEEE Computer Graphics and Application*, vol. 5, pp. 21-40, 1985.
- [71] L. Guibas and J. Stolfi, "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams," *ACM Transaction on Graphics*, vol. 4, pp. 74-123, July 1985.
- [72] O. Bournez, O. Maler, and A. Pnueli, "Orthogonal Polyhedra: Representation and Computation," 1999.
- [73] A. Aquilera and D. Ayala, "Orthogonal polyhedra as geometric bounds in constructive solid geometry," *Solid Modeling*, p. 12, 1997.

- [74] M. Karasick, "On the representation and manipulation of rigid solids," PhD, Computer science, McGill University, Montreal, Quebec, Canada, 1988.
- [75] C. L. Bajaj and T. K. Dey, "Zone Theorem and Polyhedral Decompositions," Purdue University 8-3-1990 1990.
- [76] D. Ayala and J. Rodriguez, "Connected component labeling based on the EVM Model," in *The 18th spring conference on Computer graphics*, 2002, pp. 63-71.
- [77] J. Czyzowicz, E. Rivera-Campo, and J. Urrutia, "Illuminating rectangles and triangles in the plane," *Journal of Combinatorial Theory Series B archive*, vol. 57, 1993.
- [78] S. L. Senk, *Advanced Algebra*. Chicago: Scott Foresman/Addison Wesley, 1998.
- [79] K. Daniels, V. J. Milenkovic, and D. Roth, "Finding the Largest Rectangle in Several Classes of Polygons," Harvard University, Cambridge 1995.
- [80] M. T'anase, R. C. Velkamp, and H. Haverkort, "Multiple Polyline to Polygon Matching," Utrecht University TR UU-CS-2005-0172005.
- [81] L. Wu, G. Tian, and Z. Xie, "An algorithm for splitting arbitrary polygon," presented at the 2009 Fifth International Conference on Natural Computation, 2009.
- [82] M. S. Paterson and F. F. Yao, "Optimal binary space partitions for orthogonal objects," *Journal of Algorithms archive*, vol. 13, pp. 100-106, March 1992 1992.
- [83] A. B. Shiflet, *Elementary data structure with Pascal*: West Publishing Company, 1990.
- [84] J. Lien and N. Amato, "Approximate convex decomposition of polyhedra and its applications," *Computer aided geometric design*, vol. 25, pp. 503-522, 2008.
- [85] C. Ericson, *Real-time collision detection*. San Francisco: Morgan Kaufmann 2005.
- [86] C. L. Bajaj and T. K. Dey, "Convex decomposition of polyhedra and robustness," *SIAM Journal on Computing*, vol. 21, pp. 339-364, 1992.
- [87] C. L. Bajaj and V. Pascucci, "Splitting a complex of convex polytopes in any dimension," in *The twelfth annual symposium on Computational geometry*, 1996.
- [88] W. Cheney and D. Kincaid, *Linear Algebra Theory and Applications*. Sudbury: Jones and Bartlett Publishers, 2009.
- [89] F. V. Fomin and D. Kratsch, *Exact Exponential Algorithms*. Verlag Berlin Heidelberg: Springer, 2010.
- [90] S. Ceria, P. Nobili, and A. Sassano, "A lagrangian-based heuristic for large-scale set covering problems," *Mathematical Programming*, vol. 81, pp. 215-258, 1995.
- [91] E. Boros, P. L. Hammer, T. Ibaraki, and A. Kogan, "Logical analysis of numerical data," *Mathematical Programming*, vol. 79, pp. 163-190, 2000.
- [92] D. Buezas, "Constraint-based modeling of Minimum Set Covering: Application to Species Differentiation," Master, Departamento de Informática, Universidade Nova de Lisboa, Lisboa, 2010.
- [93] A. Caprara, M. Fischetti, and P. Toth, "Algorithm for Set Covering Problems," University of Bologna and University of Padova 1998.
- [94] V. V. Vazirani, *Approximation algorithm*. New York: Springer, 2003.
- [95] A. Schrijver, *Theory of Linear and Integer Programming*. Amsterdam: John Wiley & Sons, 1998.

APPENDIX 1

The solution of 3D-AGP case (input as Table 5.1) by using dual simplex algorithm

```
>> dsimplex(type,c,a,b)
```

```
Initial tableau
```

```
a =
```

```
Columns 1 through 20
```

```
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 -1 -1 0 0 0
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 0 0 0 0 0 0 0
-1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 -1 -1 0 -1 -1 0 -1 -1
-1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 -1 -1 -1 -1 -1
0 0 0 0 0 0 0 0 0 0 0 0 -1 -1 -1 -1 -1 -1 0 0
-1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0 0 0
-1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
Columns 21 through 40
```

```
-1 -1 0 -1 -1 0 0 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-1 -1 0 -1 -1 0 0 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0
-1 -1 0 -1 -1 0 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 -1 -1 0 -1 -1 -1 0
-1 -1 -1 -1 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-1 -1 0 -1 -1 0 0 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0
-1 -1 0 -1 -1 0 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
0 0 0 0 0 0 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
Columns 41 through 50
```

```
1 0 0 0 0 0 0 0 0 -1
0 1 0 0 0 0 0 0 0 -1
0 0 1 0 0 0 0 0 0 -1
0 0 0 1 0 0 0 0 0 -1
0 0 0 0 1 0 0 0 0 -1
0 0 0 0 0 1 0 0 0 -1
0 0 0 0 0 0 1 0 0 -1
0 0 0 0 0 0 0 1 0 -1
0 0 0 0 0 0 0 0 1 -1
0 0 0 0 0 0 0 0 0 0
```

Press any key to continue ...

```
pivot row-> 1 pivot column-> 1
```

```
Tableau 1
```

```
a =
```

```
Columns 1 through 20
```

```
1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0
0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 -1 -1 0 -1 -1 0 -1 -1
0 0 1 0 0 1 0 0 1 0 0 1 0 0 -1 0 0 -1 -1 -1
0 0 0 0 0 0 0 0 0 0 0 -1 -1 -1 -1 -1 -1 0 0
0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
Columns 21 through 40
```

```
1 1 0 1 1 0 0 0 1 1 0 1 1 0 1 1 0 1 1 0
1 1 0 1 1 0 0 0 1 1 0 1 1 0 1 1 0 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-1 -1 0 -1 -1 0 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 -1 0 0 -1 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0
-1 -1 -1 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```

0 0 0 0 0 0 0 0 0 0 0 -1 0 0 -1 0 0 -1 0 0 -1
0 0 0 0 0 0 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1

```

Columns 41 through 50

```

-1 0 0 0 0 0 0 0 0 0 1
-1 1 0 0 0 0 0 0 0 0
-1 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 -1
-1 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 -1
-1 0 0 0 0 0 1 0 0 0
-1 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 -1
1 0 0 0 0 0 0 0 0 -1

```

Press any key to continue ...

pivot row-> 4 pivot column-> 13

Tableau 2

a =

Columns 1 through 20

```

1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 -1 -1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 -1
0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1
0 0 1 0 0 1 0 0 1 0 0 1 0 0 -1 0 0 -1 -1 -1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 -1 1 1
0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1

```

Columns 21 through 40

```

0 0 0 0 0 0 -1 -1 1 1 0 1 1 0 1 1 0 1 1 0
0 0 0 0 0 0 -1 -1 1 1 0 1 1 0 1 1 0 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 -1 0 0 -1 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 -1 0 0 -1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 -1 0 0 -1 0 0 -1 0 0 -1
0 0 0 0 0 0 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1

```

Columns 41 through 50

```

-1 0 0 1 0 0 0 0 0 0
-1 1 0 1 0 0 0 0 0 -1
-1 0 1 0 0 0 0 0 0 0
0 0 0 -1 0 0 0 0 0 1
-1 0 0 0 1 0 0 0 0 0
0 0 0 -1 0 1 0 0 0 0
-1 0 0 0 0 0 1 0 0 0
-1 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 -1
1 0 0 0 0 0 0 0 0 -1

```

Press any key to continue ...

pivot row-> 2 pivot column-> 19

Tableau 3

a =

Columns 1 through 20

```

1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0
0 0 1 0 0 1 0 0 1 0 0 1 0 0 -1 0 0 -1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 -1 0 0

```

```

0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0

```

Columns 21 through 40

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1 1 0 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0
0 0 -1 0 0 -1 0 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0
0 0 -1 0 0 -1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 -1 0 0 -1 0 0 -1 0 0 -1
0 0 0 0 0 0 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
0 0 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1

```

Columns 41 through 50

```

0 -1 0 0 0 0 0 0 0 1
1 -1 0 -1 0 0 0 0 0 1
-1 0 1 0 0 0 0 0 0 0
-1 1 0 0 0 0 0 0 0 0
0 -1 0 -1 1 0 0 0 0 1
-1 1 0 0 0 1 0 0 0 -1
-1 0 0 0 0 0 1 0 0 0
-1 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 -1
0 1 0 1 0 0 0 0 0 -2

```

Press any key to continue ...

pivot row-> 6 pivot column-> 41

Tableau 4

a =

Columns 1 through 20

```

1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 -1 1 1
0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0
0 0 1 0 0 1 0 0 1 0 0 1 0 0 -1 0 0 -1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0
0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0
0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0

```

Columns 21 through 40

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 -1 0 0 -1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 -1 0 0 -1 0 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0
0 0 1 0 0 1 0 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0
0 0 1 0 0 1 0 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 -1 0
0 0 1 0 0 1 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
0 0 0 0 0 0 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
0 0 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1

```

Columns 41 through 50

```

0 -1 0 0 0 0 0 0 0 1
0 0 0 -1 0 1 0 0 0 0
0 -1 1 0 0 -1 0 0 0 1
0 0 0 0 0 -1 0 0 0 1
0 -1 0 -1 1 0 0 0 0 1
1 -1 0 0 0 -1 0 0 0 1
0 -1 0 0 0 -1 1 0 0 1
0 -1 0 0 0 -1 0 1 0 1
0 0 0 0 0 0 0 1 -1
0 1 0 1 0 0 0 0 0 -2

```

Press any key to continue ...

pivot row-> 9 pivot column-> 29

Tableau 5

A =

Columns 1 through 20

1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	-1
0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0
0	0	1	0	0	1	0	0	1	0	0	1	0	0	-1	0	0	-1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Columns 21 through 40

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	-1	0	0	-1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	0	0	1
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	-1	0	0	-1	0	0	0	0	1	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Columns 41 through 50

0	-1	0	0	0	0	0	0	0	1
0	0	0	-1	0	1	0	0	0	0
0	-1	1	0	0	-1	0	0	-1	2
0	0	0	0	0	-1	0	0	0	1
0	-1	0	-1	1	0	0	0	-1	2
1	-1	0	0	0	-1	0	0	-1	2
0	-1	0	0	0	-1	1	0	-1	2
0	-1	0	0	0	-1	0	1	-1	2
0	0	0	0	0	0	0	0	-1	1
0	1	0	1	0	0	0	0	1	-3

Press any key to continue ...

Problem has a finite optimal solution

Values of the legitimate variables:

x(1)= 1.000000
x(2)= 0.000000
x(3)= 0.000000
x(4)= 0.000000
x(5)= 0.000000
x(6)= 0.000000
x(7)= 0.000000
x(8)= 0.000000
x(9)= 0.000000
x(10)= 0.000000
x(11)= 0.000000
x(12)= 0.000000
x(13)= 1.000000
x(14)= 0.000000
x(15)= 0.000000
x(16)= 0.000000
x(17)= 0.000000
x(18)= 0.000000
x(19)= 0.000000
x(20)= 0.000000
x(21)= 0.000000
x(22)= 0.000000
x(23)= 0.000000
x(24)= 0.000000
x(25)= 0.000000
x(26)= 0.000000
x(27)= 0.000000
x(28)= 0.000000
x(29)= 1.000000

x(30)= 0.000000
x(31)= 0.000000
x(32)= 0.000000
x(33)= 0.000000
x(34)= 0.000000
x(35)= 0.000000
x(36)= 0.000000
x(37)= 0.000000
x(38)= 0.000000
x(39)= 0.000000
x(40)= 0.000000

Objective value at the optimal point: $z = 3.000000$

APPENDIX 2

The solution of 3D-AGP case (input as Table 5.2) by using dual simplex algorithm

```
>> dsimplex(type,c,a,b)
```

```
Initial tableau
```

```
a =
```

```
Columns 1 through 21
```

```
-1 -1 -1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
-1 -1 -1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
-1 -1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
-1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
-1 -1 -1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
0 -1 -1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
-1 -1 -1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 -1
-1 -1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 -1
-1 -1 -1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 -1
0 -1 -1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 -1
-1 -1 -1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 -1
-1 -1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 -1
-1 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 -1
-1 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 -1
0 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1
0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 -1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
Press any key to continue ...
```

```
pivot row-> 1 pivot column-> 1
```

```
Tableau 1
```

```
a =
```

```
Columns 1 through 21
```

```
1 1 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 -1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 -1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 -1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 -1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 -1 -1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 -1
0 0 0 -1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 -1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 -1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 -1 -1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 -1
0 0 0 -1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 -1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 -1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 -1
0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 -1
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
```

```
Press any key to continue ...
```

```
pivot row-> 6 pivot column-> 2
```

```
Tableau 2
```

```
a =
```

```
Columns 1 through 21
```

```
1 0 0 -1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 -1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 -1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 -1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 -1
0 0 0 -1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 -1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 -1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 -1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 -1 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 -1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 -1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 -1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
```

```

0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 1 0 0
0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1

```

Press any key to continue ...
pivot row-> 4 pivot column-> 4
Tableau 3
a =

Columns 1 through 21

```

1 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 1 0 -1 0 -1 0 0 0 0 0 0 0 0 0 0 0 1
0 0 1 0 0 1 -1 0 -1 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 1 0 0 -1 0 -1 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 -1 1 -1 0 0 0 0 0 0 0 0 0 0 0 1
0 1 1 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 -1 0 -1 1 0 0 0 0 0 0 0 0 0 0 1
0 0 1 0 0 0 -1 0 -1 0 1 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 -1 0 -1 0 0 1 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 -1 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 -1 0 -1 0 0 0 0 1 0 0 0 0 0 0 1
0 0 1 0 0 0 -1 0 -1 0 0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 -1 0 -1 0 0 0 0 0 0 1 0 0 0 0 1
0 0 0 0 0 0 -1 0 -1 0 0 0 0 0 0 0 1 0 0 0 1
0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 1 0 0 0
0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1
0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 -2

```

Press any key to continue ...
pivot row-> 17 pivot column-> 3
Tableau 4

a =

Columns 1 through 21

```

1 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 1 0 -1 0 -1 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 1 -1 0 -1 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 1 0 0 -1 0 -1 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 -1 1 -1 0 0 0 0 0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 -1 0 -1 1 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 -1 0 -1 0 1 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 -1 0 -1 0 0 1 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 -1 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 -1 0 -1 0 0 0 0 1 0 0 0 0 0 0 1
0 0 0 0 0 0 -1 0 -1 0 0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 -1 0 -1 0 0 0 0 0 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 1
0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 -2

```

Press any key to continue ...
Problem has a finite optimal solution
Values of the legitimate variables:
x(1)= 1.000000
x(2)= 0.000000
x(3)= 1.000000
Objective value at the optimal point:
z= 2.000000