EXTRACTING AUXETIC PATTERNS FROM MESHES FOR 3D PRINTING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

LEVEND MEHMET MERT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

FEBRUARY 2018

Approval of the thesis:

EXTRACTING AUXETIC PATTERNS FROM MESHES FOR 3D PRINTING

submitted by **LEVEND MEHMET MERT** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** ______________

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** ______________

Assoc. Prof. Dr. Yusuf Sahillioğlu
Supervisor, **Computer Engineering Department, METU** ______________

Assist. Prof. Dr. Ulaş Yaman
Co-supervisor, **Mechanical Engineering Department, METU** ______________

Examining Committee Members:

Prof. Dr. Tolga Can
Computer Engineering Department, METU ______________

Assoc. Prof. Dr. Yusuf Sahillioğlu
Computer Engineering Department, METU ______________

Assist. Prof. Dr. Ufuk Çelikcan
Computer Engineering Department, HU ______________

Date: ______________

ABSTRACT

EXTRACTING AUXETIC PATTERNS FROM MESHES FOR 3D PRINTING

Mert, Levend Mehmet
M.S., Department of Computer Engineering
Supervisor : Assoc. Prof. Dr. Yusuf Sahillioğlu
Co-Supervisor : Assist. Prof. Dr. Ulaş Yaman

February 2018, 136 pages

3D printing is a manufacturing process which generates a physical, three-dimensional object from a digital design. In our day, there exist lots of devices, called 3D printers, having different working principles and using various materials for this manufacturing process. A model designed digitally can be transformed into a three-dimensional object in the real world by 3D printers. The elasticity of objects created by 3D printers depends on the materials used during manufacturing and the digital designs. If the digital design would be transformed into an object after 3D printing is modified properly, the printed object can be more flexible even though the material used during manufacturing remains the same.

In this thesis, we propose methods for extracting auxetic patterns from 3D printable digital designs by modifying the existing mesh primitives directly and fully-automatically. Proposed methods were employed to extract auxetic patterns from some digital designs. These extracted auxetic patterns were 3D printed. It was observed that 3D printed objects from extracted auxetic patterns are more flexible when they are compared to 3D printed objects from the original digital designs. Benefits of extracting auxetic patterns from digital designs and 3D print them instead of the original digital designs were revealed. Besides, challenges of 3D printing extracted auxetic patterns and how to deal with these challenges were also referred. Finally, constraints and performances of the proposed methods were explained and acquired results of these methods for different digital designs were compared.

Keywords: Auxetic, Texture Synthesis, Filigree Synthesis, Geometry Processing, 3D Printing, Digital Fabrication

ÖZ

EKLEMELİ ÜRETİM İÇİN ÖRGÜ YAPILARINDAN OGZETİK DESENLERİN ÇIKARTILMASI

Mert, Levend Mehmet
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Yusuf Sahillioğlu
Ortak Tez Yöneticisi : Yrd. Doç. Dr. Ulaş Yaman

Şubat 2018 , 136 sayfa

Eklemeli üretim dijital bir tasarımdan fiziksel, 3 boyutlu bir nesne üretme işlemidir. Günümüzde, bu işlem için farklı çalışma prensipleri olan ve çeşitli malzemeler kullanabilen bir çok 3B yazıcı mevcuttur. Dijital ortamda oluşturulan bir tasarım bu 3B yazıcılar sayesinde gerçek dünyada 3 boyutlu bir nesneye dönüştürülebilir. Eklemeli üretim ile üretilen nesnelerin esneklikleri; üretim için kullanılan malzemelere ve dijital tasarımlara bağlıdır. Eklemeli üretim sonucu nesneye dönüşecek bir dijital tasarım uygun şekilde değiştirilirse üretimde kullanılacak malzeme aynı kalmasına rağmen elde edilecek nesne daha esnek olabilir.

Bu tez kapsamında eklemeli üretim ile üretilebilecek dijital tasarımların; temel özellikleri doğrudan değişmeyecek şekilde, otomatik olarak esnek özellik gösteren desenlerle kaplanması için yöntemler önerilmektedir. Önerilen yöntemler kullanılarak bazı dijital tasarımlar esnek özellik gösteren desenlerle kaplanmışlardır. Bazı dijital tasarımların esnek özellik gösteren desenlerle kaplanmış halleri eklemeli üretim ile üretilmişlerdir.Üretilen nesnelerin, dijital tasarımların orjinal hallerinin aynı malzeme kullanılarak eklemeli üretim ile üretilmesi sonucu elde edilecek nesnelerden daha fazla esnek özelliğe sahip oldukları gözlenmiştir. Dijital tasarımların orjinal halleri yerine esnek özellik gösteren desenlerle kaplanmış hallerinin eklemeli üretim ile üretilmesinin sağladığı faydalar belirlenmiştir. Bu faydalara ek olarak, dijital tasarımların orjinal halleri yerine esnek özellik gösteren desenlerle kaplanmış hallerinin eklemeli üretim sürecine getirdiği zorluklar ve bu zorlukları aşmak için yapılabilecekler de belirlenmiştir. Son olarak dijital tasarımları esnek özellik gösteren desenlerle kaplamak için önerilen yöntemlerin kısıtlarından ve başarımlarından bahsedilmiş, farklı dijital tasarımlar üzerinde elde edilen sonuçlar kıyaslanmıştır.

Anahtar Kelimeler: Ogzetik, Yapı Sentezleme, Desen Sentezleme, Geometri İşleme, 3B Yazdırma, Dijital Üretim

To my lovely family

ACKNOWLEDGMENTS

I would first like to thank my thesis advisor Assoc. Prof. Dr. Yusuf Sahillioğlu and co-advisor Asst. Prof. Dr. Ulaş Yaman for their surveillance, guidance and encouragement.

I would like to thank my employer ASELSAN A.Ş. for allowing me to proceed my education and use its resources during my education. I also thank my superiors and colleagues for supporting me.

I wish to thank my friends who have helped and inspired me during my thesis study.

I would like to thank my fiancee for her patience and support.

Finally, I am grateful to my family for raising me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

3D	Three Dimensional
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
DLP	Digital Light Processing
DOF	Degree Of Freedom
mm	Millimeter

CHAPTER 1

INTRODUCTION

3D printing is current technology for creating physical, three-dimensional objects from digital designs. Nowadays, 3D printers are widely used in various fields from homes to the industry. There are lots of 3D printers using diverse materials and based on different technologies. Although there exists 3D printers using diverse materials, using plastics are more common because of their price. Besides, plastics are also cheaper than other materials used by 3D printers. These plastics used by 3D printers have different flexibility characteristics. If an elastic object is desired as the output of the 3D printer, the material used by the 3D printer also has to be elastic. 3D printers just print the digital designs with specified materials, they are not aware of the digital designs' flexibility.

However, digital designs can be modified to behave elastically when they are 3D printed even though the material used during 3D printing is not very elastic. This can be done by changing the surface of the digital design with connected auxetic patterns. The term auxetic refers a structure that becomes thicker when it stretched. These structures bring flexibility to the objects. In other words, objects consisting of auxetic patterns would be elastic.

Replacing surfaces of digital designs is always a popular topic in computer graphics world. Many types of research have been studied on this topic for different purposes and obtaining 3D printed elastic objects is also one of them. Various complex methods are proposed to change surfaces of digital designs in order to 3D print them and obtain elastic objects with inelastic materials.

How to 3D print auxetic structures is another subject for computer graphics society because they are vacuolar structures. 3D printing vacuolar structures vertically is very hard without using internal support structures. If internal support structures are employed, then finally obtained 3D printed object deviates from the original digital design.

1.1 Motivation

In this thesis, we put forth new simple methods different from existing ones to extract auxetic patterns from digital designs. We also propose 3D printing of these extracted auxetic patterns instead of original digital designs in order to acquire elastic objects. Additionally, we suggest a procedure to generate digital designs consisting of auxetic patterns which can create simple elastic objects and cover complex shapes when they are 3D printed. We promise elasticity for objects 3D printed with inelastic material by using auxetic patterns.

1.2 Contributions

The main contributions of this thesis are as follows:

- In contrast to many pattern synthesis techniques that introduce additional primitives such as curve networks, we modify the existing mesh primitives into auxetic patterns directly and fully-automatically. To this end, novel auxetic pattern extraction algorithms are designed for triangle and quadrilateral meshes.
- Simple shapes consisting of auxetic patterns are generated in order to cover complex shapes.
- Elastic fabrication is achieved by using auxetic tiling of inelastic and cheap material.

1.3 Thesis Organization

The rest of this thesis is organized as follows:

Chapter 2 presents the related work on extracting patterns from digital models for 3D printing and known and potential application areas of auxetics.

Chapter 3 describes auxetic patterns extraction processes from digital models and how to prepare this extracted auxetic patterns for 3D printing.

In Chapter 4, benefits of 3D printing extracted auxetic patterns instead of digital models and of difficulties of 3D printing extracted auxetic patterns are discussed. Besides restrictions and success of our auxetic pattern extraction methods are reported.

In Chapter 5, the conclusion of this thesis and future work can be done on this thesis are emphasized.

CHAPTER 2

RELATED WORK

Texture and filigree synthesis for digital fabrication, in other words; 3D printing, are important topics in computer graphics community and there are lots of studies carried out on these topics in recent years. Although these studies have different purposes, for example; visuality, flexibility, cost-effective manufacturing, fast prototyping; they try to achieve the same thing: replacing the objects' surfaces with patterns.

In this chapter, previous studies about texture, filigree and pattern synthesis for 3D printing having various purposes are classified and expressed. Finally, application areas of auxetics are stated.

2.1 Studies On Visuality

The aesthetic appearance of the digital models is quite important in computer graphics as these models lend themselves to subsequent processes such as video games and fabrication. There exist lots of studies about texture synthesis which enhances the aesthetic appearance. Wei et al. [5] made an extensive survey on texture synthesis methods. Different methods were designed on image-based texture synthesis such as covering target surface with patches [6], mapping texture directly on the target surface [7]. Zhou et al. [8] also designed a method which is named Mesh Quilting for geometric texture synthesis. Ma et al. [9], developed a data-driven method for synthesizing large domains with small input textures. Garg et al. [10] presented an approach to design meshes consisting of woven wires. Their approach combines small wires woven in a plain weave to create wire meshes which represent large objects.

Torres et al. [11] stated that haptic characteristics which contribute to aesthetic value are generally lacking in 3D printed objects. Therefore, they created a tool for mapping texture elements onto objects for 3D printing. Dumas et al. [12] also synthesize patterns which are not only fully connected but also structurally stable for 3D printing along surfaces. Additionally, Chen et al. [13] managed to automatically synthesize set of inputs creating filigrees together for target surfaces of objects. Their automatic filigree creation results are also 3D printable. Furthermore, a tool for designers created by Schumacher, Thomaszewski and Gross [14] in order to generate surfaces structured with decorative patterns. The user specifies the decorative pattern and target shape and tool covers the target shape with the given decorative pattern while considering the target shape's durableness. Therefore, target shape with covered decorative pattern can be 3D printed. Zhender et al. [15] also develop a tool for designing ornamental surfaces consisting of curve networks. Their tool admits user-defined spline curves as an essential design parameter. Therefore, the user can control aesthetic. Besides, their tool considers the soundness of the surfaces for 3D printing.

Besides, Zhao et al. [16] have a different perspective. They focused on how to 3D print efficiently objects synthesized with patterns. They suggest synthesizing surfaces with connected spirals which can be 3D printed without on-off switching of the print nozzle. Their approach selects continuity of a tool path followed by the print nozzle during the 3D printing operation as a baseline.

2.2 Studies On Controlling Elasticity

Elastic models appear in many computer graphics applications ranging from deformation [17] to non-rigid correspondence [18]. It is also a desirable feature in real-world models such as toys and tools. Spillmann and Teschner [19] proposed an approach to modeling objects as networks of elastic rods. Perez et al. [20] creates a tool for 3D printing deformable objects consisting of rod meshes. Their tool gets a deformable object and a set of deformed poses of its and generates 3D printable rod mesh which resembles the input object. When generated rod mesh is 3D printed, it exhibits the desired deformation.

Panetta et al. [21] designed a pattern family consisting of small structures which are 3D printable with a single material and without internal supports. Their patterns are also elastic. They can combine these small patterns from their designed pattern family in order to fabricate large objects having desired mechanical behaviors. Similarly, Schumacher et al. [22] proposed a method for 3D printing objects having different elastic properties at different parts. They generated microstructures to control elasticity at different parts of the objects. Their method enables to print 3D objects consisting of this microstructures and demonstrate varied elastic properties at desired parts of these objects with a single material. Additionally, Martinez et al. [23] used microstructures inspired by Voronoi open-cell foams for creating 3D printed elastic objects. They allow the user to specify elasticity on different parts of the model. Then, they generate corresponding structures in order to provide these elasticities. These generated structures also can be 3D printed directly.

Konakovic et al. [4] and Guseinov et al. [3] also have methods to design elastic objects from 3D printed flat surfaces consisting of patterns which provide elasticity.

2.3 Studies For Cost Effective Digital Fabrication

One of the biggest issues in 3D printing is the material cost. Because of the high material costs in 3D printing, Wang et al. [24] came up with the idea that 3D printing skin-frame states of objects. They developed a method for automatically designing skin-frame states of objects. Their skin-frame structures resemble the objects which they derived and they are proper for 3D printing. Moreover, these structures decreases the material would be used during 3D printing. Lu et al. [25] designed an hollowing optimization method for generating honeycomb-like structures inside the objects in order to reduce material costs in 3D printing.

2.4 Studies For Rapid Fabrication

Another important issue in 3D printing is the fabrication time. Mueller et al. [1] complained about that 3D printing an object takes a long time and they proposed to

3D print wireframe previews instead of the objects. In other words, they replaced the objects surfaces with wireframe meshes and then 3D printed them. They modified a 3D printer in order to 3D print these wireframe meshes because layer-wise 3D printing was not proper to 3D print edges of the wireframe meshes. Their modified 3D printer extrudes material directly into 3D space and uses extra cooling mechanisms while 3D printing edges of wireframe meshes.

Peng et al. [2] advanced the 3D printing wireframe meshes and they provided on-the-fly 3D printing opportunity for designers. Their system integrated with CAD/CAM software which designers create digital objects. The digital objects created designers analyzed by their system and turned into wireframe meshes. These wireframe meshes 3D printed in parallel. Changes on the digital objects are immediately reflected on the 3D print.

Wu et al. [26] took wireframe 3D printing technology one step further with their 5DOF wireframe 3D printer. They developed a method for generating 3D printable (by using their 5DOF wireframe 3D printer) wireframe meshes of arbitrary meshes. Their 5DOF wireframe 3D printer can rotate the print during 3D printing operation if it is necessary. Thus, any edge of the wireframe mesh can be 3D printed.

2.5 Application Areas of Auxetics

The most important property of auxetics is they become thicker when they stretched. Under favor of this property, they have many application areas summarized by Liu and Hu [27]:

- Textiles; rivets, fastener, fishnet, rope, fastener
- Industry: packing materials
- Aerospace; wing panels, aircraft noses
- Protection; crash helmet, bulletproof armor, shin and knee pad, glove, porous barrier
- Biomedical; bandage reacting swells, muscle and ligament anchors

Additionally, auxetics have potential defense industry applications such as equipment can be used by soldiers for protection and logistics stated by Underhill [28] and explosion proof military vehicles indicated by Imbalzano et al [29].

New application areas have being discovered day to day.

CHAPTER 3

AUXETIC PATTERNS EXTRACTION METHODS

In this chapter, preferred auxetic pattern is introduced in Section 3.1. Additionally, reasons for choosing this auxetic pattern are also stated in the same section. Secondly, extraction method of this pattern from triangle meshes is explained in Section 3.2. Finally, extraction method of this pattern from quadrilateral, quad in short, meshes is explained in Section 3.3.

3.1 Auxetic Pattern

We preferred the re-entrant honeycomb structure as an auxetic pattern. Figure 3.1 shows this structure.

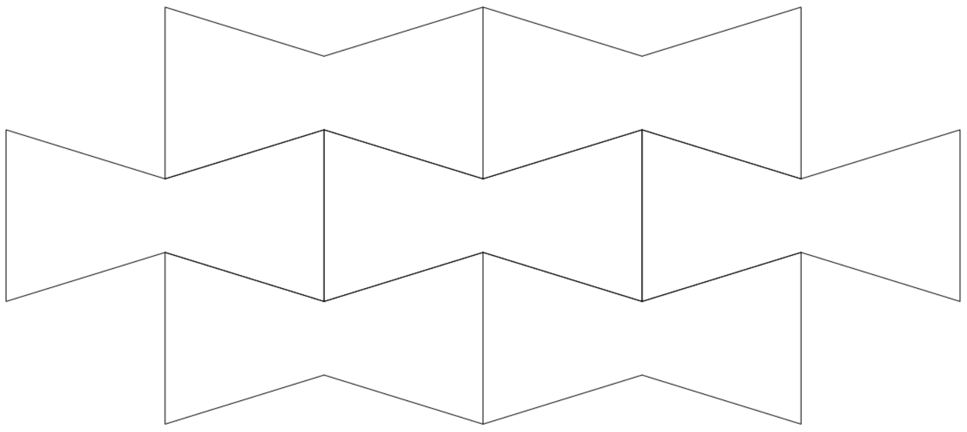


Figure 3.1: Re-entrant honeycomb structure

The re-entrant honeycomb pattern is consisting of hexagons. Structure of hexagon is proper to exhibit auxetic feature. When a tensile force applied to the hexagon, it expands in a parallel direction to the applied force's direction. The auxetic feature of a hexagon can be seen in Figure 3.2.

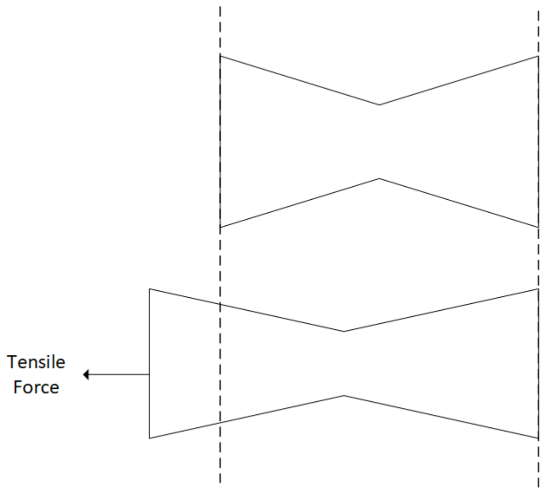
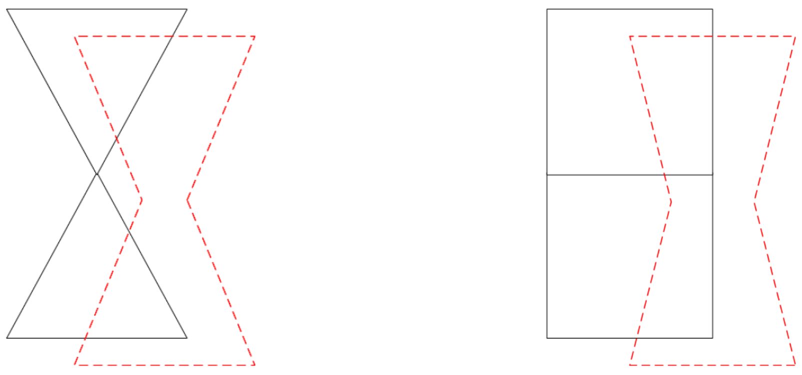


Figure 3.2: Hexagon and its expansion

The re-entrant honeycomb pattern is chosen between auxetic patterns because the hexagons creating this pattern can be derived easily from properly combined triangles and quads. The shape of the hexagon is very similar to two triangles intersecting at one point and their opposite edges of this point are parallel to each other shown in Figure 3.3a. Likewise, two quads having one common edge are also associated with a hexagon shape as shown in Figure 3.3b.



(a) Properly combined triangles (b) Properly combined quads

Figure 3.3: Resembling hexagon

In other words, meshes consisting of triangles or quads already host hexagons. Therefore, re-entrant honeycomb pattern which consists of hexagons can be extracted from triangle and quad meshes.

3.2 Extracting Auxetic Patterns From Triangle Meshes

We designed a novel approach to extract auxetic patterns from triangle meshes. Extraction of the auxetic patterns from given triangle mesh is shown in Algorithm 3.2.1. This algorithm searches and extracts honeycomb patterns on given triangle mesh starting from given triangle and return the resulted structure.

Hexagons creating the auxetic patterns are named as bow ties and the extracted auxetic pattern structure referred as bow tie mesh at the rest of this section.

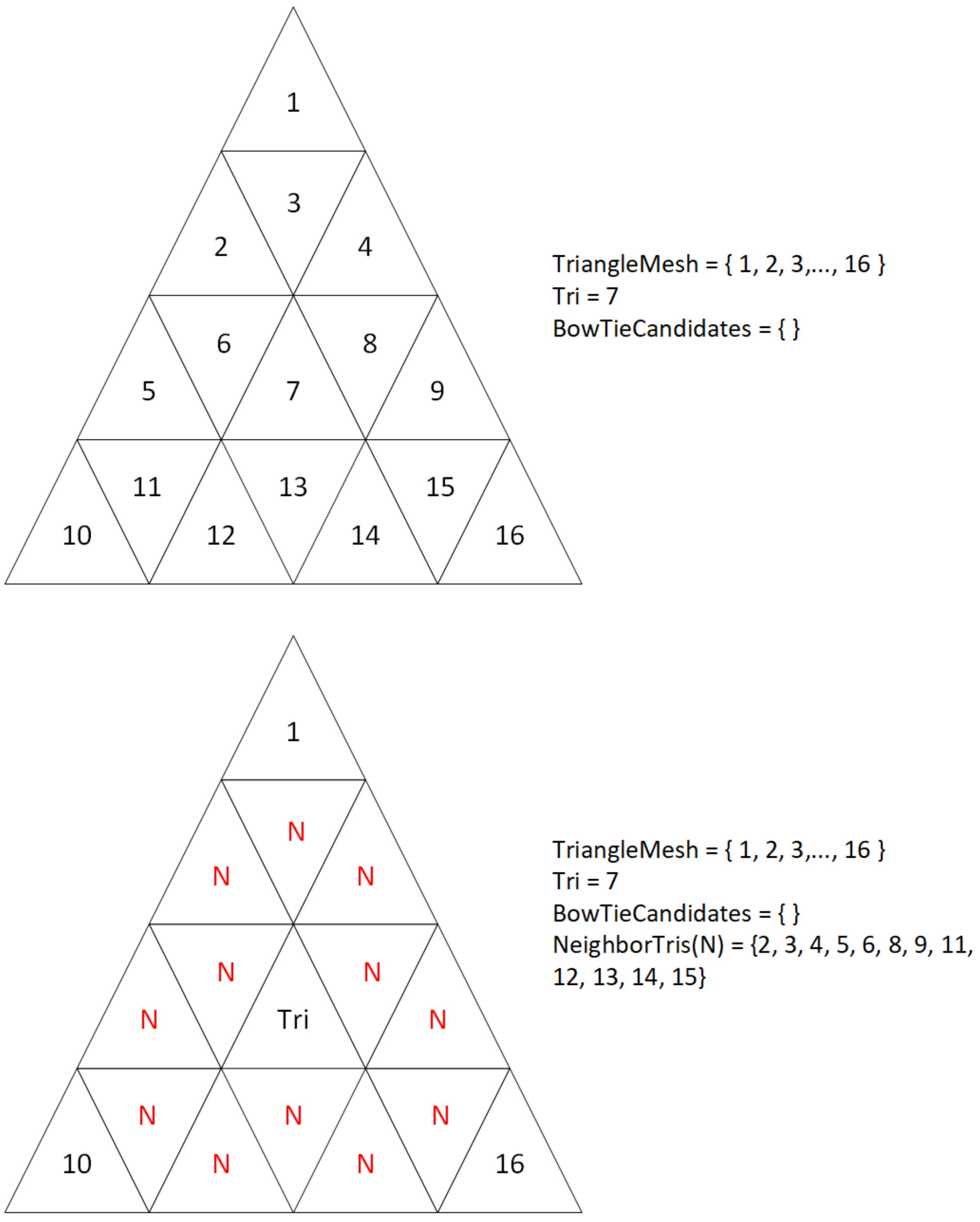
Algorithm 3.2.1 SEARCH(TriangleMesh, StartingTri)

```
1:  BowTieMesh = empty set of bow ties, their points and their edges
2:  BowTieCandidateTris = GETBOWTIECANDIDATETRIS(TriangleMesh, StartingTri)
3:  Write BowTieCandidateTris
4:  Read PairTri
5:  BowTieTriPairs = FINDBOWTIES(BowTieMesh, TriangleMesh, StartingTri, PairTri)
6:  for each triangle pair created bow tie $P \in$ BowTieTriPairs do
7:      ContiguousBowTieTriPair = FINDCONTIGUOUSBOWTIE(TriangleMesh, $P.First$, $P.Second$)

8:      if ContiguousBowTieTriPair $\neq null$ then
9:          NewBowTieTriPairs = FINDBOWTIES(BowTieMesh, TriangleMesh, ContiguousBowTieTriPair.First, ContiguousBowTieTriPair.Second)
10:         Add NewBowTieTriPairs to BowTieTriPairs
11:     end if
12: end for
13: return BowTieMesh
```

The SEARCH algorithm uses GETBOWTIECANDIDATETRIS algorithm in order to get bow tie candidates of a triangle. The triangle and its bow tie candidate create hexagon together. How to get bow tie candidates of a given triangle is shown in Algorithm 3.2.2. This algorithm gets the bow tie candidates of the triangle on given mesh and returns them. Execution of this algorithm is shown in Figure 3.4.

Algorithm 3.2.2 GETBOWTIECANDIDATETRIS(TriangleMesh, Tri)

```
1: BowTieCandidateTris = empty set of triangles
2: NeighborTris = triangles having at least one common point with Tri
3: EdgeNeighborTris = triangles having one common edge with Tri
4: UnsuitableTris = EdgeNeighborTris
5: for each triangle $NeighborTri \in$ NeighborTris do
6:     if $NeighborTri$ has also common edge with one of the triangles from EdgeNeighborTris then
7:         Add $NeighborTri$ to UnsuitableTris
8:     end if
9: end for
10: BowTieCandidateTris = NeighborTris - UnsuitableTris
11: return BowTieCandidateTris
```

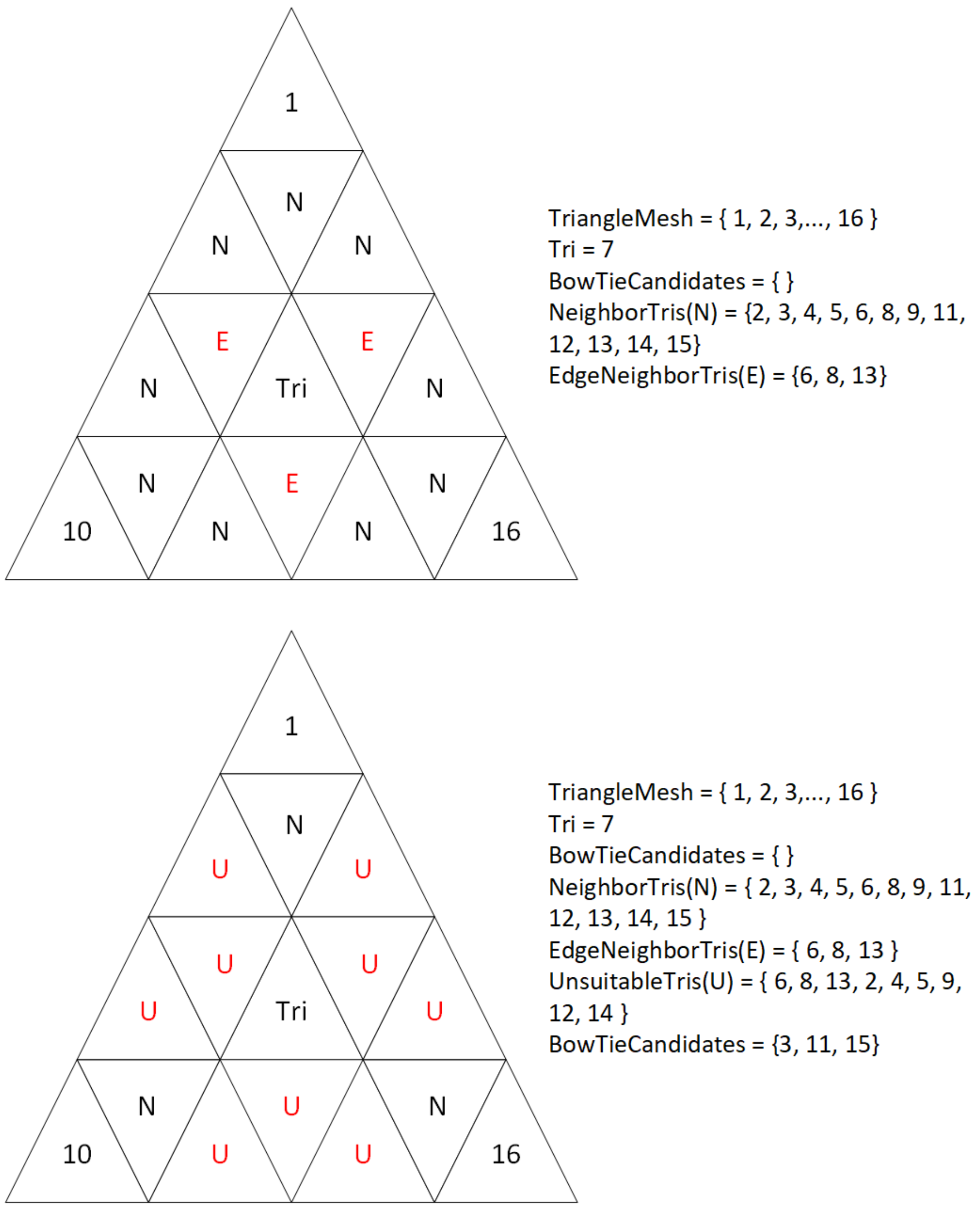


TriangleMesh = { 1, 2, 3,..., 16 }
Tri = 7
BowTieCandidates = { }

TriangleMesh = { 1, 2, 3,..., 16 }
Tri = 7
BowTieCandidates = { }
NeighborTris(N) = {2, 3, 4, 5, 6, 8, 9, 11, 12, 13, 14, 15}

(a)

Figure 3.4: Execution steps of the GETBOWTIECANDIDATETRIS algorithm


1
N
N
N
N
E
Tri
E
N
N
E
N
10
N
N
16
TriangleMesh = { 1, 2, 3,..., 16 }
Tri = 7
BowTieCandidates = { }
NeighborTris(N) = {2, 3, 4, 5, 6, 8, 9, 11, 12, 13, 14, 15}
EdgeNeighborTris(E) = {6, 8, 13}
1
N
U
U
U
U
U
Tri
U
N
U
N
10
U
U
16
TriangleMesh = { 1, 2, 3,..., 16 }
Tri = 7
BowTieCandidates = { }
NeighborTris(N) = { 2, 3, 4, 5, 6, 8, 9, 11, 12, 13, 14, 15 }
EdgeNeighborTris(E) = { 6, 8, 13 }
UnsuitableTris(U) = { 6, 8, 13, 2, 4, 5, 9, 12, 14 }
BowTieCandidates = {3, 11, 15}

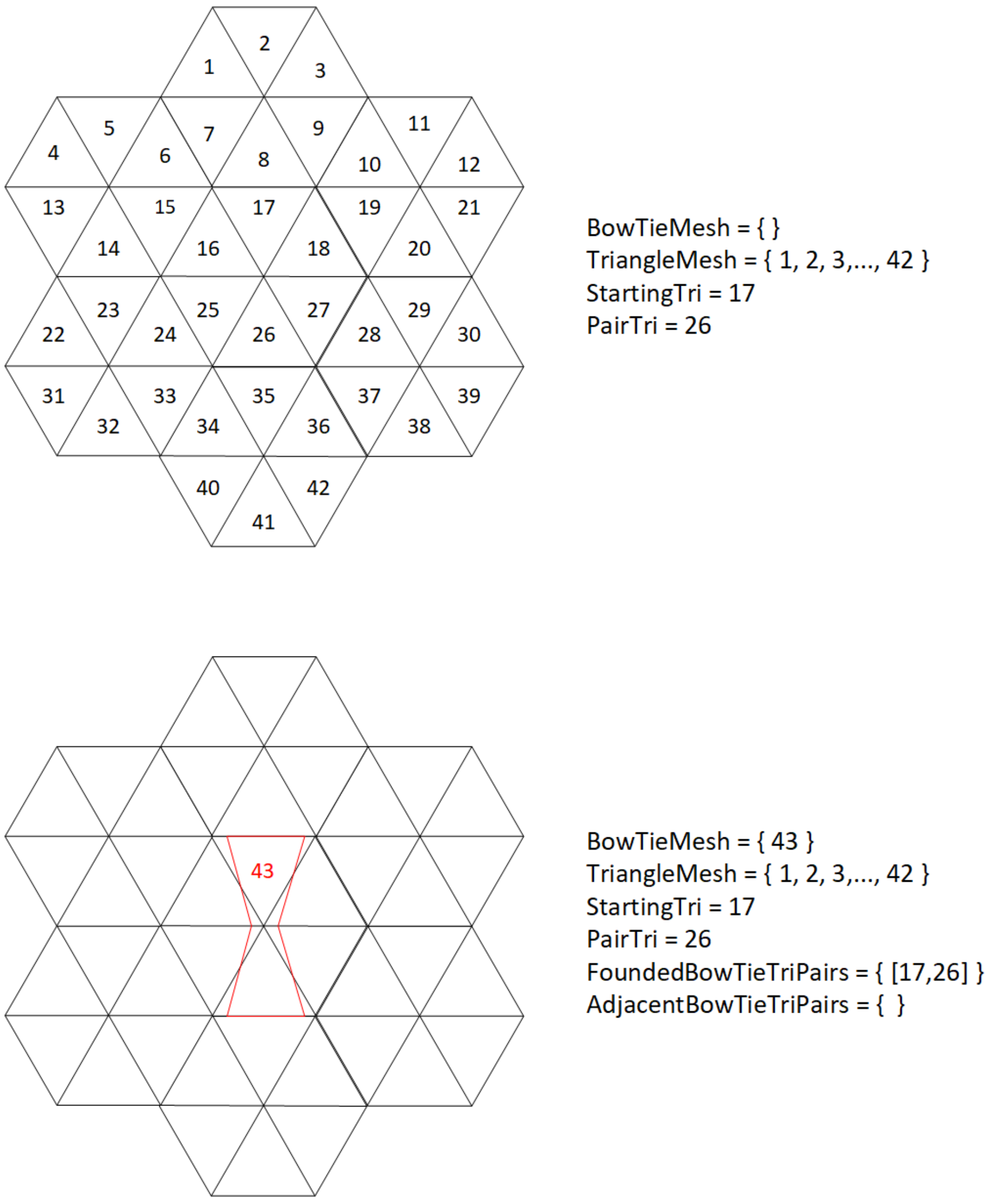
(b)

Figure 3.4: Execution steps of the GETBOWTIECANDIDATETRIS algorithm (cont.)

SEARCH algorithm also uses FINDBOWTIES algorithm to find bow ties and insert them into bow tie mesh structure. How to find, create and insert bow ties starting from a given triangle pair is shown in Algorithm 3.2.3. This algorithm gets the triangle pair, tries to create and insert a bow tie from this pair, if it succeeds then it tries to find out all bow ties can be created from neighbor triangle pairs of this pair recursively until it cannot find any bow tie. Execution of this algorithm is shown in Figure 3.5.

Algorithm 3.2.3 FINDBOWTIES(BowTieMesh, TriangleMesh, StartingTri, PairTri)

```
1: FoundedBowTieTriPairs = empty pair of triangles which created bow tie
2: INSERTBOWTIE(BowTieMesh, TriangleMesh, StartingTri, PairTri)
3: if bow tie was inserted from StartingTri and PairTri then
4:    Add [StartingTri, PairTri] to FoundedBowTieTriPairs
5:    AdjacentPairs = empty set of adjacent pairs of tris
6:    AdjacentPairs = GETADJACENTPAIRS(TriangleMesh, StartingTri, PairTri)
7:    while AdjacentPairs $\neq empty$ do
8:       NewAdjacentPairs = empty set of adjacent pairs of tris
9:       for each adjacent pair $P \in$ AdjacentPairs do
10:         INSERTBOWTIE(BowTieMesh, TriangleMesh, $P.First$, $P.Second$)
11:         if bow tie was inserted from $P.First$ and $P.Second$ then
12:            Add [$P.First$, $P.Second$] to FoundedBowTieTriPairs
13:            AdcajentPairsFounded = GETADJACENTPAIRS(TriangleMesh, $P.First$, $P.Second$)

14:            Add AdcajentPairsFounded to NewAdjacentPairs
15:         end if
16:      end for
17:      AdjacentPairs = NewAdjacentPairs
18:   end while
19:   ContiguousBowTieTriPair = FINDCONTIGUOUSBOWTIE(TriangleMesh, StartingTri, Pair-
      Tri)
20:   if ContiguousBowTieTriPair $\neq null$ then
21:      NewFoundedBowTieTriPairs = FINDBOWTIES(BowTieMesh, TriangleMesh, Contiguous-
         BowTieTriPair.First, ContiguousBowTieTriPair.Second)
22:      Add NewFoundedBowTieTriPairs to FoundedBowTieTriPairs
23:   end if
24:   if ContiguousBowTieTriPair $\neq null$ then
25:      NewFoundedBowTieTriPairs = FINDBOWTIES(BowTieMesh, TriangleMesh, Contiguous-
            BowTieTriPair.First, ContiguousBowTieTriPair.Second)
26:         Add NewFoundedBowTieTriPairs to FoundedBowTieTriPairs
27:     **end if**
28: **end if**
29: **return** FoundedBowTieTriPairs
```

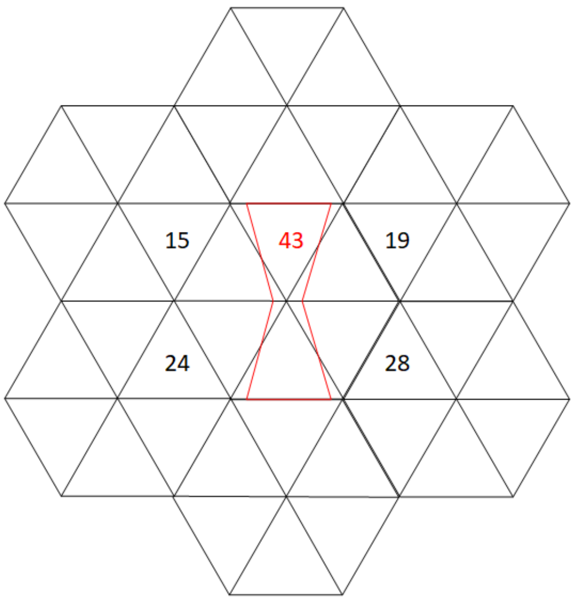


BowTieMesh = { }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26

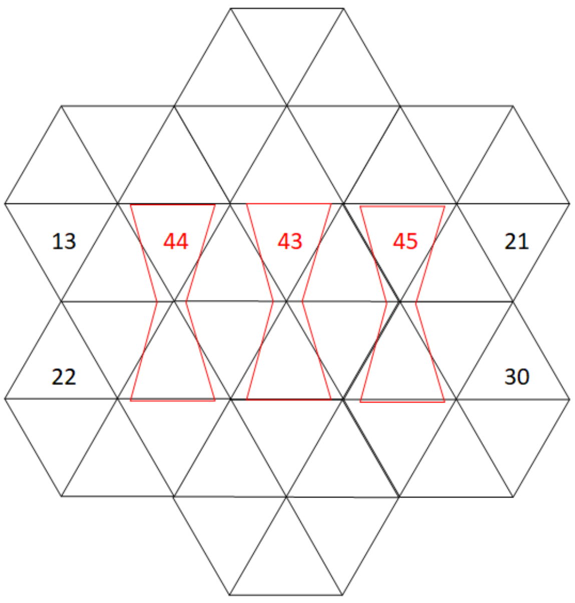
BowTieMesh = { 43 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26] }
AdjacentBowTieTriPairs = { }

(a)

Figure 3.5: Execution steps of the FINDBOWTIES algorithm



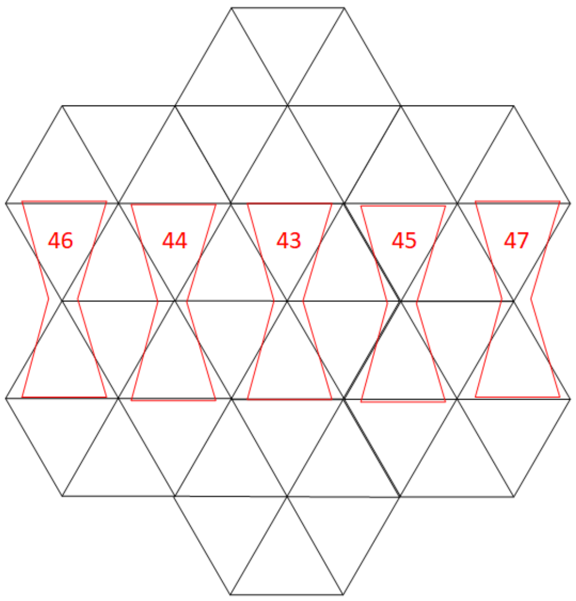
BowTieMesh = { 43 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26] }
AdjacentPairs = { [15,24], [19,28] }



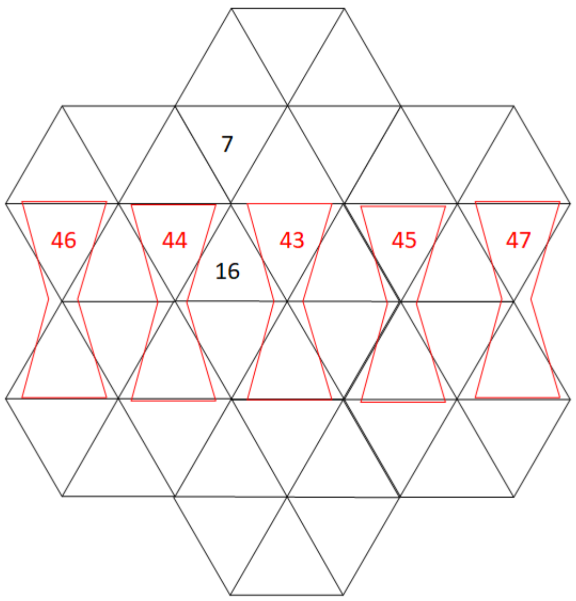
BowTieMesh = { 43, 44, 45 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28] }
AdjacenPairs = { [13,22], [21,30] }

(b)

Figure 3.5: Execution steps of the FINDBOWTIES algorithm (cont.)



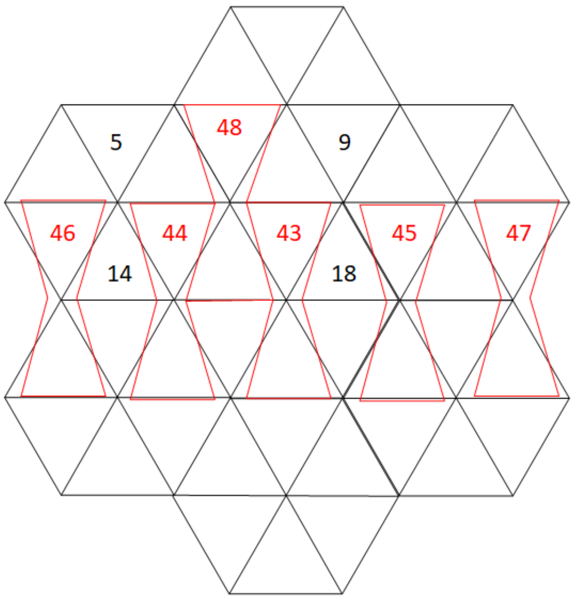
BowTieMesh = { 43, 44, 45, 46, 47 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30] }
AdjacenPairs = { }



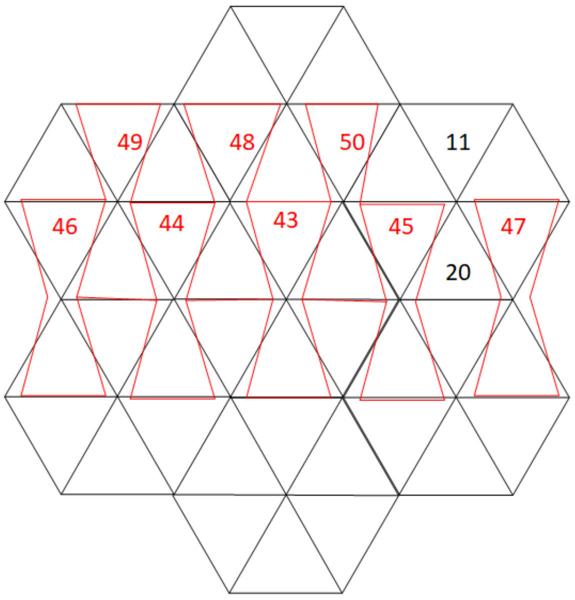
BowTieMesh = { 43, 44, 45, 46, 47 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30] }
AdjacenPairs = { }
ContiguousBowTieTriPair = [16,7]

(c)

Figure 3.5: Execution steps of the FINDBOWTIES algorithm (cont.)



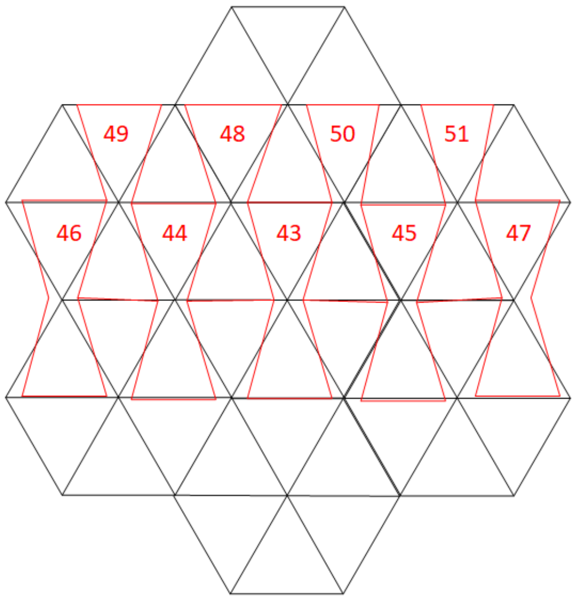
BowTieMesh = { 43, 44, 45, 46, 47, 48 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30], [16,7] }
AdjacenPairs = { [5,14], [9,18] }
ContiguousBowTieTriPair = [16,7]



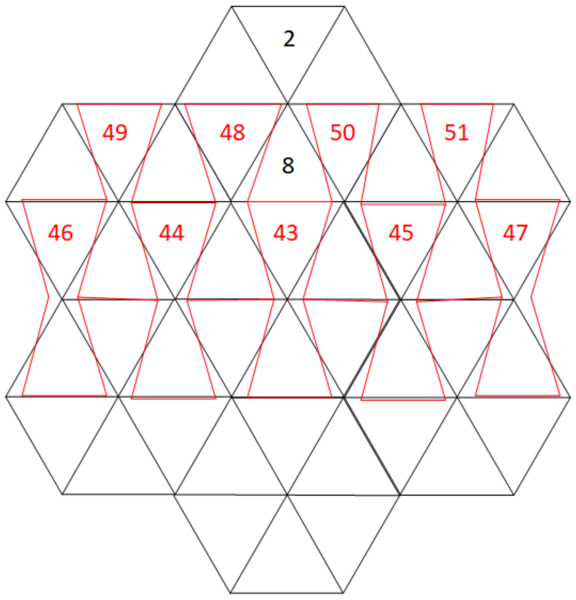
BowTieMesh = { 43, 44, 45, 46, 47, 48, 49, 50 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30], [16,7], [5,14], [9,18] }
AdjacenPairs = { [11,20] }
ContiguousBowTieTriPair = [16,7]

(d)

Figure 3.5: Execution steps of the FINDBOWTIES algorithm (cont.)



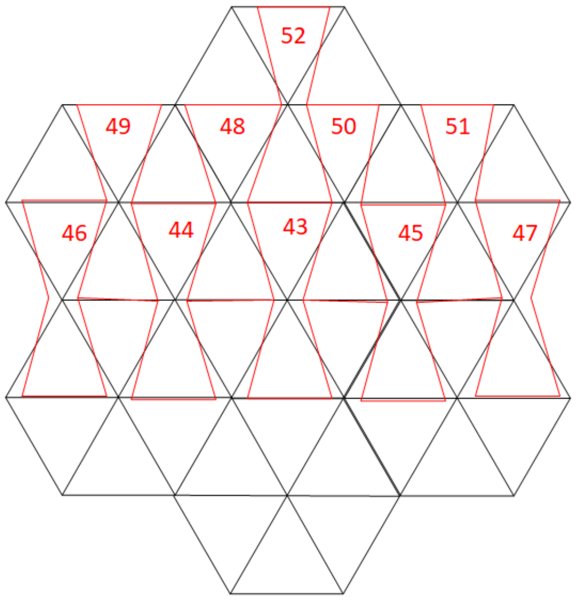
BowTieMesh = { 43, 44, 45, 46, 47, 48, 49, 50, 51 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30], [16,7], [5,14], [9,18], [11,20] }
AdjacenPairs = { }
ContiguousBowTieTriPair = [16,7]



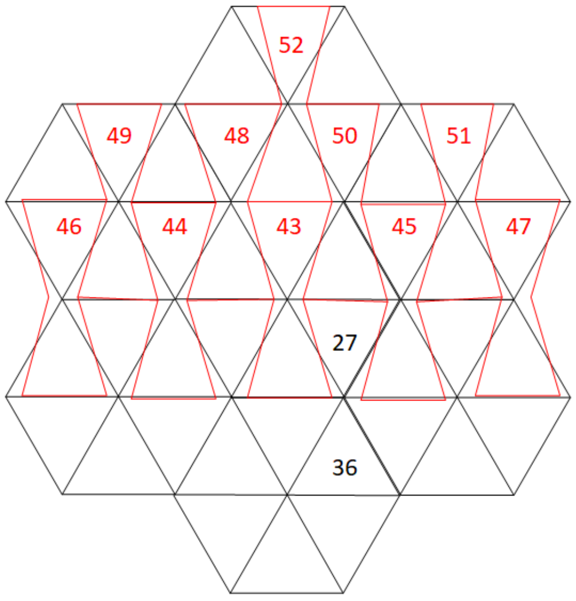
BowTieMesh = { 43, 44, 45, 46, 47, 48, 49, 50, 51 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30], [16,7], [5,14], [9,18], [11,20] }
AdjacenPairs = { }
ContiguousBowTieTriPair = [2,8]

(e)

Figure 3.5: Execution steps of the FINDBOWTIES algorithm (cont.)



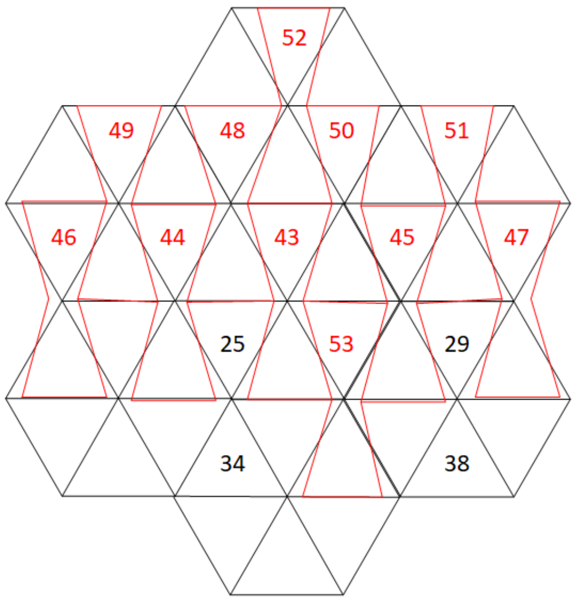
BowTieMesh = { 43, 44, 45, 46, 47, 48, 49, 50, 51, 52 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30], [16,7], [5,14], [9,18], [11,20], [2,8] }
AdjacenPairs = { }
ContiguousBowTieTriPair = NULL



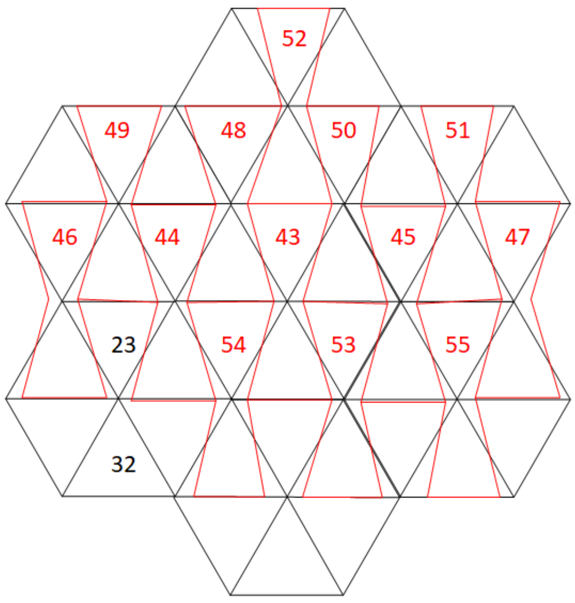
BowTieMesh = { 43, 44, 45, 46, 47, 48, 49, 50, 51, 52 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30], [16,7], [5,14], [9,18], [11,20], [2,8] }
AdjacenPairs = { }
ContiguousBowTieTriPair = [27,36]

(f)

Figure 3.5: Execution steps of the FINDBOWTIES algorithm (cont.)



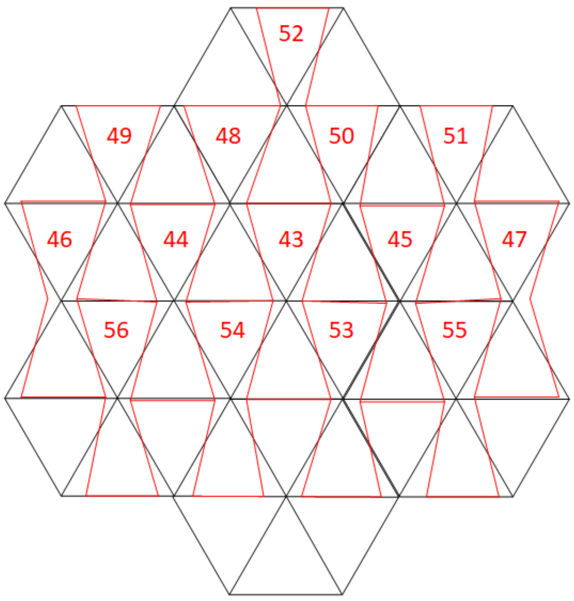
BowTieMesh = { 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30], [16,7], [5,14], [9,18], [11,20], [2,8], [27,36] }
AdjacenPairs = { [25,34], [29,38] }
ContiguousBowTieTriPair = [27,36]



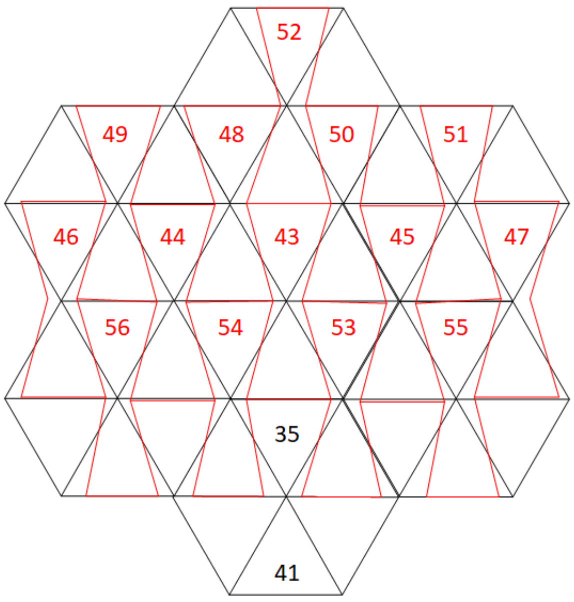
BowTieMesh = { 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30], [16,7], [5,14], [9,18], [11,20], [2,8], [27,36], [25,34], [29,38] }
AdjacenPairs = { [23,32] }
ContiguousBowTieTriPair = [27,36]

(g)

Figure 3.5: Execution steps of the FINDBOWTIES algorithm (cont.)



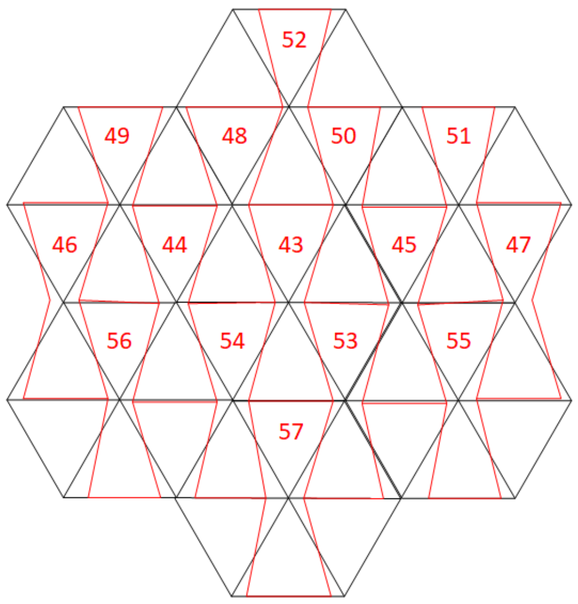
BowTieMesh = { 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30], [16,7], [5,14], [9,18], [11,20], [2,8], [27,36], [25,34], [29,38], [23,32] }
AdjacenPairs = { }
ContiguousBowTieTriPair = [35,41]



BowTieMesh = { 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30], [16,7], [5,14], [9,18], [11,20], [2,8], [27,36], [25,34], [29,38], [23,32] }
AdjacenPairs = { }
ContiguousBowTieTriPair = [27,36]

(h)

Figure 3.5: Execution steps of the FINDBOWTIES algorithm (cont.)



BowTieMesh = { 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57 }
TriangleMesh = { 1, 2, 3,..., 42 }
StartingTri = 17
PairTri = 26
FoundedBowTieTriPairs = { [17,26], [15,24], [19,28], [13,22], [21,30], [16,7], [5,14], [9,18], [11,20], [2,8], [27,36], [25,34], [29,38], [23,32], [35,41] }
AdjacenPairs = { }
ContiguousBowTieTriPair = NULL

(i)

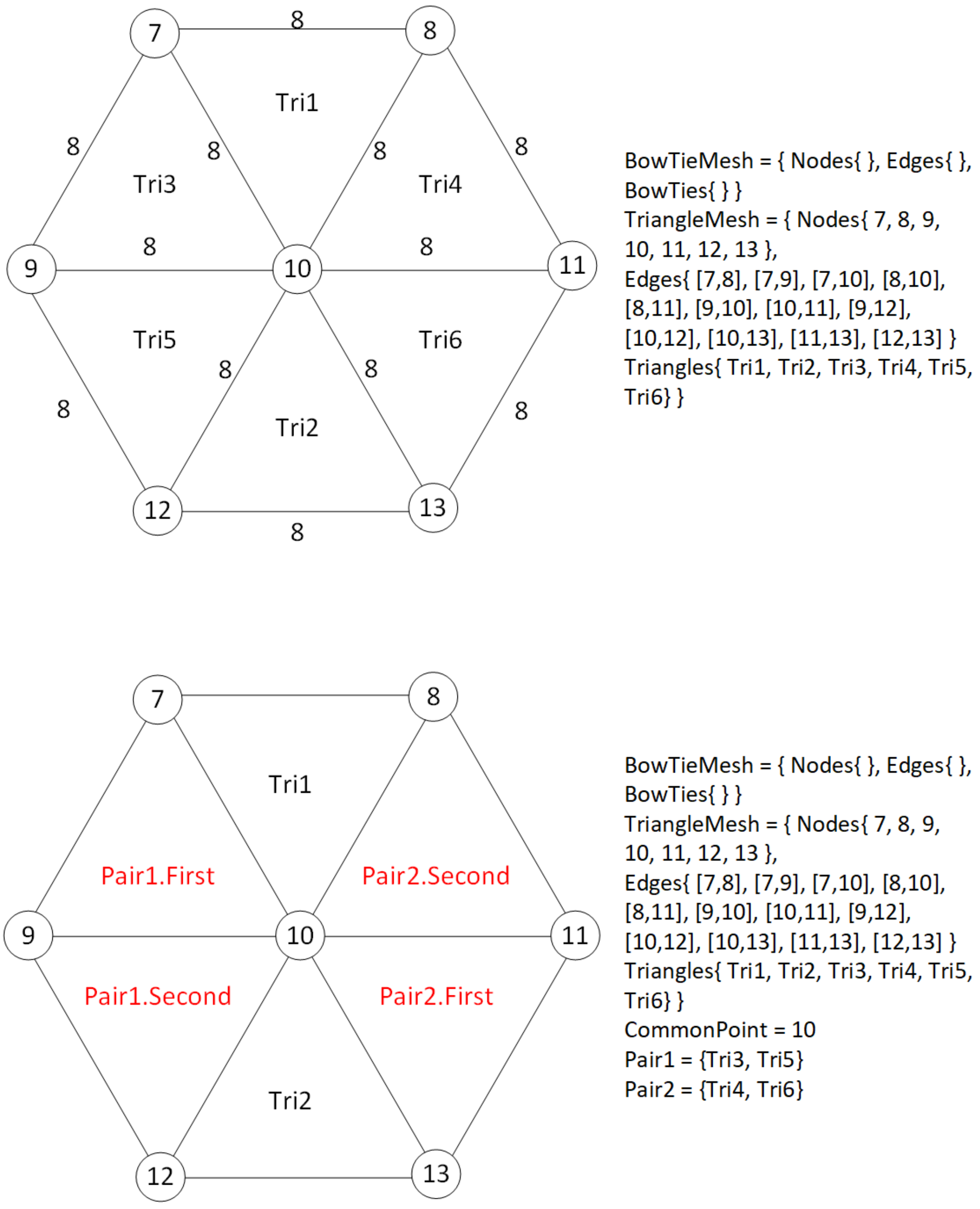
Figure 3.5: Execution steps of the FINDBOWTIES algorithm (cont.)

FINDBOWTIES algorithm employs INSERTBOWTIE algorithm to create and insert bow tie. Bow tie creation from triangle pair is shown in Algorithm 3.2.4. This algorithm calculates coordinates of points creating the bow tie and orders these points to create related a bow tie. Then, inserts the points and edges consisting of these points into bow tie mesh. Execution of this algorithm is shown in Figure 3.6.

Algorithm 3.2.4 INSERTBOWTIE(BowTieMesh, TriangleMesh, Tri1, Tri2)

```
1:  CommonPoint = common point of Tri1 and Tri2
2:  if CommonPoint exists then
3:      Pair1 = CommonPoint's triangles which have common edge
4:      Pair2 = CommonPoint's triangles which have common edge
5:      if Pair1 and Pair2 exists then
6:          FacingEdge = Tri1's facing edge of CommonPoint
7:          p1 = FacingEdge.First + (length of FacingEdge) * 1/8
8:          p2 = FacingEdge.Second + (length of FacingEdge) * 1/8
9:          CommonEdge = Pair1's triangles common edge
10:         p3 = CommonPoint + (length of CommonEdge) * 1/8
11:         CommonEdge = Pair2's triangles common edge
12:         p4 = CommonPoint + (length of CommonEdge) * 1/8
13:         FacingEdge = Tri2's facing edge of CommonPoint
14:         p5 = FacingEdge.First + (length of FacingEdge) * 1/8
15:         p6 = FacingEdge.Second + (length of FacingEdge) * 1/8
16:         Add points p1, p2, p3, p4, p5, p6 to BowTieMesh if they are not added before
17:         BowTie = empty set of ordered points creating bow tie
18:         Add point p1 to BowTie
19:         Add point p2 to BowTie
20:         if p3 is closer than p4 to p2 then
21:             Add point p3 to BowTie
22:             if p5 is closer than p6 to p3 then
23:                 Add point p5 to BowTie
24:                 Add point p6 to BowTie
25:             else
26:                 Add point p6 to BowTie
27:                 Add point p5 to BowTie
28:             end if
29:             Add point p4 to BowTie
```

```
30:         else
31:             Add point p4 to BowTie
32:             if p5 is closer than p6 to p3 then
33:                 Add point p5 to BowTie
34:                 Add point p6 to BowTie
35:             else
36:                 Add point p6 to BowTie
37:                 Add point p5 to BowTie
38:             end if
39:             Add point p3 to BowTie
40:         end if
41:         if Tri1's normal vector and BowTie's normal vector are in the opposite directions then
42:             Reverse points of BowTie
43:         end if
44:         Add BowTie to BowTieMesh
45:         Tri1.IsMatched = $true$
46:         Tri2.IsMatched = $true$
47:     end if
48: end if
```

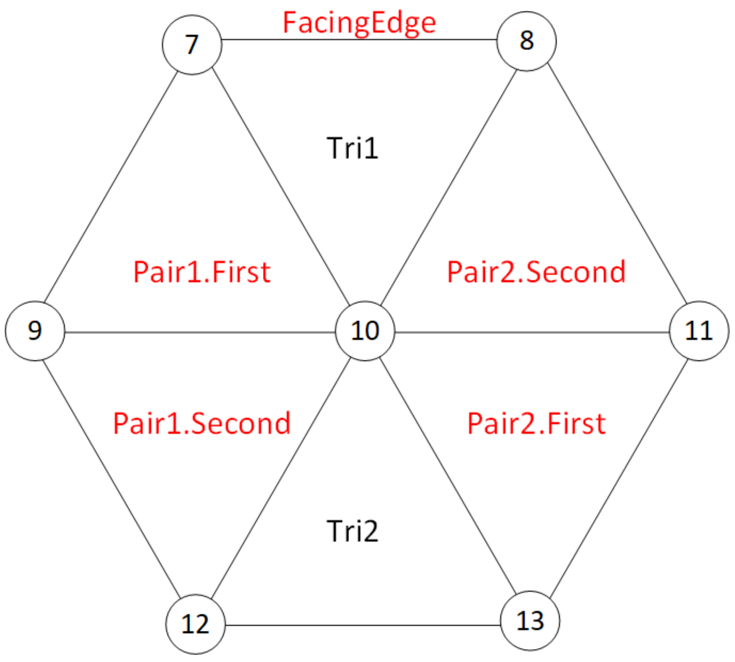


BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 }, Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] } Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }

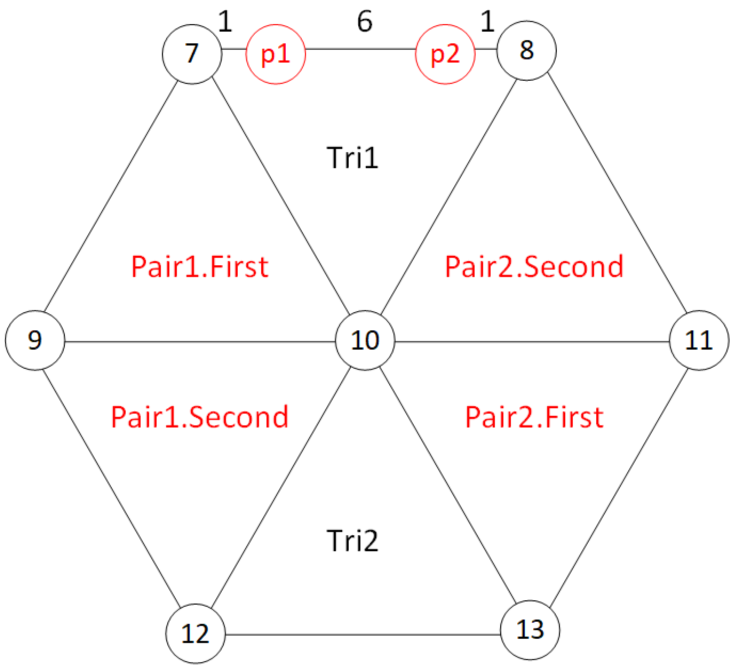
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 }, Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] } Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
CommonPoint = 10
Pair1 = {Tri3, Tri5}
Pair2 = {Tri4, Tri6}

(a)

Figure 3.6: Execution steps of the INSERTBOWTIE algorithm



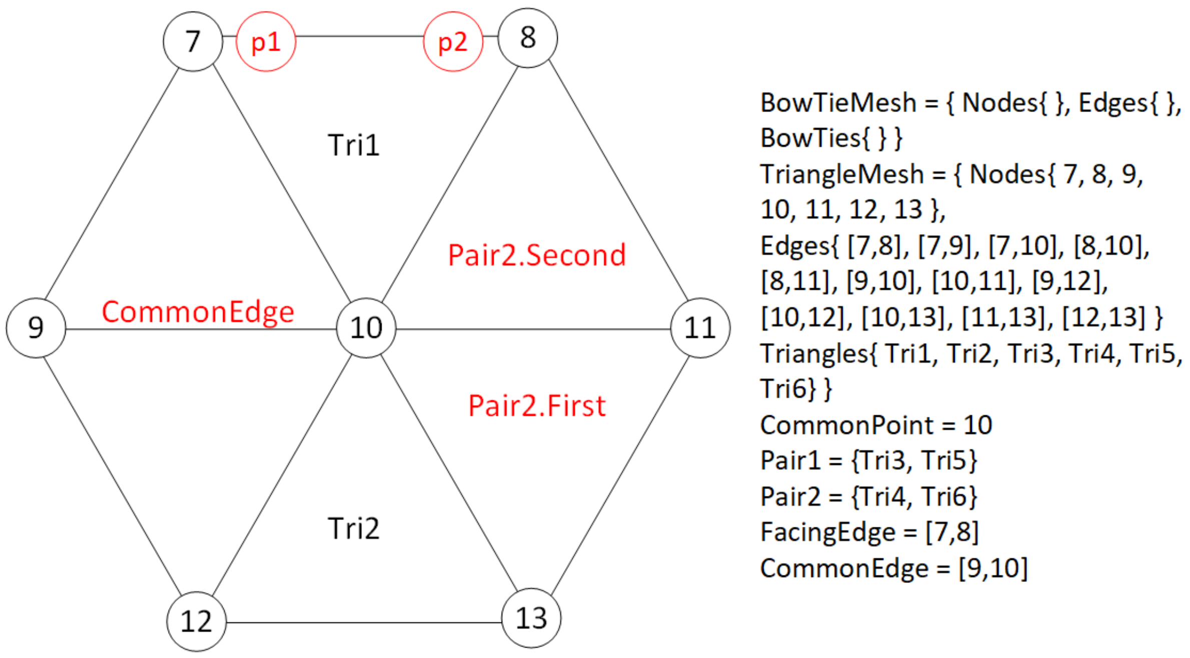
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 }, Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] } Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
CommonPoint = 10
Pair1 = {Tri3, Tri5}
Pair2 = {Tri4, Tri6}
FacingEdge = [7,8]



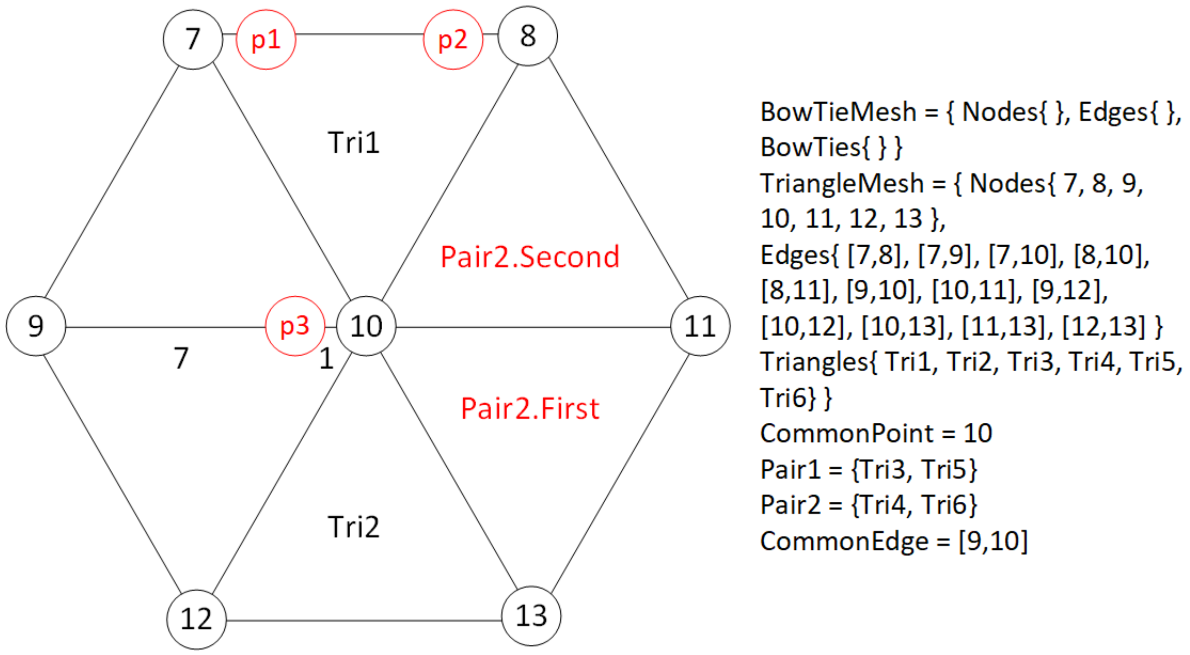
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 }, Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] } Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
CommonPoint = 10
Pair1 = {Tri3, Tri5}
Pair2 = {Tri4, Tri6}
FacingEdge = [7,8]

(b)

Figure 3.6: Execution steps of the INSERTBOWTIE algorithm (cont.)



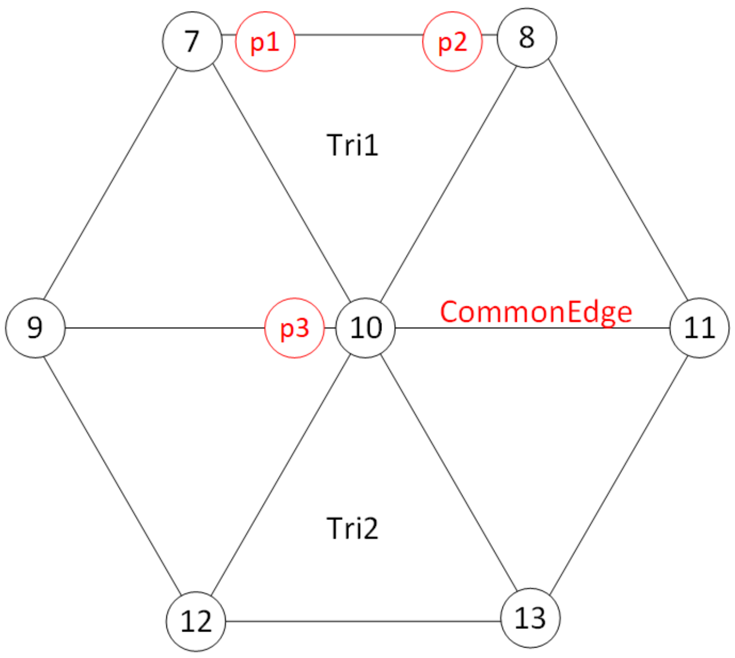
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 },
Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] }
Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
CommonPoint = 10
Pair1 = {Tri3, Tri5}
Pair2 = {Tri4, Tri6}
FacingEdge = [7,8]
CommonEdge = [9,10]



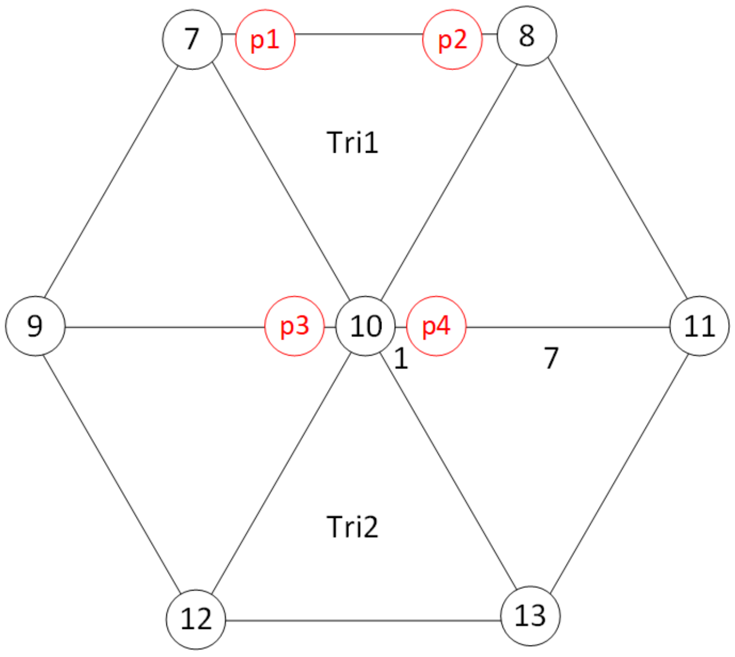
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 },
Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] }
Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
CommonPoint = 10
Pair1 = {Tri3, Tri5}
Pair2 = {Tri4, Tri6}
CommonEdge = [9,10]

(c)

Figure 3.6: Execution steps of the INSERTBOWTIE algorithm (cont.)



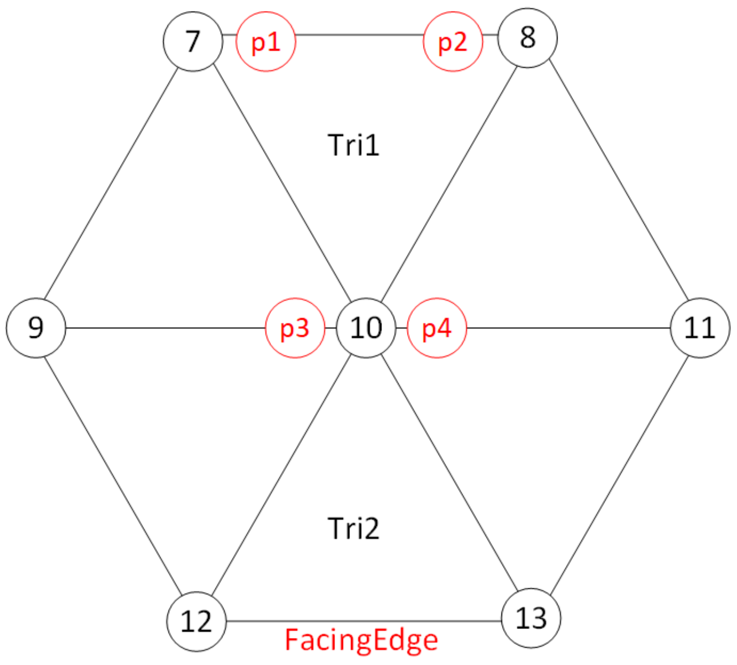
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 }, Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] } Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
CommonPoint = 10
Pair1 = {Tri3, Tri5}
Pair2 = {Tri4, Tri6}
CommonEdge = [10,11]



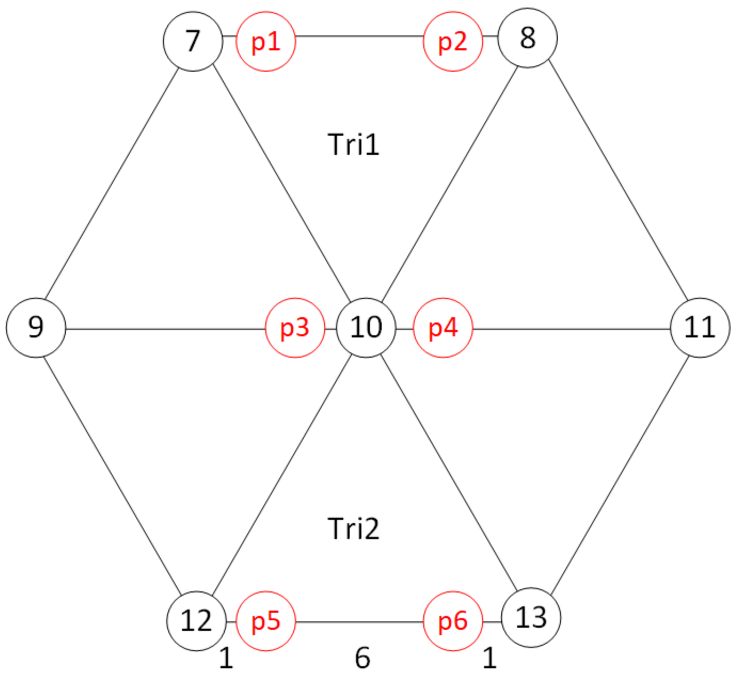
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 }, Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] } Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
CommonPoint = 10
Pair1 = {Tri3, Tri5}
Pair2 = {Tri4, Tri6}
CommonEdge = [10,11]

(d)

Figure 3.6: Execution steps of the INSERTBOWTIE algorithm (cont.)



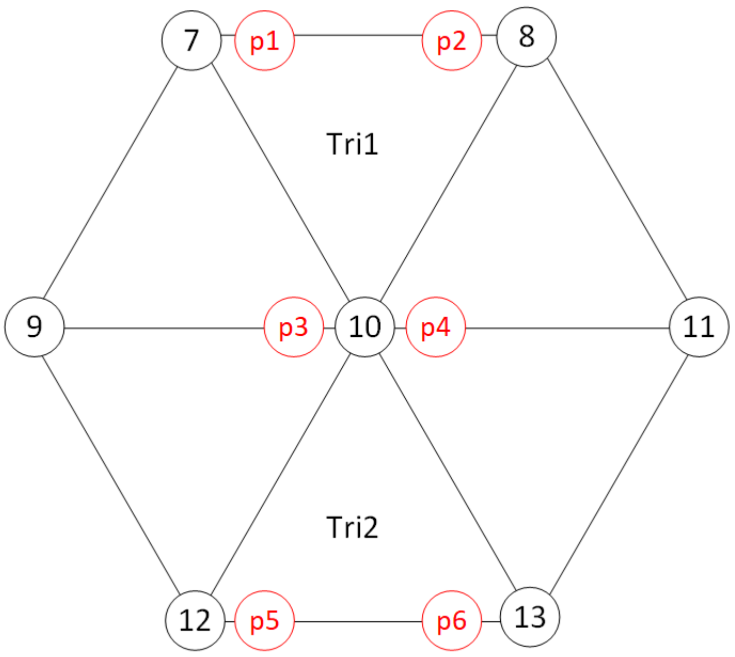
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 },
Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] }
Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
CommonPoint = 10
Pair1 = {Tri3, Tri5}
Pair2 = {Tri4, Tri6}
FacingEdge = [12,13]



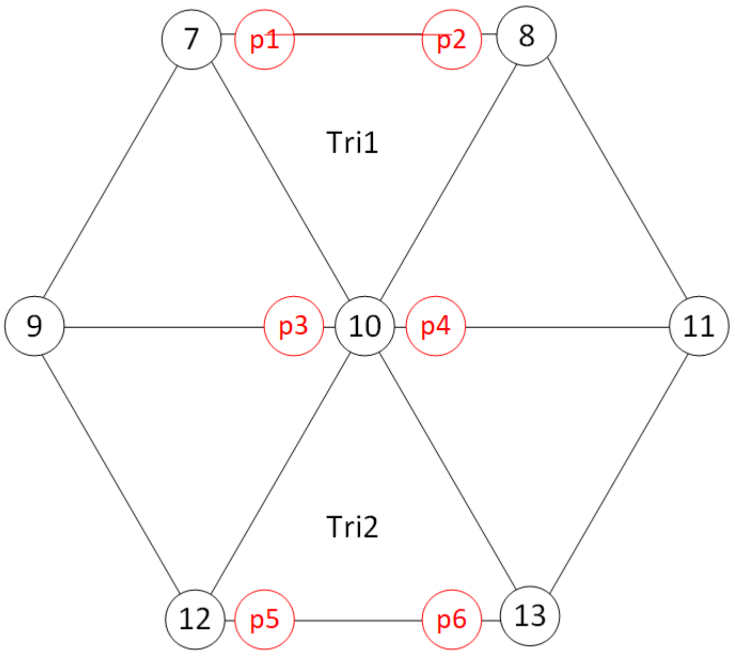
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 },
Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] }
Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
CommonPoint = 10
Pair1 = {Tri3, Tri5}
Pair2 = {Tri4, Tri6}
FacingEdge = [12,13]

(e)

Figure 3.6: Execution steps of the INSERTBOWTIE algorithm (cont.)



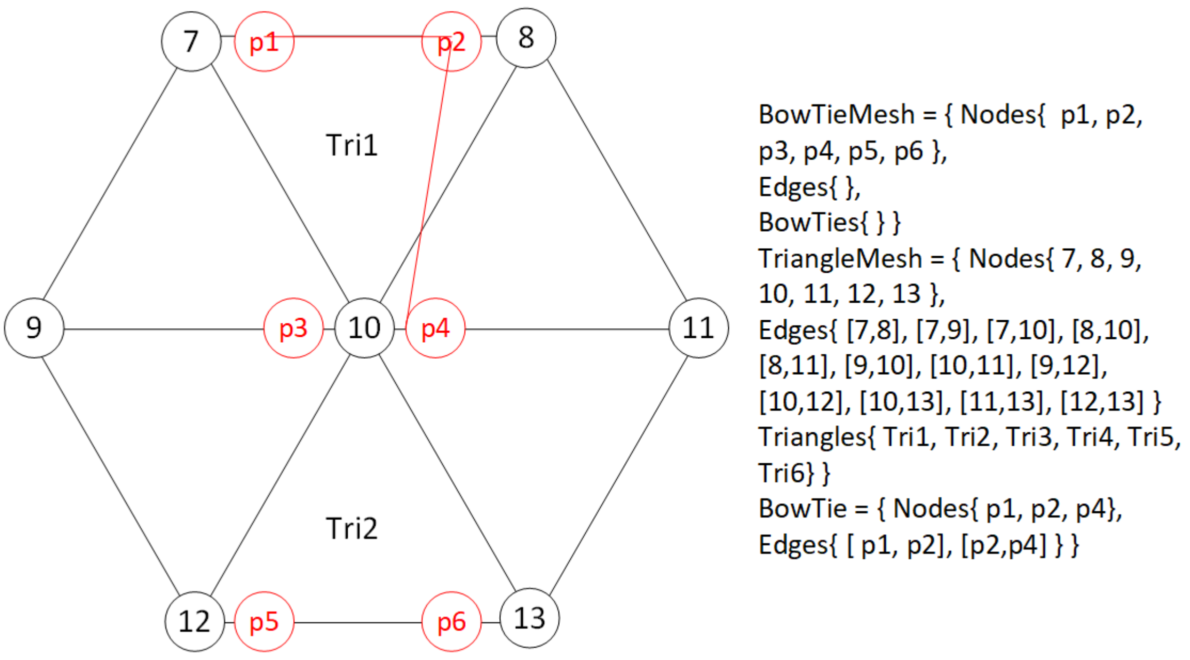
BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6 }, Edges{ }, BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 }, Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] } Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
BowTie = { Nodes{ }, Edges{ } }



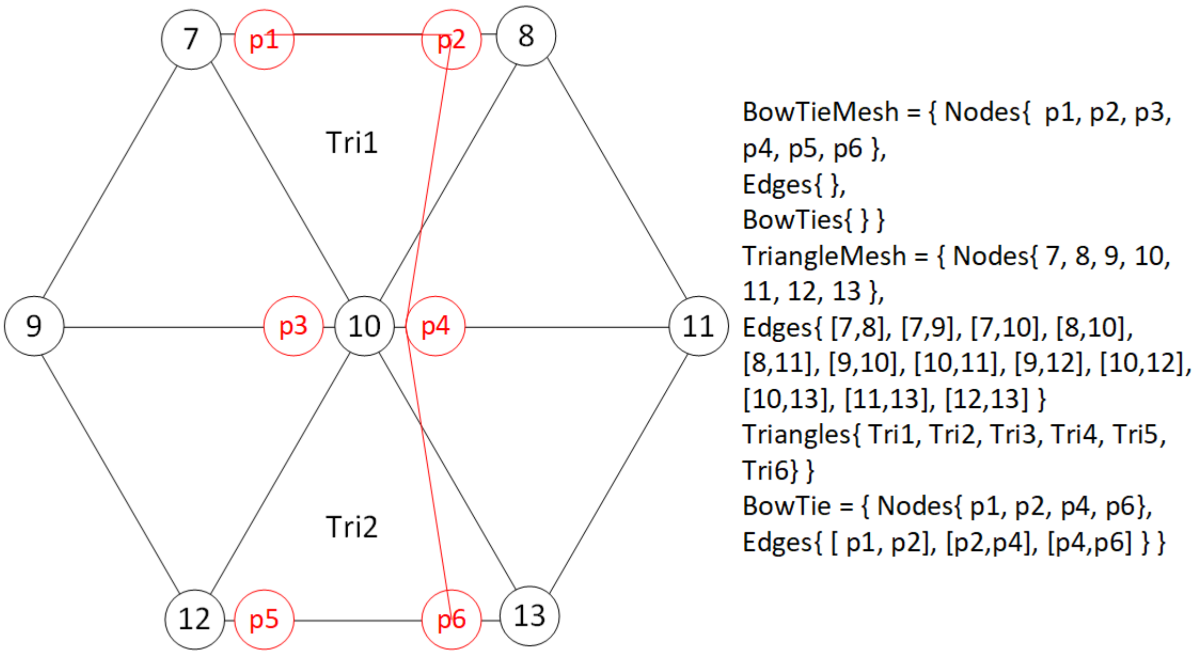
BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6 }, Edges{ }, BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 }, Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] } Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
BowTie = { Nodes{ p1, p2}, Edges{ [p1, p2] } }

(f)

Figure 3.6: Execution steps of the INSERTBOWTIE algorithm (cont.)



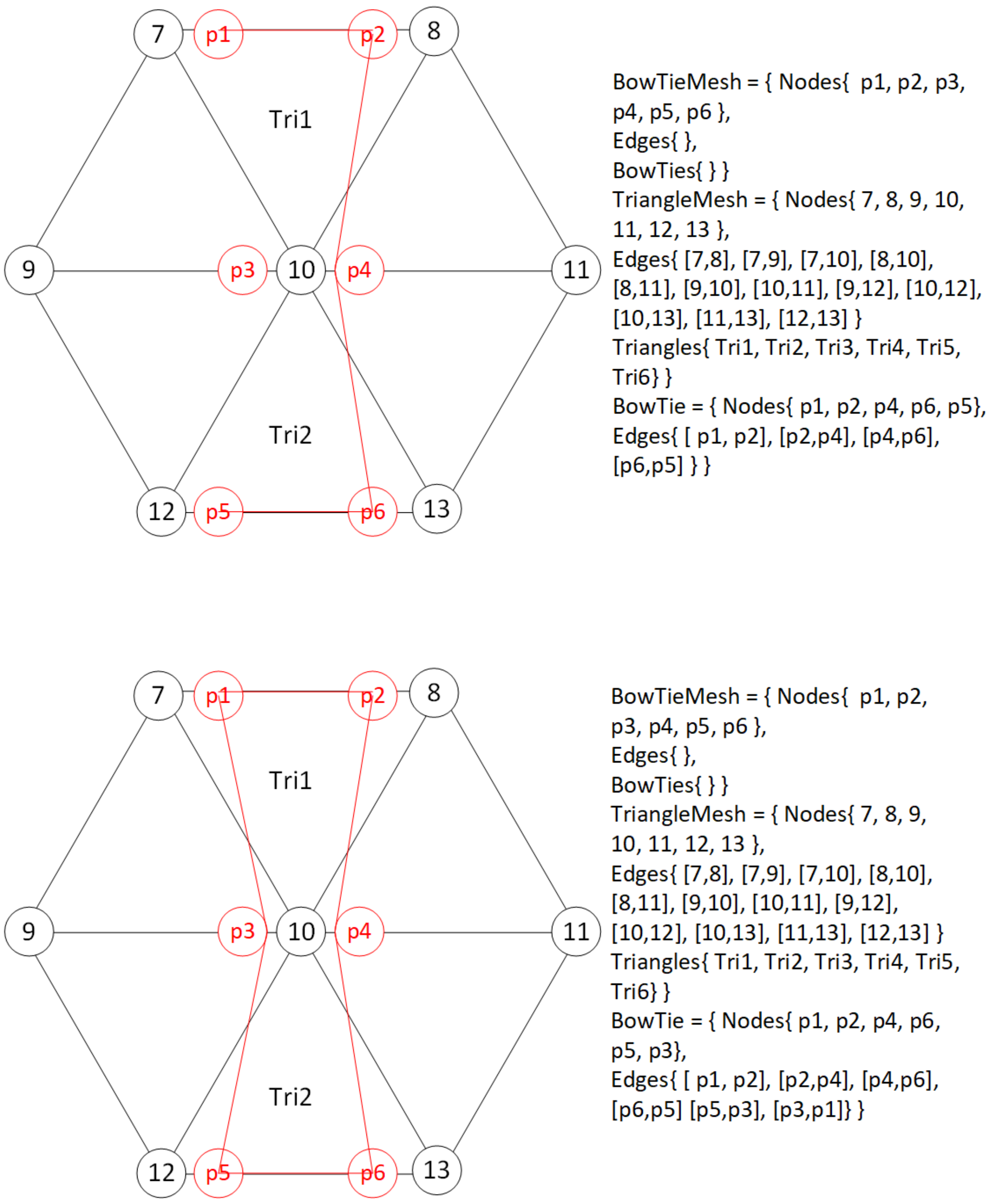
BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6 },
Edges{ },
BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 },
Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] }
Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
BowTie = { Nodes{ p1, p2, p4},
Edges{ [p1, p2], [p2,p4] } }



BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6 },
Edges{ },
BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 },
Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] }
Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
BowTie = { Nodes{ p1, p2, p4, p6},
Edges{ [p1, p2], [p2,p4], [p4,p6] } }

(g)

Figure 3.6: Execution steps of the INSERTBOWTIE algorithm (cont.)

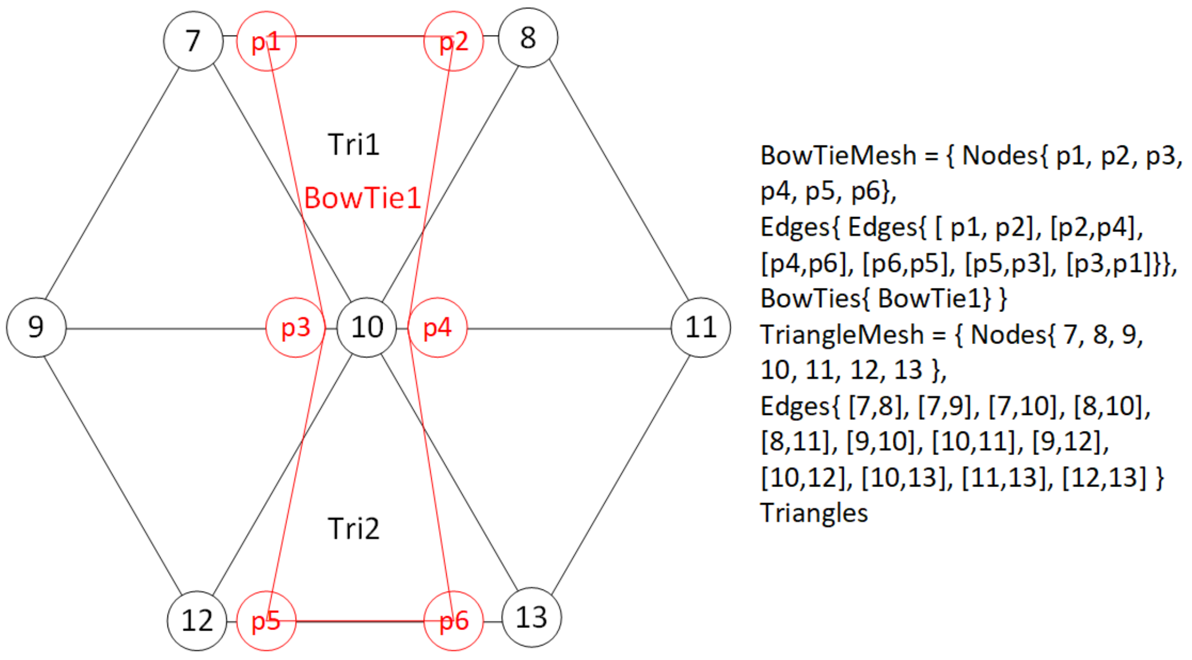


BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6 },
Edges{ },
BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 },
Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] }
Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
BowTie = { Nodes{ p1, p2, p4, p6, p5},
Edges{ [p1, p2], [p2,p4], [p4,p6], [p6,p5] } }

BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6 },
Edges{ },
BowTies{ } }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 },
Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] }
Triangles{ Tri1, Tri2, Tri3, Tri4, Tri5, Tri6} }
BowTie = { Nodes{ p1, p2, p4, p6, p5, p3},
Edges{ [p1, p2], [p2,p4], [p4,p6], [p6,p5] [p5,p3], [p3,p1]} }

(h)

Figure 3.6: Execution steps of the INSERTBOWTIE algorithm (cont.)



BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6},
Edges{ Edges{ [p1, p2], [p2,p4], [p4,p6], [p6,p5], [p5,p3], [p3,p1]}},
BowTies{ BowTie1} }
TriangleMesh = { Nodes{ 7, 8, 9, 10, 11, 12, 13 },
Edges{ [7,8], [7,9], [7,10], [8,10], [8,11], [9,10], [10,11], [9,12], [10,12], [10,13], [11,13], [12,13] }
Triangles

(i)

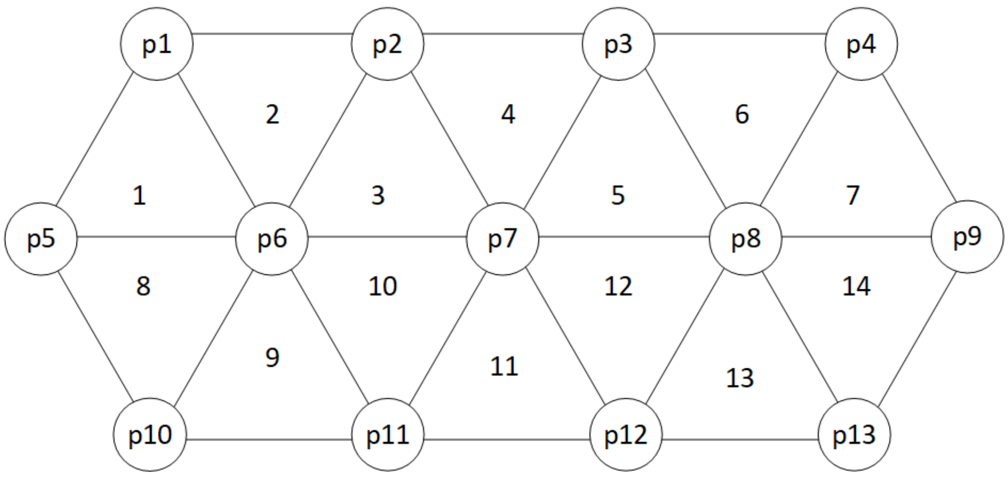
Figure 3.6: Execution steps of the INSERTBOWTIE algorithm (cont.)

Additionally, FINDBOWTIES algorithm uses GETADJACENTPAIRS algorithm for finding adjacent triangle pairs to triangle pairs created bow ties. Finding procedure is shown in Algorithm 3.2.5. This algorithm checks neighbor triangles of the given triangle pair and couples proper ones which can create new bow ties. Execution of this algorithm is shown in Figure 3.7.

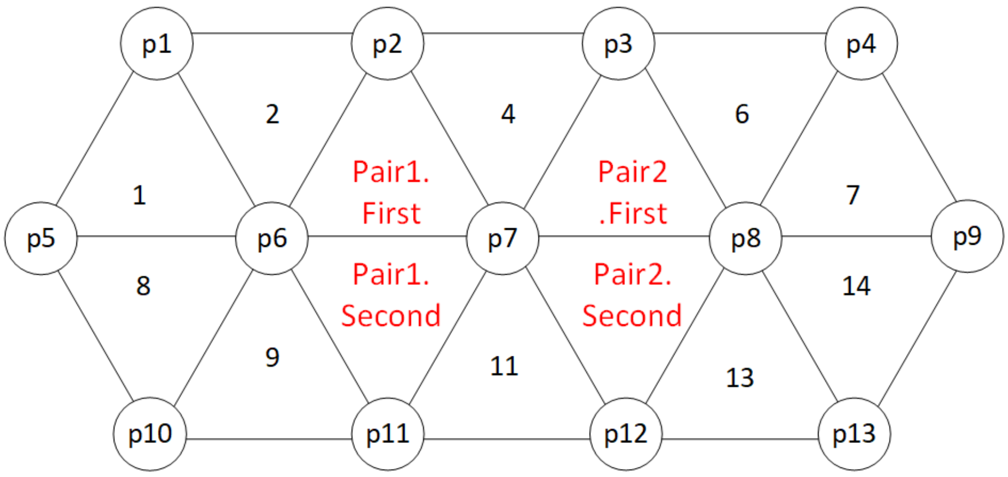
Algorithm 3.2.5 GETADJACENTPAIRS(TriangleMesh, Tri1, Tri2)

```
1: Pairs = empty set of pairs of triangles
2: CommonPoint = common point of Tri1 and Tri2
3: if CommonPoint exists then
4:     Pair1 = CommonPoint's triangles which have common edge
5:     Pair2 = CommonPoint's triangles which have common edge
6:     if Pair1 exists then
7:         CommonEdge = Pair1's triangles' common edge
8:         OtherPoint = CommonEdge's other point than CommonPoint
9:         Triangles = triangles containing OtherPoint
10:        First = $null$
11:        Second = $null$
12:        for each triangle $Tri \in$ Triangles do
13:            if $Tri \neq$ Pair1.First $and$ $Tri \neq$ Pair1.Second $and$ $Tri$ has common edge with Pair1.First
               or Pair1.Second then
14:                if First = $null$ then
15:                    First = $Tri$
16:                else
17:                    Second = $Tri$
18:                end if
19:            end if
20:            if First $\neq null$ $and$ !First.IsMatched $and$ Second $\neq null$ $and$ !Second.IsMatched then
21:                Add [First, Second] to Pairs
22:                break
23:            end if
24:        end for
25:    end if
26:    if Pair2 exists then
27:        CommonEdge = Pair2's triangles' common edge
28:        OtherPoint = CommonEdge's other point than CommonPoint
```

```
29:         Triangles = triangles containing OtherPoint
30:         First = $null$
31:         Second = $null$
32:         **for each** triangle $Tri \in$ Triangles **do**
33:             **if** $Tri \neq$ Pair2.First $and$ $Tri \neq$ Pair2.Second $and$ $Tri$ has common edge with Pair2.First
                or Pair2.Second **then**
34:                 **if** First = $null$ **then**
35:                     First = $Tri$
36:                 **else**
37:                     Second = $Tri$
38:                 **end if**
39:             **end if**
40:             **if** First $\neq$ $null$ $and$ !First.IsMatched $and$ Second $\neq$ $null$ $and$ !Second.IsMatched **then**
41:                 Add [First, Second] to Pairs
42:                 **break**
43:             **end if**
44:         **end for**
45:     **end if**
46: **end if**
47: **return** Pairs
```



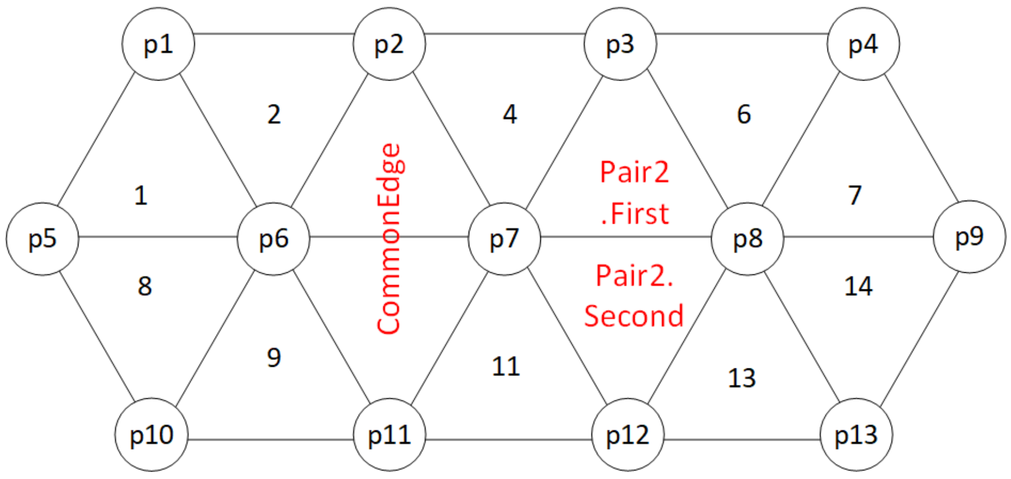
TriangleMesh = { Nodes{ p1,..., p16 }, Edges{ [p1,p2],..., [p12,p13] }, Tris{ 1,..., 14 } }
Pairs = { }
Tri1 = 4 Tri2 = 11



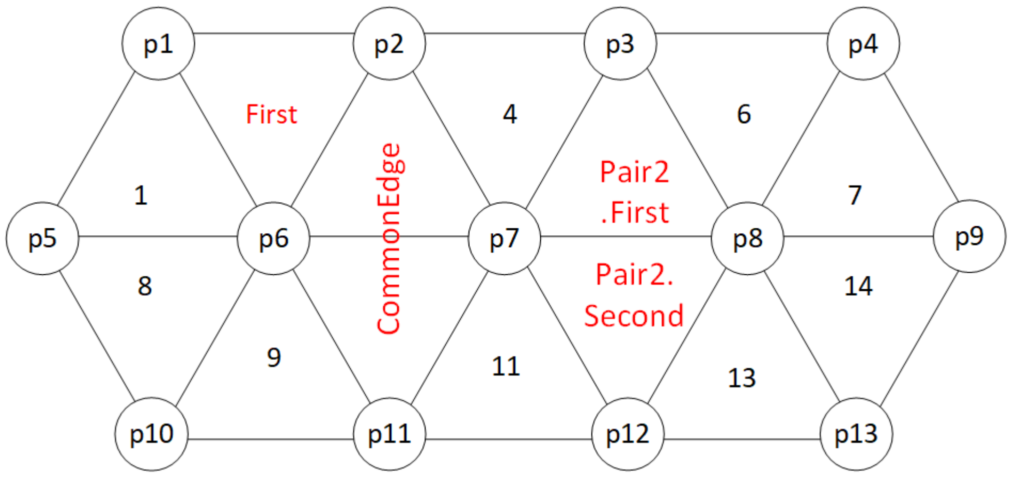
TriangleMesh = { Nodes{ p1,..., p16 }, Edges{ [p1,p2],..., [p12,p13] }, Tris{ 1,..., 14 } }
Pairs = { }
Tri1 = 4 Tri2 = 11
CommonPoint = p7
Pair1 = {3,10} Pair2 = {5,12}

(a)

Figure 3.7: Execution steps of the GETADJACENTPAIRS algorithm



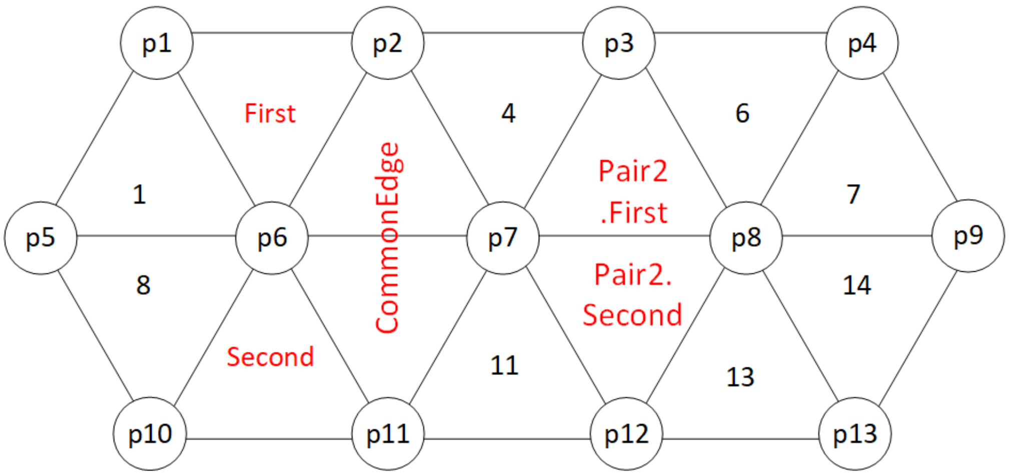
TriangleMesh = { Nodes{ p1,..., p16 }, Edges{ [p1,p2],..., [p12,p13] }, Tris{ 1,..., 14 } }
Pairs = { }
Tri1 = 4 Tri2 = 11
CommonPoint = p7
Pair1 = {3,10} Pair2 = {5,12}
CommonEdge = [p6,p7]
OtherPoint = p6
Triangles = {1, 2, 3, 8, 9, 10}



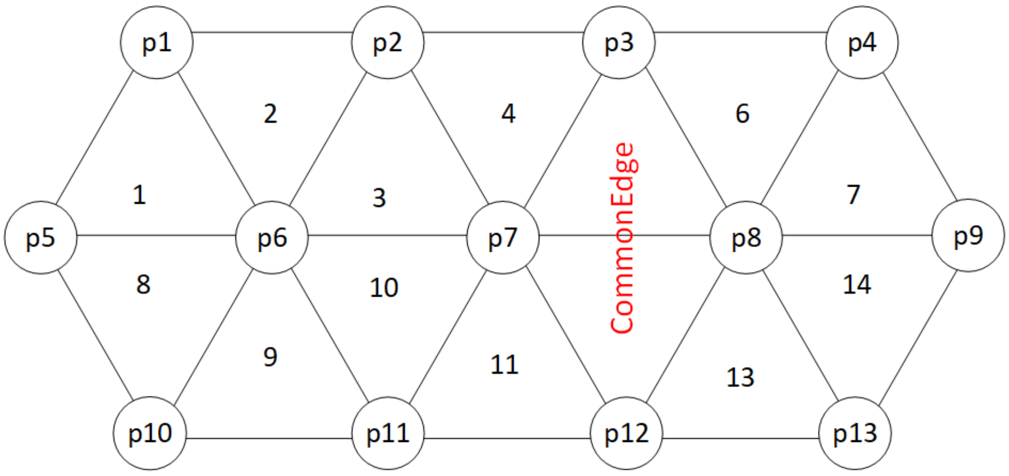
TriangleMesh = { Nodes{ p1,..., p16 }, Edges{ [p1,p2],..., [p12,p13] }, Tris{ 1,..., 14 } }
Pairs = { }
Tri1 = 4 Tri2 = 11
CommonPoint = p7
Pair1 = {3,10} Pair2 = {5,12}
CommonEdge = [p6,p7]
OtherPoint = p6
Triangles = {1, 2, 3, 8, 9, 10}
First = 2

(b)

Figure 3.7: Execution steps of the GETADJACENTPAIRS algorithm (cont.)



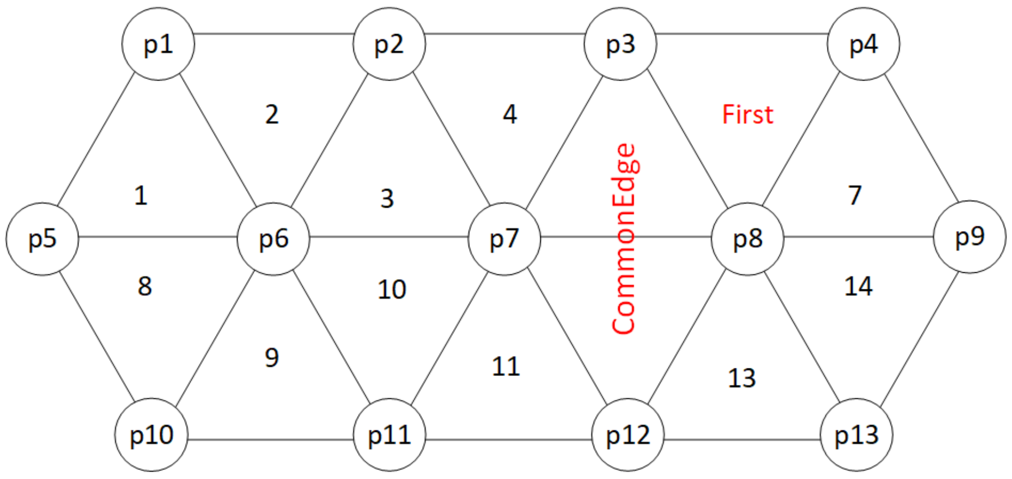
TriangleMesh = { Nodes{ p1,..., p16 }, Edges{ [p1,p2],..., [p12,p13] }, Tris{ 1,..., 14 } }
Pairs = { }
Tri1 = 4 Tri2 = 11
CommonPoint = p7
Pair1 = {3,10} Pair2 = {5,12}
CommonEdge = [p6,p7]
OtherPoint = p6
Triangles = {1, 2, 3, 8, 9, 10}
First = 2 Second = 9



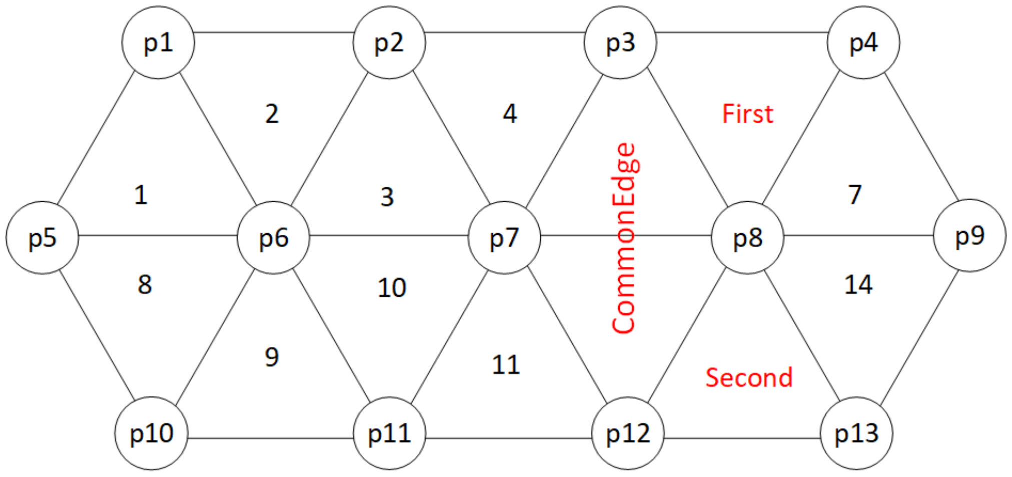
TriangleMesh = { Nodes{ p1,..., p16 }, Edges{ [p1,p2],..., [p12,p13] }, Tris{ 1,..., 14 } }
Pairs = { [2,9] }
Tri1 = 4 Tri2 = 11
CommonPoint = p7
Pair1 = {3,10} Pair2 = {5,12}
CommonEdge = [p7,p8]
OtherPoint = p8
Triangles = {5, 6, 7, 12, 13, 14}

(c)

Figure 3.7: Execution steps of the GETADJACENTPAIRS algorithm (cont.)



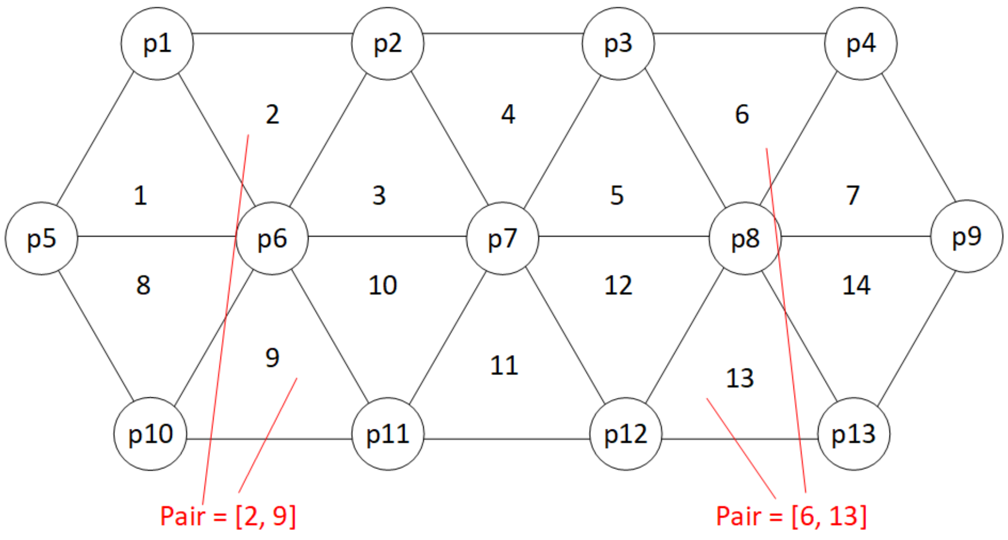
TriangleMesh = { Nodes{ p1,..., p16 }, Edges{ [p1,p2],..., [p12,p13] }, Tris{ 1,..., 14 } }
Pairs = { [2,9] }
Tri1 = 4 Tri2 = 11
CommonPoint = p7
Pair1 = {3,10} Pair2 = {5,12}
CommonEdge = [p7,p8]
OtherPoint = p8
Triangles = {5, 6, 7, 12, 13, 14}
First = 6



TriangleMesh = { Nodes{ p1,..., p16 }, Edges{ [p1,p2],..., [p12,p13] }, Tris{ 1,..., 14 } }
Pairs = { [2,9] }
Tri1 = 4 Tri2 = 11
CommonPoint = p7
Pair1 = {3,10} Pair2 = {5,12}
CommonEdge = [p7,p8]
OtherPoint = p8
Triangles = {5, 6, 7, 12, 13, 14}
First = 6 Second = 13

(d)

Figure 3.7: Execution steps of the GETADJACENTPAIRS algorithm (cont.)



TriangleMesh = { Nodes{ p1,..., p16 }, Edges{ [p1,p2],..., [p12,p13] }, Tris{ 1,..., 14 } }
Pairs = { [2,9], [6,13] }
Tri1 = 4 Tri2 = 11

(e)

Figure 3.7: Execution steps of the GETADJACENTPAIRS algorithm (cont.)

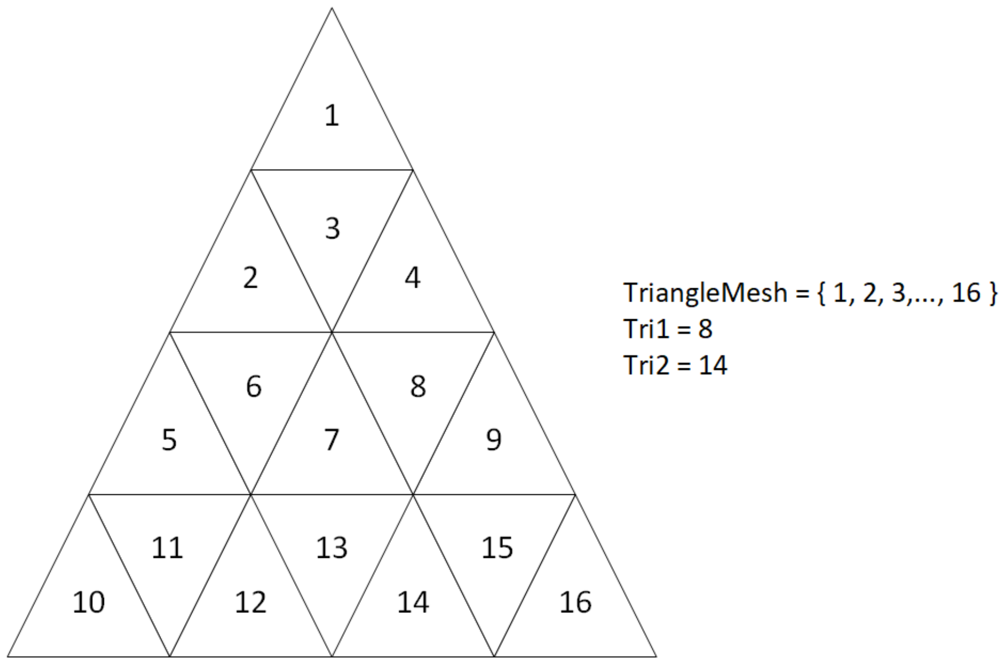
Finally, FINDBOWTIES algorithm delegates getting new triangle pair that can create a contiguous bow tie to a founded bow tie task to FINDCONTIGUOUSBOWTIE algorithm. How this task is achieved can be seen in Algorithm 3.2.6. This algorithm checks neighbor triangles contiguous to the given triangle pair and returns first neighbor couple which can create the new bow tie contiguous the bow tie created by given triangle pair. While during this operation, FINDCONTIGUOUSBOWTIE utilizes GETBOWTIECANDIDATETRIS described in Algorithm 3.2.2. Execution of the FINDCONTIGUOUSBOWTIE algorithm is shown in Figure 3.8.

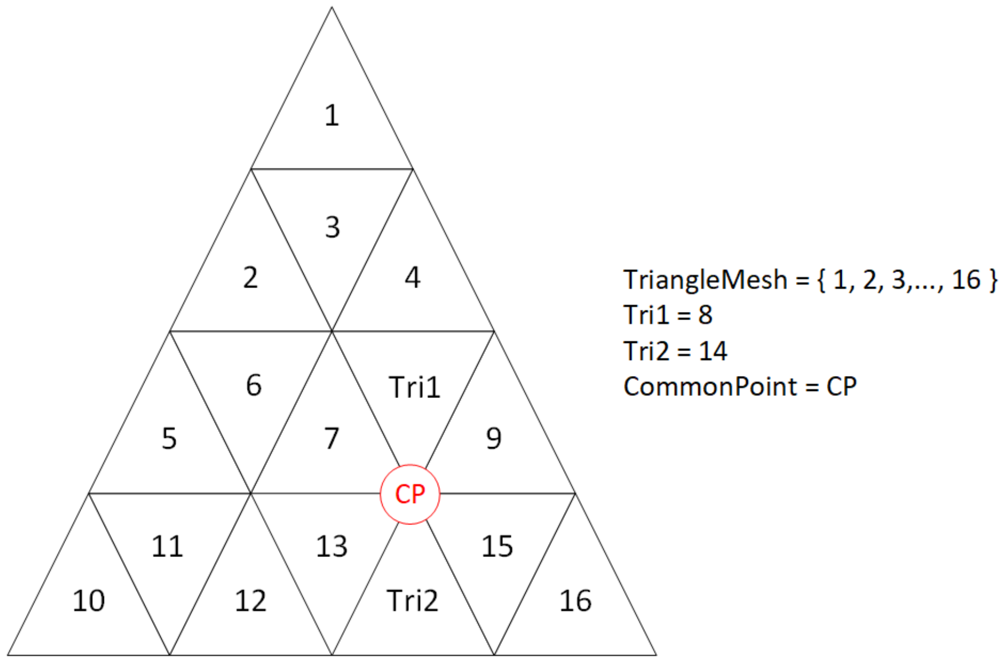
Besides, the algorithm FINDCONTIGUOUSBOWTIE is also employed by SEARCH described in 3.2.1. After finding bow ties process, SEARCH algorithm scans the founded bow ties and searches triangle pairs that can create contiguous bow ties to founded bow ties by using FINDCONTIGUOUSBOWTIE algorithm.

Algorithm 3.2.6 FINDCONTIGUOUSBOWTIE(TriangleMesh, Tri1, Tri2)

```
1: ContiguousBowTieTriPair = pair of triangles which can create bow tie
2: FirstTri = $null$
3: SecondTri = $null$
4: CommonPoint = common point of Tri1 and Tri2
5: if CommonPoint exists then
6:     Triangles = triangles containing CommonPoint
7:     for each triangle $Tri \in$ Triangles do
8:         if $Tri \neq$ Tri1 $and\ Tri \neq$ Tri2 $and\ !Tri.IsMatched$ then
9:             FirstTri = $Tri$
10:            break
11:        end if
12:    end for
13: end if
14: if FirstTri $\neq null$ then
15:     OtherPoint = other common point of Tri1 or Tri2 and FirstTri than CommonPoint
16:     if OtherPoint exists then
17:         BowTieCandidateTris = GETBOWTIECANDIDATETRIS(TriangleMesh, FirstTri)
18:         if BowTieCandidateTris $\neq$ empty then
19:             for each bow tie candidate tri $Tri \in$ BowTieCandidateTris do
20:                 if OtherPoint = common point of Tri and FirstTri $and\ !Tri.IsMatched$ then
21:                     SecondTri = $Tri$
```

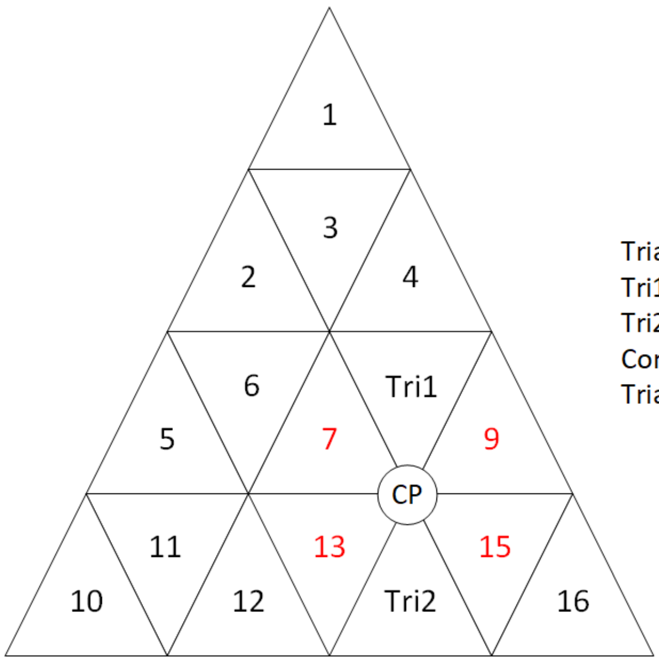
```
                    break
                end if
            end for
        end if
    end if
    if SecondTri $\neq null$ then
        ContiguousBowTieTriPair.First = FirstTri
        ContiguousBowTieTriPair.Second = SecondTri
    end if
end if
return ContiguousBowTieTriPair
```



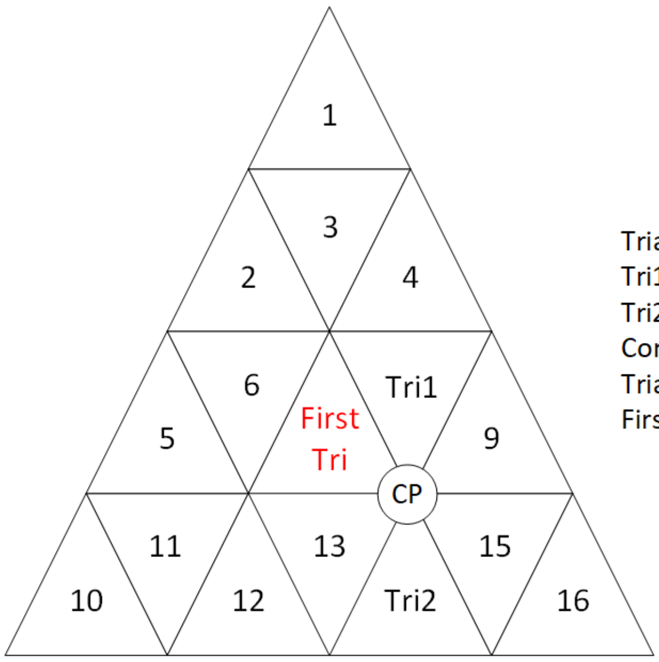


(a)

Figure 3.8: Execution steps of the FINDCONTIGUOUSBOWTIE algorithm



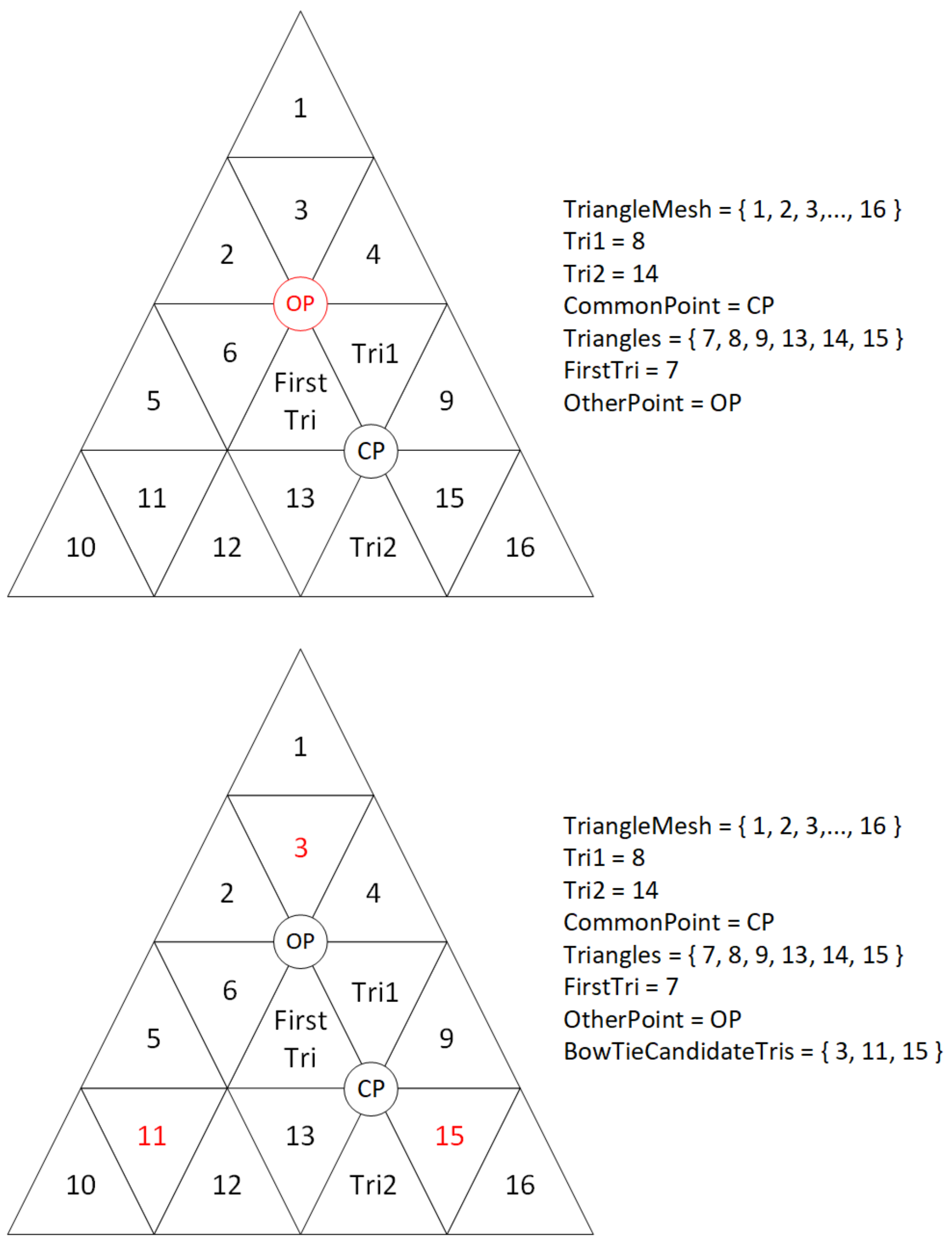
TriangleMesh = { 1, 2, 3,..., 16 }
Tri1 = 8
Tri2 = 14
CommonPoint = CP
Triangles = { 7, 8, 9, 13, 14, 15 }



TriangleMesh = { 1, 2, 3,..., 16 }
Tri1 = 8
Tri2 = 14
CommonPoint = CP
Triangles = { 7, 8, 9, 13, 14, 15 }
FirstTri = 7

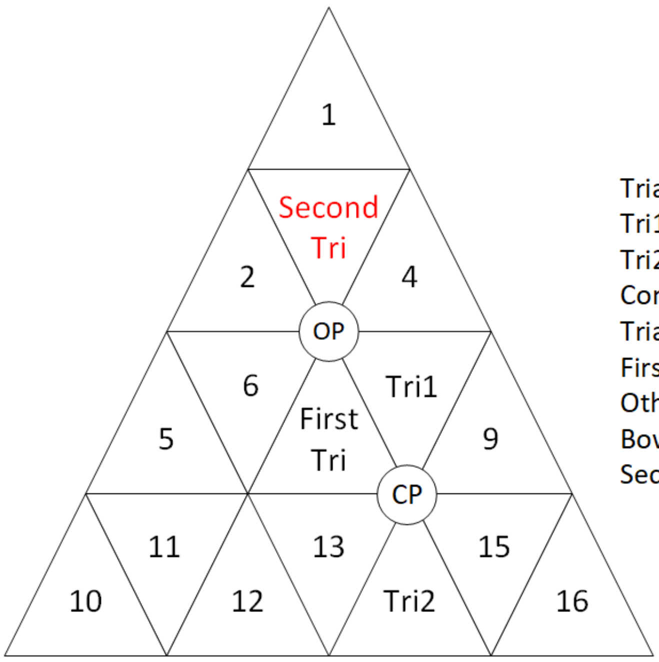
(b)

Figure 3.8: Execution steps of the FINDCONTIGUOUSBOWTIE algorithm (cont.)

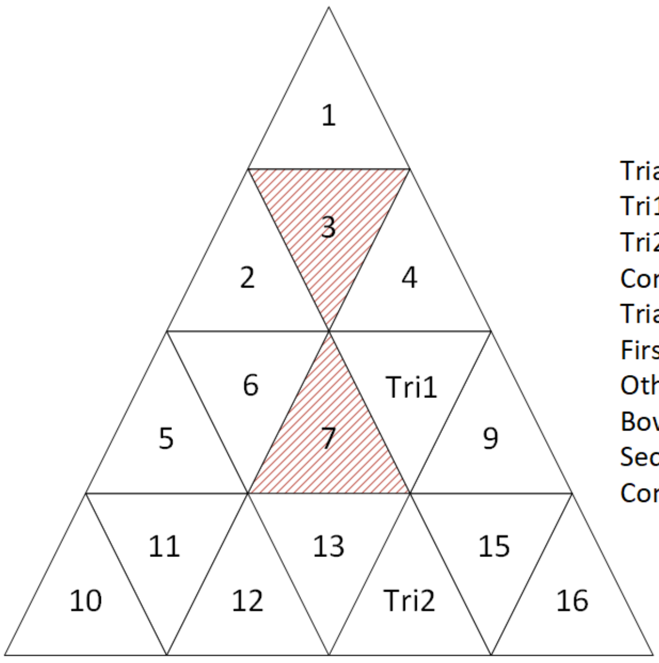


(c)

Figure 3.8: Execution steps of the FINDCONTIGUOUSBOWTIE algorithm (cont.)



TriangleMesh = { 1, 2, 3,..., 16 }
Tri1 = 8
Tri2 = 14
CommonPoint = CP
Triangles = { 7, 8, 9, 13, 14, 15 }
FirstTri = 7
OtherPoint = OP
BowTieCandidateTris = { 3, 11, 15 }
SecondTri = 3



TriangleMesh = { 1, 2, 3,..., 16 }
Tri1 = 8
Tri2 = 14
CommonPoint = CP
Triangles = { 7, 8, 9, 13, 14, 15 }
FirstTri = 7
OtherPoint = OP
BowTieCandidateTris = { 3, 11, 15 }
SecondTri = 3
ContiguousBowTieTriPair = [7, 3]

(d)

Figure 3.8: Execution steps of the FINDCONTIGUOUSBOWTIE algorithm (cont.)

This algorithm can extract auxetic patterns from triangle meshes; however, extracted auxetic patterns are tilted in some cases. A triangle mesh plane is shown in Figure 3.9. Extracted auxetic patterns from this plane by using the algorithm are shown in Figures 3.10, 3.11 and 3.12. There is no way to extract isotropic auxetic patterns that has a uniform flow over the input plane mesh. We observe an undesired anisotropic behavior that emphasizes a particular direction, which in turn causes problems after fabrication, e.g., output bends towards those directions. The reason for the non-uniform flow is that our algorithm creates hexagons with respect to triangles.

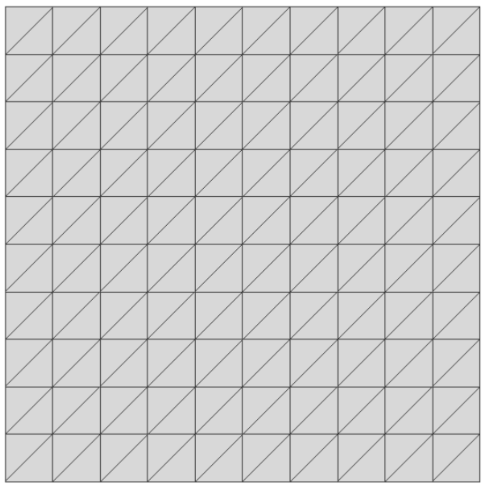


Figure 3.9: Triangle Mesh Plane

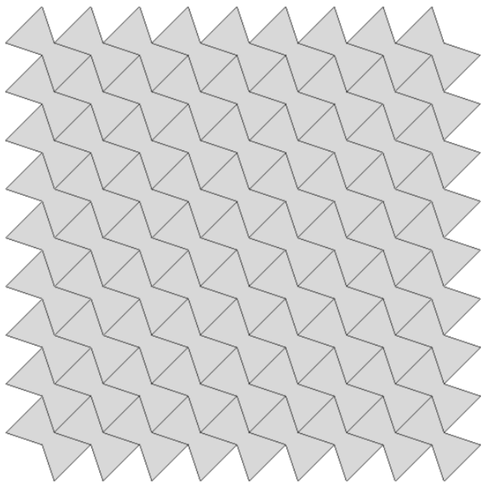


Figure 3.10: Extracted Auxetic Patterns - Cross

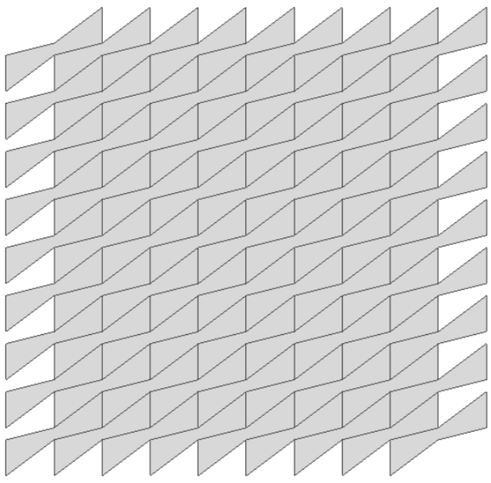


Figure 3.11: Extracted Auxetic Patterns - Horizontal

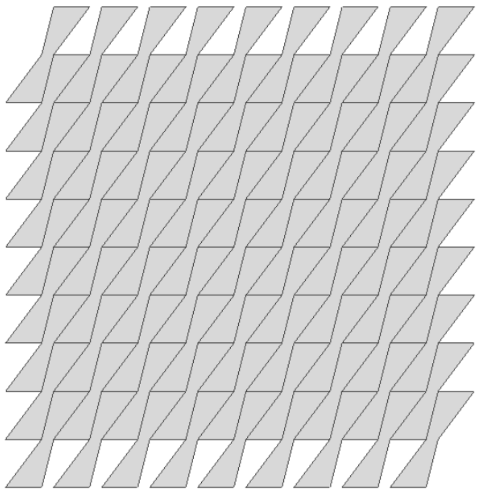


Figure 3.12: Extracted Auxetic Patterns - Vertical

3.3 Extracting Auxetic Patterns From Quad Meshes

Unlike triangle meshes, extracting isotropic auxetic patterns is possible from quad meshes because usual quad meshes are mostly consisting of quads combined as shown in Figure 3.3b. After designing an approach described in Section 3.2 to extract auxetic patterns from triangle meshes, we modified this approach for quad meshes. Additionally, we saw that we cannot extract auxetic patterns everywhere on shapes with the approach for triangle meshes. This situation rises disconnectedness problems at the resulted auxetic patterns. Then, we also extended our method in order to maintain connectivity of shapes while extracting auxetic patterns. The extended approach creates half hexagons at locations that cannot create complete hexagons if it is possible.

Extraction of the auxetic patterns from given quad mesh is shown in Algorithm 3.3.1. This algorithm searches and extracts auxetic patterns on given quad mesh starting from given quad and return the resulted structure.

Hexagons creating the auxetic patterns are named as bow ties and the extracted auxetic pattern structure referred as bow tie mesh at the rest of this section.

Algorithm 3.3.1 SEARCH(QuadMesh, StartingQuad)

```
1:  BowTieMesh = empty set of bow ties, their points and their edges
2:  QuadNeighbors = quads having a common edge with StartingQuad
3:  Write QuadNeighbors
4:  Read PairQuad
5:  FINDBOWTIES(BowTieMesh, QuadMesh, StartingQuad, PairQuad)
6:  for each quad $Q \in$ QuadMesh do
7:      if $Q.IsMatched$ then
8:          QuadNeighbors = quads having a common edge with $Q$
9:          for each neighbor quad $NQ \in$ QuadNeighbors do
10:             if $!NQ.IsMatched\ and\ Q$ is the only neighbor of $NQ$ which is matched then
11:                 NewStartingQuad = $NQ$
12:                 NewPairQuad = GETOPPOSITESIDENEIGHBOR(QuadMesh, NewStartingQuad ,
                    $Q$)
13:                 if NewPairQuad $\neq null$ then
14:                     FINDBOWTIES(BowTieMesh, QuadMesh, NewStartingQuad, NewPairQuad)
15:                     break
```

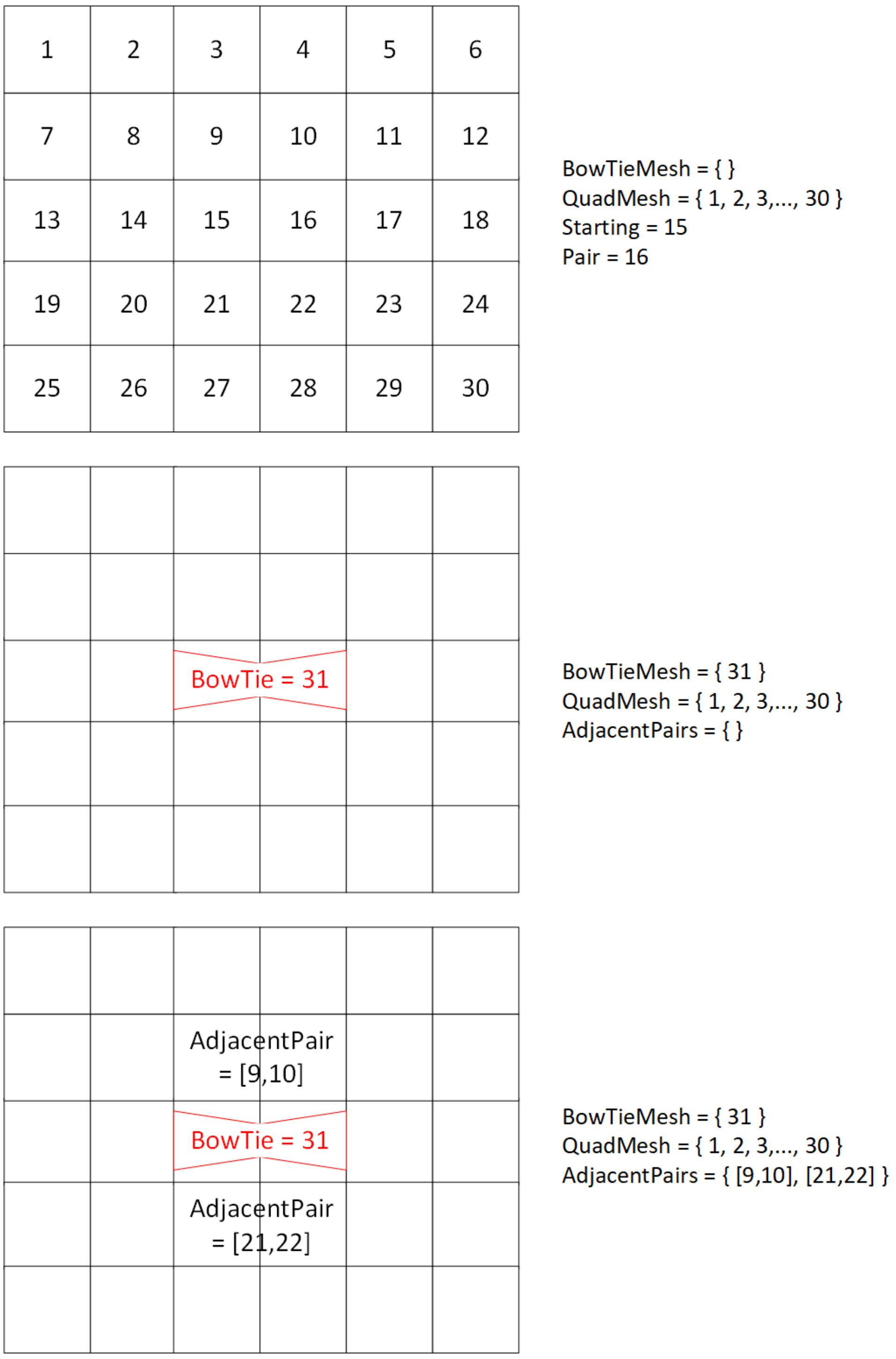
```
16:             end if
17:           end if
18:         end for
19:     end if
20: end for
21: FINDINTERNALBOWTIES(BowTieMesh, QuadMesh, StartingQuad)
22: for each quad $Q \in$ QuadMesh do
23:     if $!Q.IsInternalMatched\ and\ Q$ does not have any internal matched neighbor then
24:         NewStartingQuad = $Q$
25:         FINDINTERNALBOWTIES(BowTieMesh, QuadMesh, NewStartingQuad)
26:     end if
27: end for
28: return BowTieMesh
```

The SEARCH algorithm uses FINDBOWTIES algorithm to find bow ties and insert them into bow tie mesh structure. How to find, create and insert bow ties starting from a given triangle pair on is shown in Algorithm 3.3.2. This algorithm gets the quad pair, tries to create and insert a bow tie from this pair, if it succeeds then it tries to find out all bow ties can be created from neighbor quad pairs of this pair recursively, until it cannot find any bow tie. Execution of this algorithm is shown in Figure 3.13.

Algorithm 3.3.2 FINDBOWTIES(BowTieMesh, QuadMesh, StartingQuad, PairQuad)

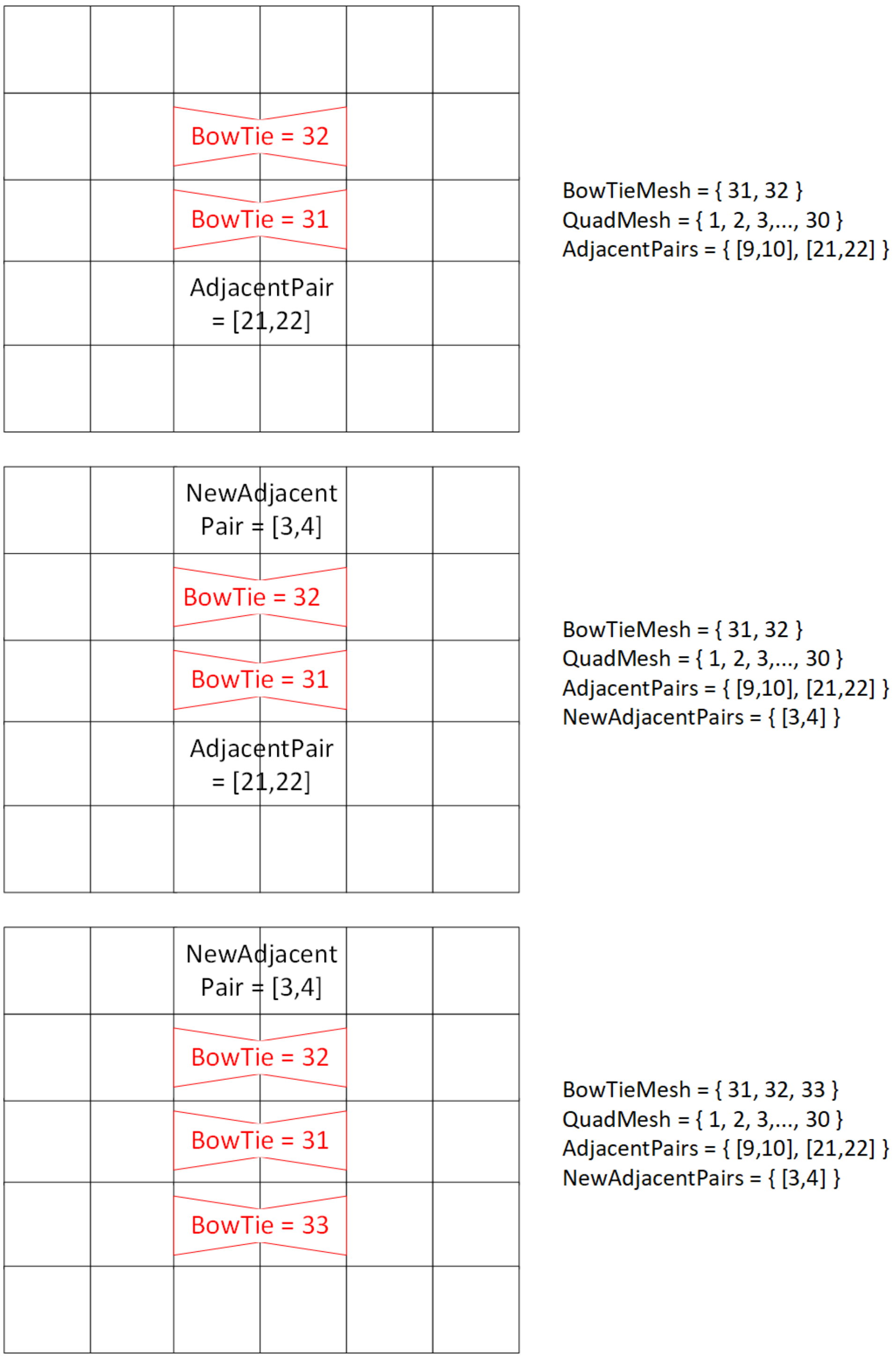
```
1: INSERTBOWTIE(BowTieMesh, QuadMesh, StartingQuad, PairQuad)
2: if bow tie was inserted from StartingQuad and PairQuad then
3:     AdjacentPairs = empty set of adjacent pairs of quads
4:     AdjacentPairs = GETADJACENTPAIRS(QuadMesh, Starting, Pair)
5:     while AdjacentPairs $\neq empty$ do
6:         NewAdjacentPairs = empty set of adjacent pairs of quads
7:         for each adjacent pair $P \in$ AdjacentPairs do
8:             INSERTBOWTIE(BowTieMesh, QuadMesh, $P.First$, $P.Second$)
9:             if bow tie was inserted from $P.First$ and $P.Second$ then
10:                Add [$P.First$, $P.Second$] to FoundedBowTieTriPairs
11:                AdcajentPairsFounded = GETADJACENTPAIRS(QuadMesh, $P.First$, $P.Second$)
12:                Add AdcajentPairsFounded to NewAdjacentPairs
13:            end if
```

```
14:         end for
15:         AdjacentPairs = NewAdjacentPairs
16:     end while
17:     NewStartingQuad = GETOPPOSITESIDENEIGHBOR(QuadMesh, StartingQuad, PairQuad)

18:     if NewStartingQuad $\neq null$ then
19:         NewPairQuad = GETOPPOSITESIDENEIGHBOR(QuadMesh, NewStartingQuad , Start-
            ingQuad)
20:         if NewPairQuad $\neq null$ then
21:             FINDBOWTIES(BowTieMesh, Quads, NewStartingQuad , NewPairQuad)
22:             NewStartingQuad = GETOPPOSITESIDENEIGHBOR(QuadMesh, PairQuad, StartingQuad)

23:             if NewStartingQuad $\neq null$ then
24:                 NewPairQuad = GETOPPOSITESIDENEIGHBOR(QuadMesh, NewStartingQuad ,
                    PairQuad)
25:                 if NewPairQuad $\neq null$ then
26:                     FINDBOWTIES(BowTieMesh, QuadMesh, NewStartingQuad, NewPairQuad)
27:                 end if
28:             end if
29:         end if
30:     end if
31: end if
```



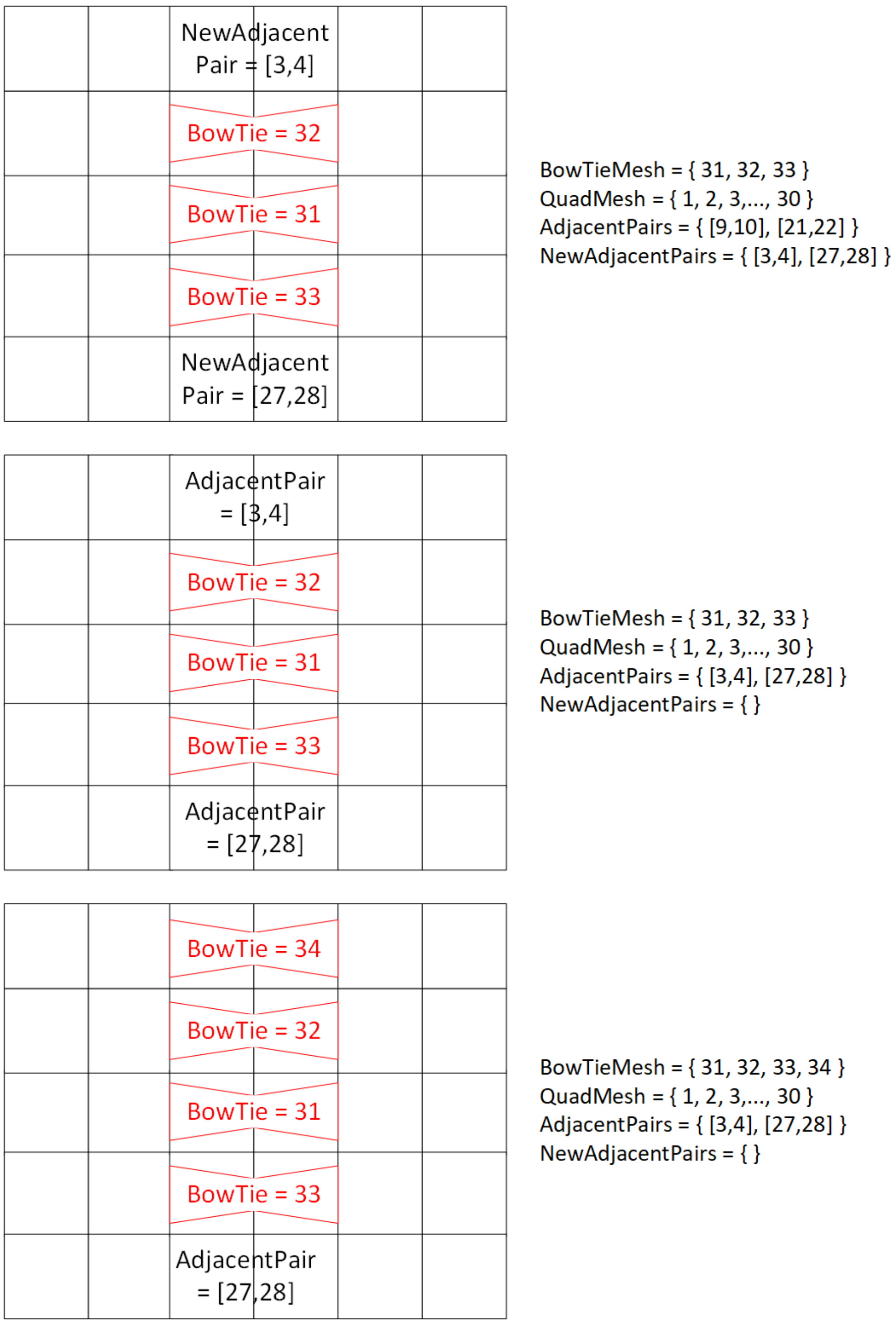
(a)

Figure 3.13: Execution steps of the FINDBOWTIES algorithm


BowTie = 32
BowTie = 31
AdjacentPair = [21,22]
BowTieMesh = { 31, 32 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [9,10], [21,22] }
NewAdjacent Pair = [3,4]
BowTie = 32
BowTie = 31
AdjacentPair = [21,22]
BowTieMesh = { 31, 32 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [9,10], [21,22] }
NewAdjacentPairs = { [3,4] }
NewAdjacent Pair = [3,4]
BowTie = 32
BowTie = 31
BowTie = 33
BowTieMesh = { 31, 32, 33 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [9,10], [21,22] }
NewAdjacentPairs = { [3,4] }

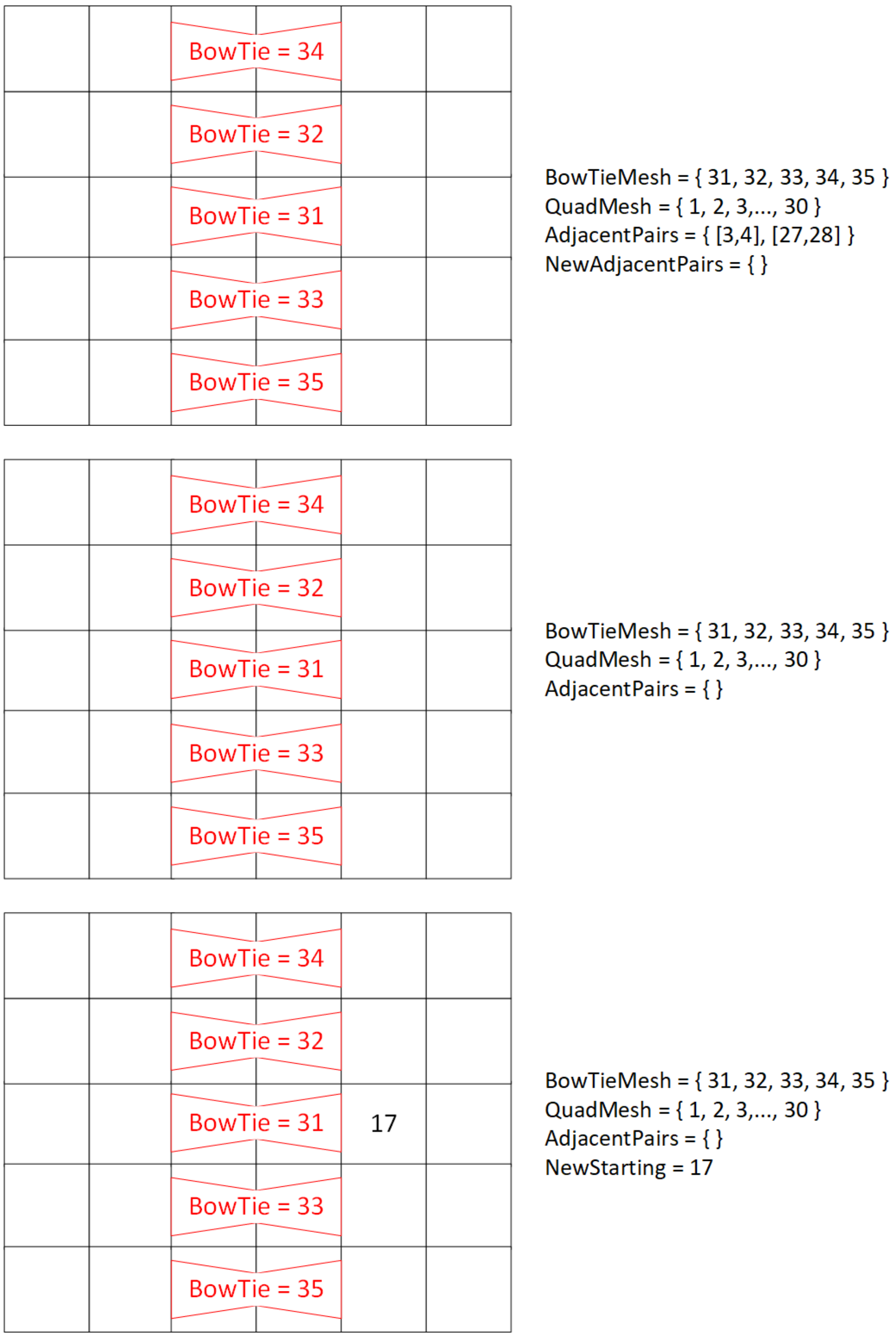
(b)

Figure 3.13: Execution steps of the FINDBOWTIES algorithm (cont.)


NewAdjacent Pair = [3,4]
BowTie = 32
BowTie = 31
BowTie = 33
NewAdjacent Pair = [27,28]
BowTieMesh = { 31, 32, 33 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [9,10], [21,22] }
NewAdjacentPairs = { [3,4], [27,28] }
AdjacentPair = [3,4]
BowTie = 32
BowTie = 31
BowTie = 33
AdjacentPair = [27,28]
BowTieMesh = { 31, 32, 33 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [3,4], [27,28] }
NewAdjacentPairs = { }
BowTie = 34
BowTie = 32
BowTie = 31
BowTie = 33
AdjacentPair = [27,28]
BowTieMesh = { 31, 32, 33, 34 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [3,4], [27,28] }
NewAdjacentPairs = { }

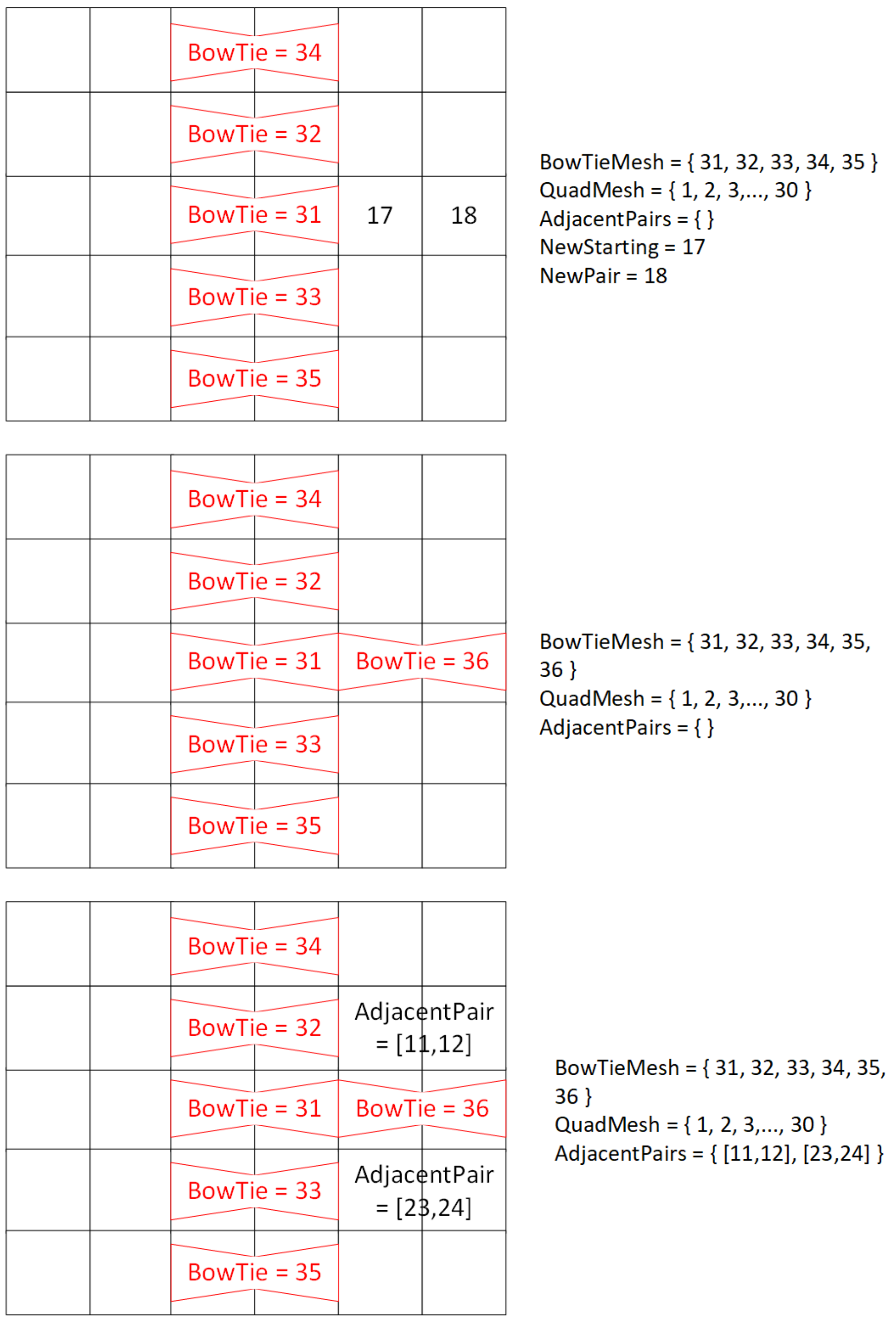
(c)

Figure 3.13: Execution steps of the FINDBOWTIES algorithm (cont.)



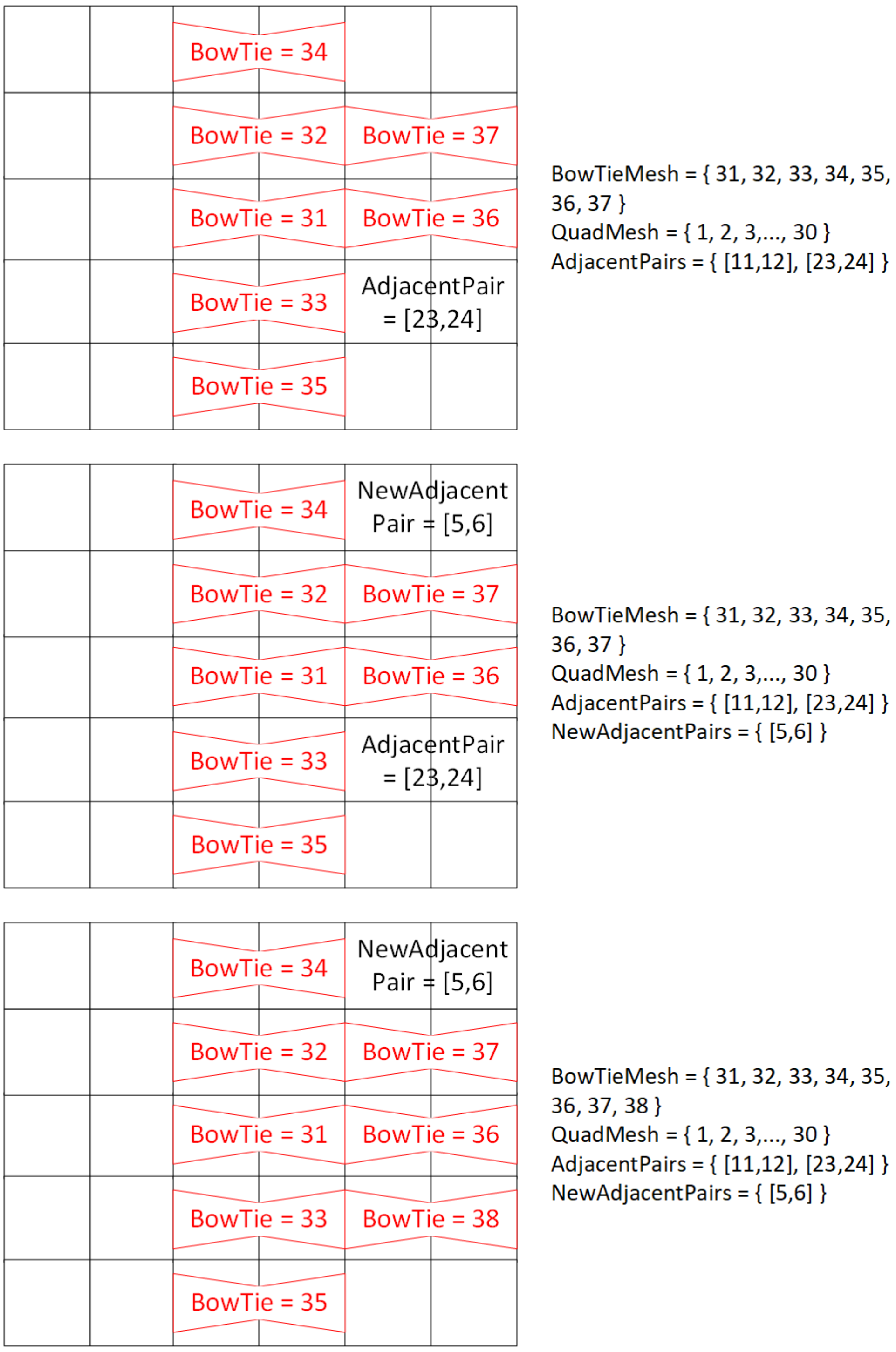
(d)

Figure 3.13: Execution steps of the FINDBOWTIES algorithm (cont.)



(e)

Figure 3.13: Execution steps of the FINDBOWTIES algorithm (cont.)



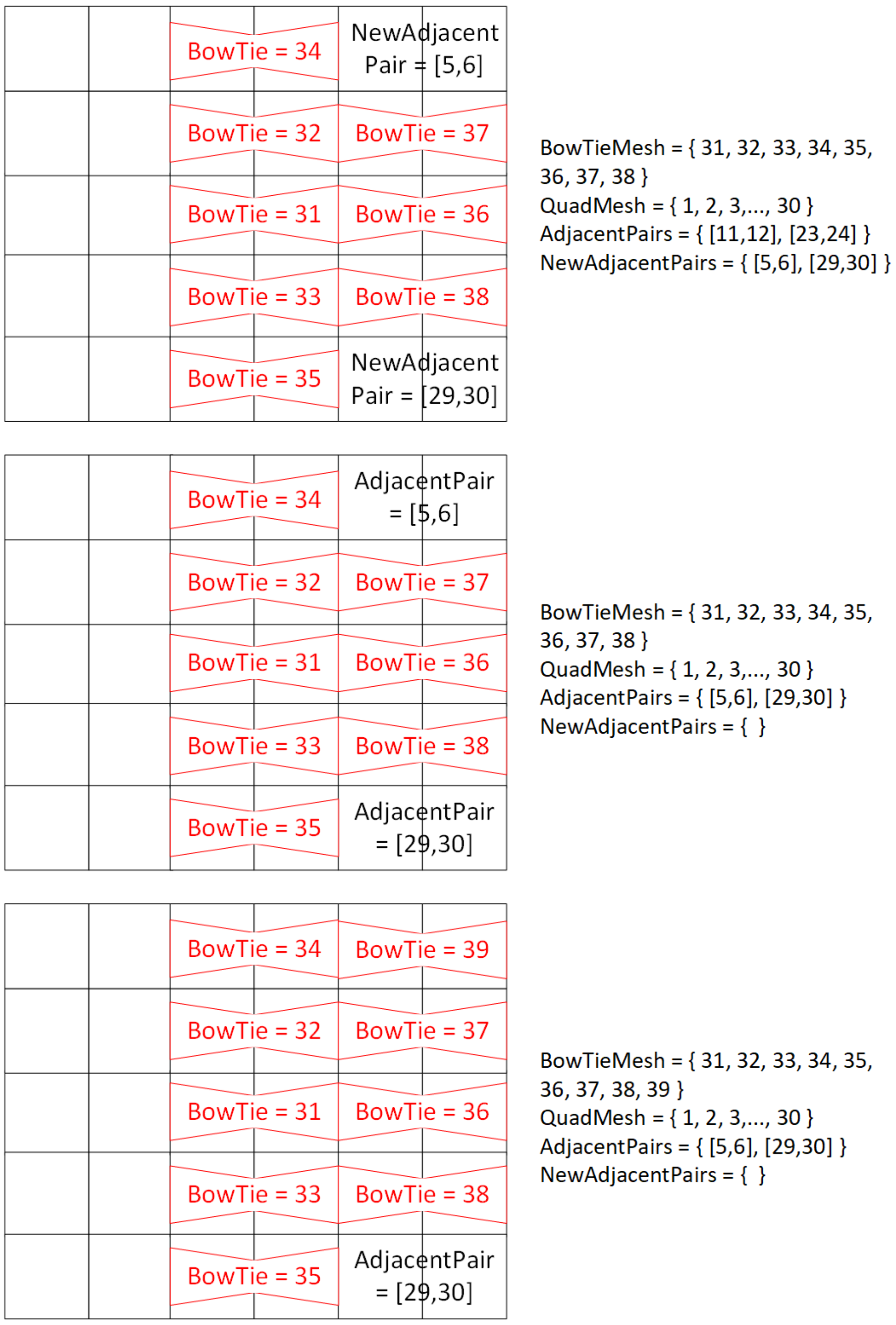
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [11,12], [23,24] }

BowTieMesh = { 31, 32, 33, 34, 35, 36, 37 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [11,12], [23,24] }
NewAdjacentPairs = { [5,6] }

BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [11,12], [23,24] }
NewAdjacentPairs = { [5,6] }

(f)

Figure 3.13: Execution steps of the FINDBOWTIES algorithm (cont.)



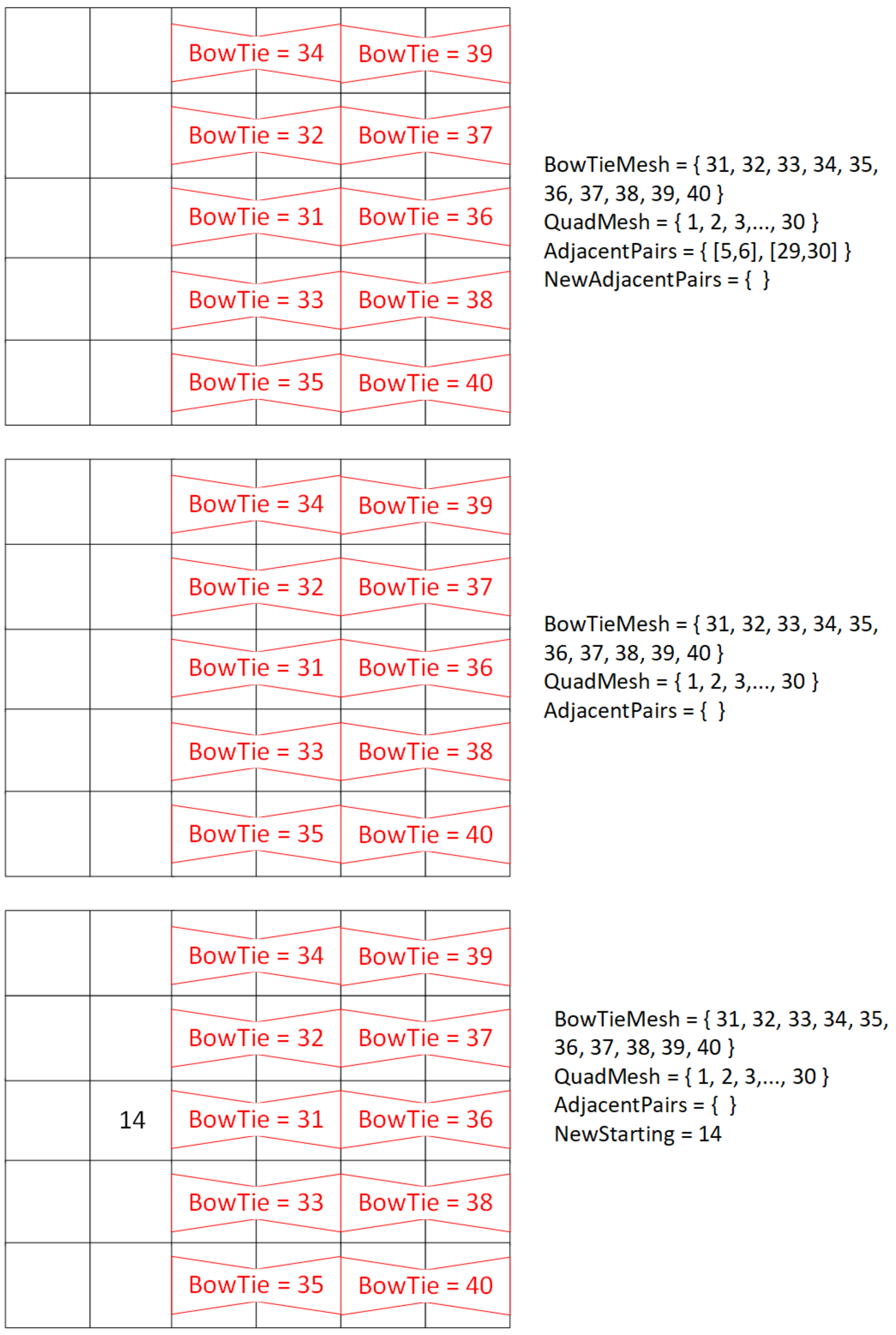
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [11,12], [23,24] }
NewAdjacentPairs = { [5,6], [29,30] }

BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [5,6], [29,30] }
NewAdjacentPairs = { }

BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [5,6], [29,30] }
NewAdjacentPairs = { }

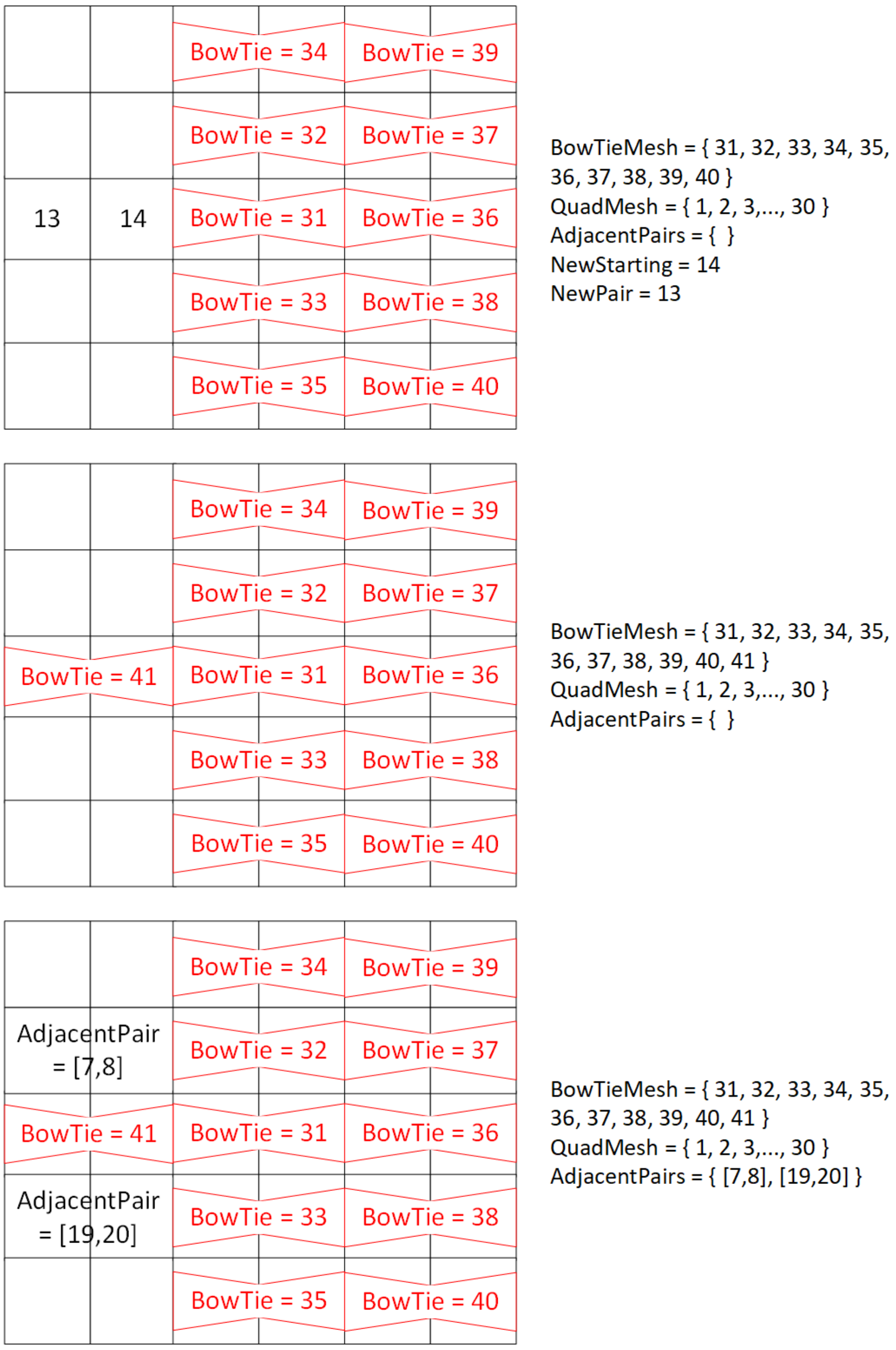
(g)

Figure 3.13: Execution steps of the FINDBOWTIES algorithm (cont.)



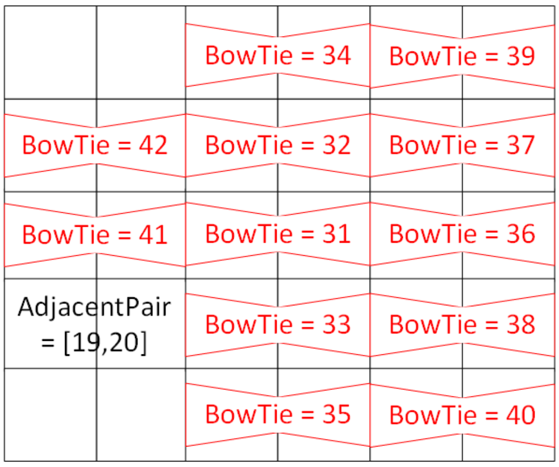
(h)

Figure 3.13: Execution steps of the FINDBOWTIES algorithm (cont.)

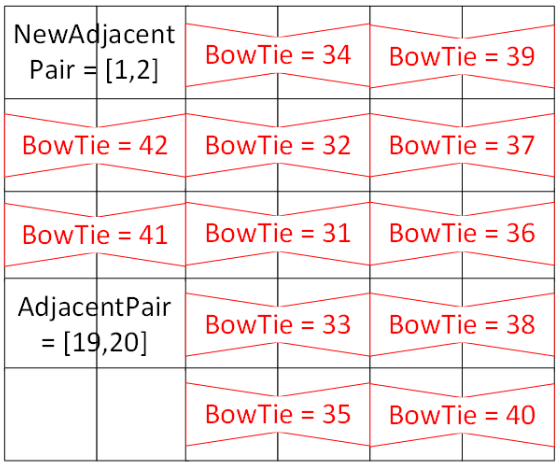


(i)

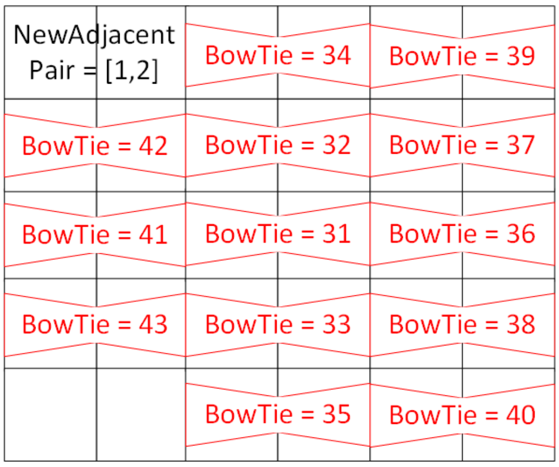
Figure 3.13: Execution steps of the FINDBOWTIES algorithm (cont.)



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [7,8], [19,20] }



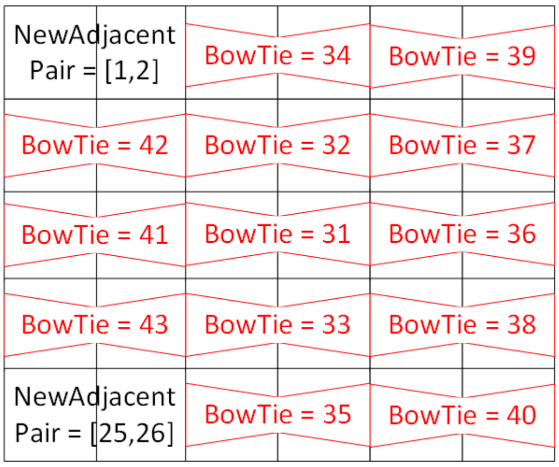
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [7,8], [19,20] }
NewAdjacentPairs = { [1,2] }



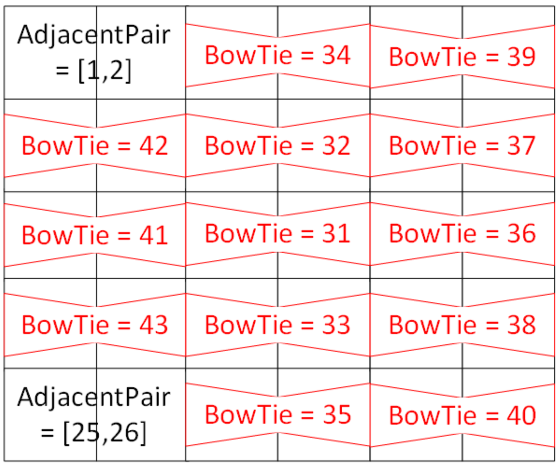
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [7,8], [19,20] }
NewAdjacentPairs = { [1,2] }

(j)

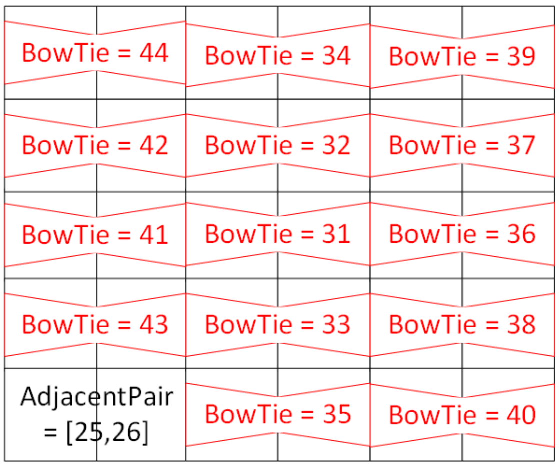
Figure 3.13: Execution steps of the FINDBOWTIES algorithm (cont.)



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [7,8], [19,20] }
NewAdjacentPairs = { [1,2], [25,26] }



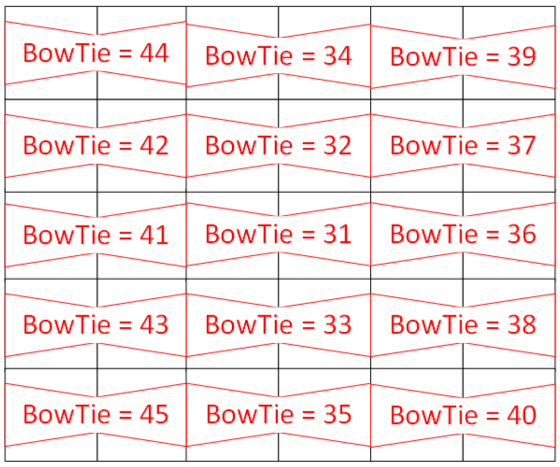
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [1,2], [25,26] }
NewAdjacentPairs = { }



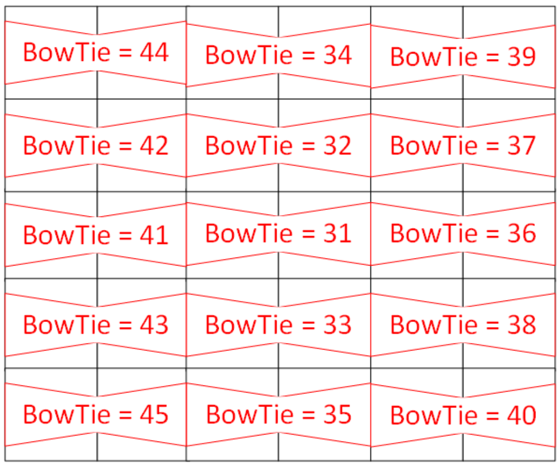
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [1,2], [25,26] }
NewAdjacentPairs = { }

(k)

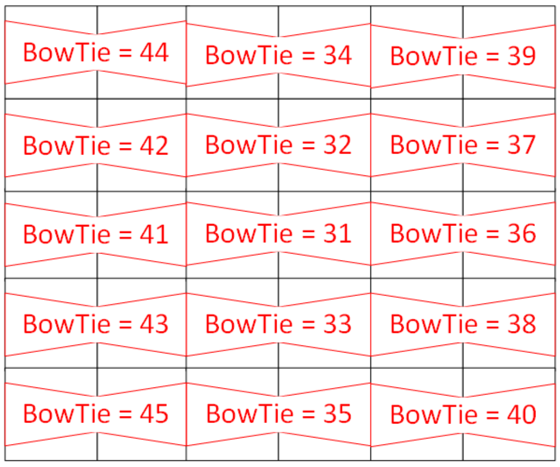
Figure 3.13: Execution steps of the FINDBOWTIES algorithm (cont.)



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { [1,2], [25,26] }
NewAdjacentPairs = { }



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { }



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45 }
QuadMesh = { 1, 2, 3,..., 30 }
AdjacentPairs = { }
NewStarting = NULL

(l)

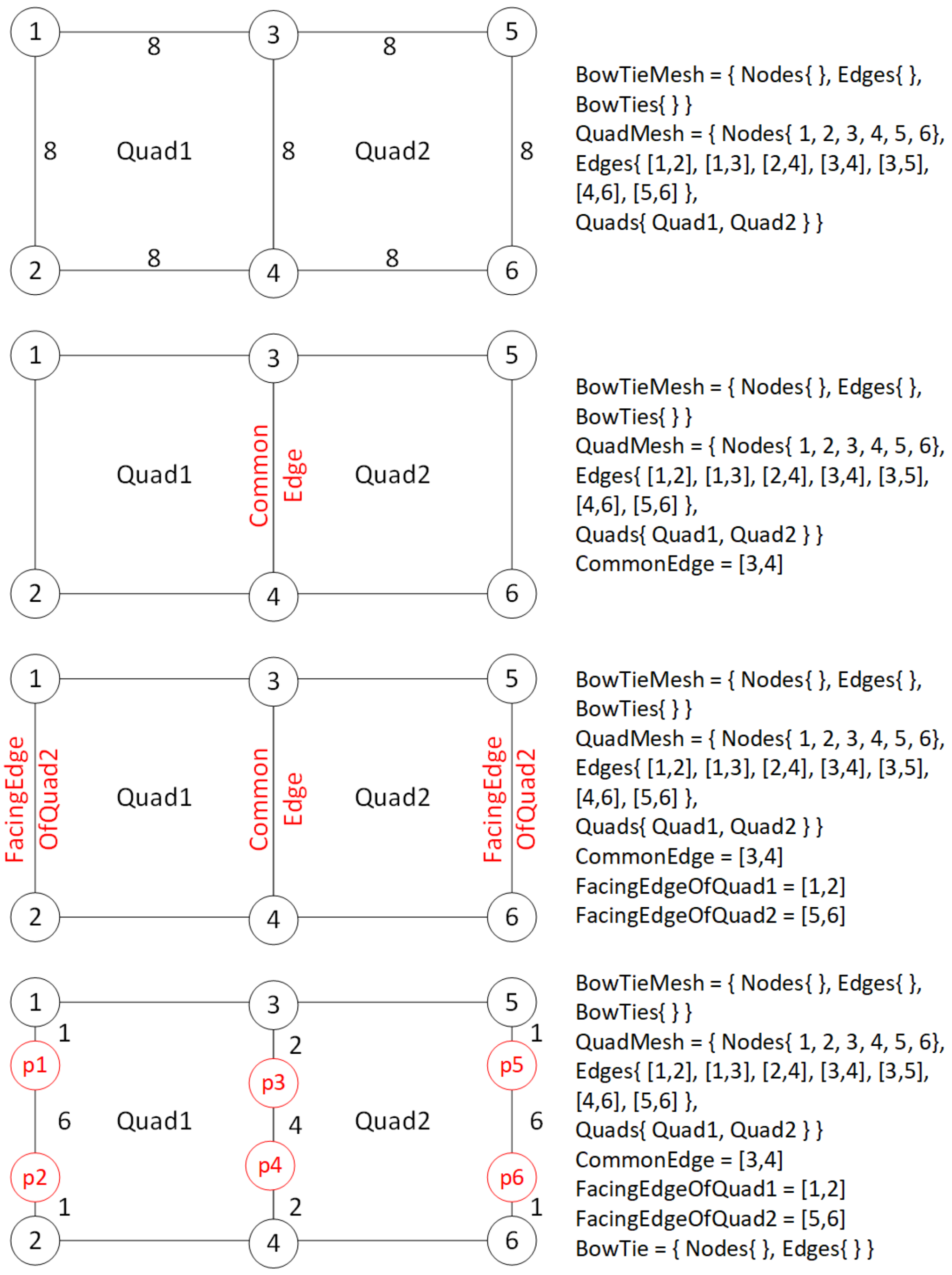
Figure 3.13: Execution steps of the FINDBOWTIES algorithm (cont.)

FINDBOWTIES algorithm employs INSERTBOWTIE algorithm to create and insert bow tie. Bow tie creation from a quad pair is shown in Algorithm 3.3.3. This algorithm calculates coordinates of points creating the bow tie and orders these points to create a related bow tie. Then, inserts the points and edges consisting of these points into bow tie mesh. Execution of this algorithm is shown in Figure 3.14.

Algorithm 3.3.3 INSERTBOWTIE(BowTieMesh, QuadMesh, Quad1, Quad2)

```
1: if !Quad1.IsMatched $and$ !Quad2.IsMatched then
2:     CommonEdge = common edge between Quad1 and Quad2
3:     if CommonEdge exists then
4:         FacingEdgeOfQuad1 = Quad1's facing edge to CommonEdge
5:         FacingEdgeOfQuad2 = Quad2's facing edge to CommonEdge
6:         if FacingEdgeOfQuad1 exists $and$ FacingEdgeOfQuad2 exists then
7:             p1 = FacingEdgeOfQuad1.First + (lengt of FacingEdgeOfQuad1 ) * 1/8
8:             p2 = FacingEdgeOfQuad1.Second + (lengt of FacingEdgeOfQuad1 ) * 1/8
9:             p3 = CommonEdge.First + (length of CommonEdge) * 3 / 8
10:            p4 = CommonEdge.Second + (length of CommonEdge) * 3 / 8
11:            p5 = FacingEdgeOfQuad2.First + (length of FacingEdgeOfQuad2) * 1/8
12:            p6 = FacingEdgeOfQuad2.Second + (length of FacingEdgeOfQuad2) * 1/8
13:            Add points p1, p2, p3, p4, p5, p6 to BowTieMesh if they are not added before
14:            BowTie = empty set of ordered points creating bow tie
15:            Add point p1 to BowTie
16:            Add point p2 to BowTie
17:            if p3 is closer than p4 to p2 then
18:                Add point p3 to BowTie
19:                if p5 is closer than p6 to p3 then
20:                    Add point p5 to BowTie
21:                    Add point p6 to BowTie
22:                else
23:                    Add point p6 to BowTie
24:                    Add point p5 to BowTie
25:                end if
26:                Add point p4 to BowTie
27:            else
28:                Add point p4 to BowTie
29:                if p5 is closer than p6 to p3 then
```

```
30:                     Add point p5 to BowTie
31:                     Add point p6 to BowTie
32:                 else
33:                     Add point p6 to BowTie
34:                     Add point p5 to BowTie
35:                 end if
36:                 Add point p3 to BowTie
37:             end if
38:             if Quad1's normal vector and BowTie's normal vector are in the opposite directions
                then
39:                 Reverse points of BowTie
40:             end if
41:             Add BowTie to BowTieMesh
42:             Quad1.IsMatched = $true$
43:             Quad1.MatchedQuad = Quad2
44:             Quad2.IsMatched = $true$
45:             Quad2.MatchedQuad = Quad1
46:         end if
47:     end if
48: end if
```



BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] }, Quads{ Quad1, Quad2 } }

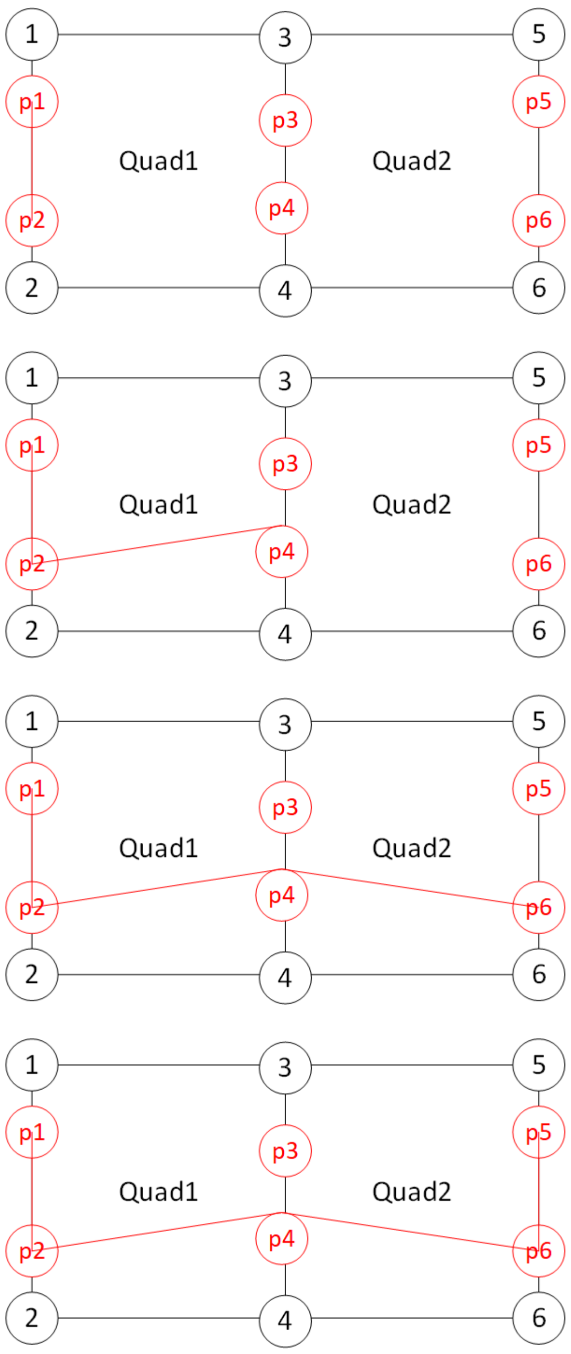
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] }, Quads{ Quad1, Quad2 } }
CommonEdge = [3,4]

BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] }, Quads{ Quad1, Quad2 } }
CommonEdge = [3,4]
FacingEdgeOfQuad1 = [1,2]
FacingEdgeOfQuad2 = [5,6]

BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] }, Quads{ Quad1, Quad2 } }
CommonEdge = [3,4]
FacingEdgeOfQuad1 = [1,2]
FacingEdgeOfQuad2 = [5,6]
BowTie = { Nodes{ }, Edges{ } }

(a)

Figure 3.14: Execution steps of the INSERTBOWTIE algorithm



BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6}, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] },
BowTie = { Nodes{ p1, p2}, Edges{ [p1, p2] } }

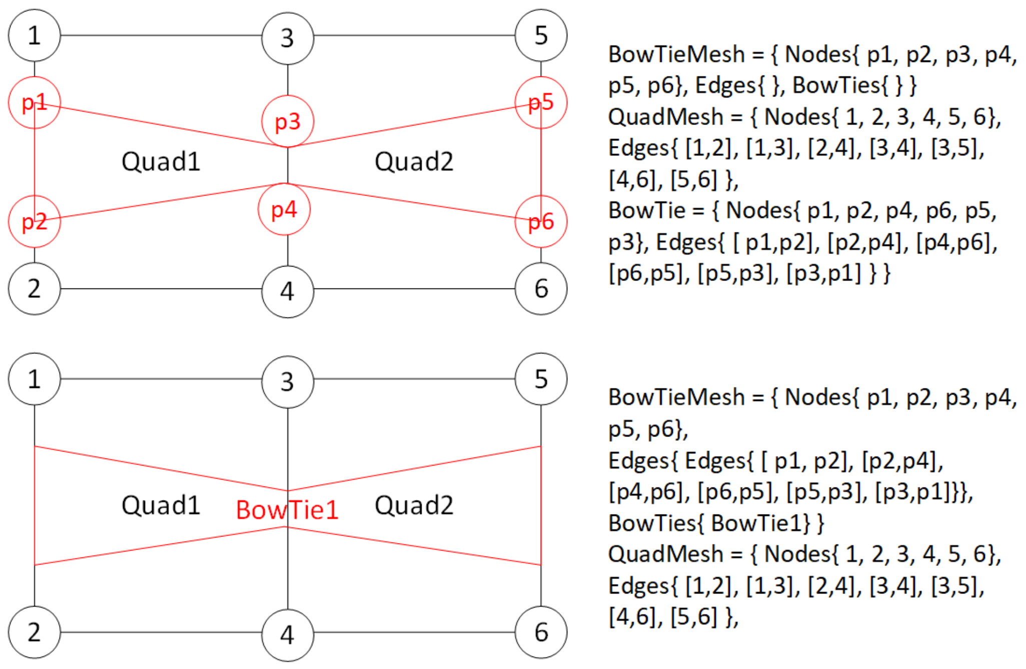
BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6}, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] },
BowTie = { Nodes{ p1, p2, p4}, Edges{ [p1,p2], [p2,p4] } }

BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6}, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] },
BowTie = { Nodes{ p1, p2, p4, p6}, Edges{ [p1,p2], [p2,p4], [p4,p6] } }

BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6}, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] },
BowTie = { Nodes{ p1, p2, p4, p6, p5}, Edges{ [p1,p2], [p2,p4], [p4,p6], [p6,p5] } }

(b)

Figure 3.14: Execution steps of the INSERTBOWTIE algorithm (cont.)



BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6}, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] },
BowTie = { Nodes{ p1, p2, p4, p6, p5, p3}, Edges{ [p1,p2], [p2,p4], [p4,p6], [p6,p5], [p5,p3], [p3,p1] } }

BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6},
Edges{ Edges{ [p1, p2], [p2,p4], [p4,p6], [p6,p5], [p5,p3], [p3,p1]}},
BowTies{ BowTie1} }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] },

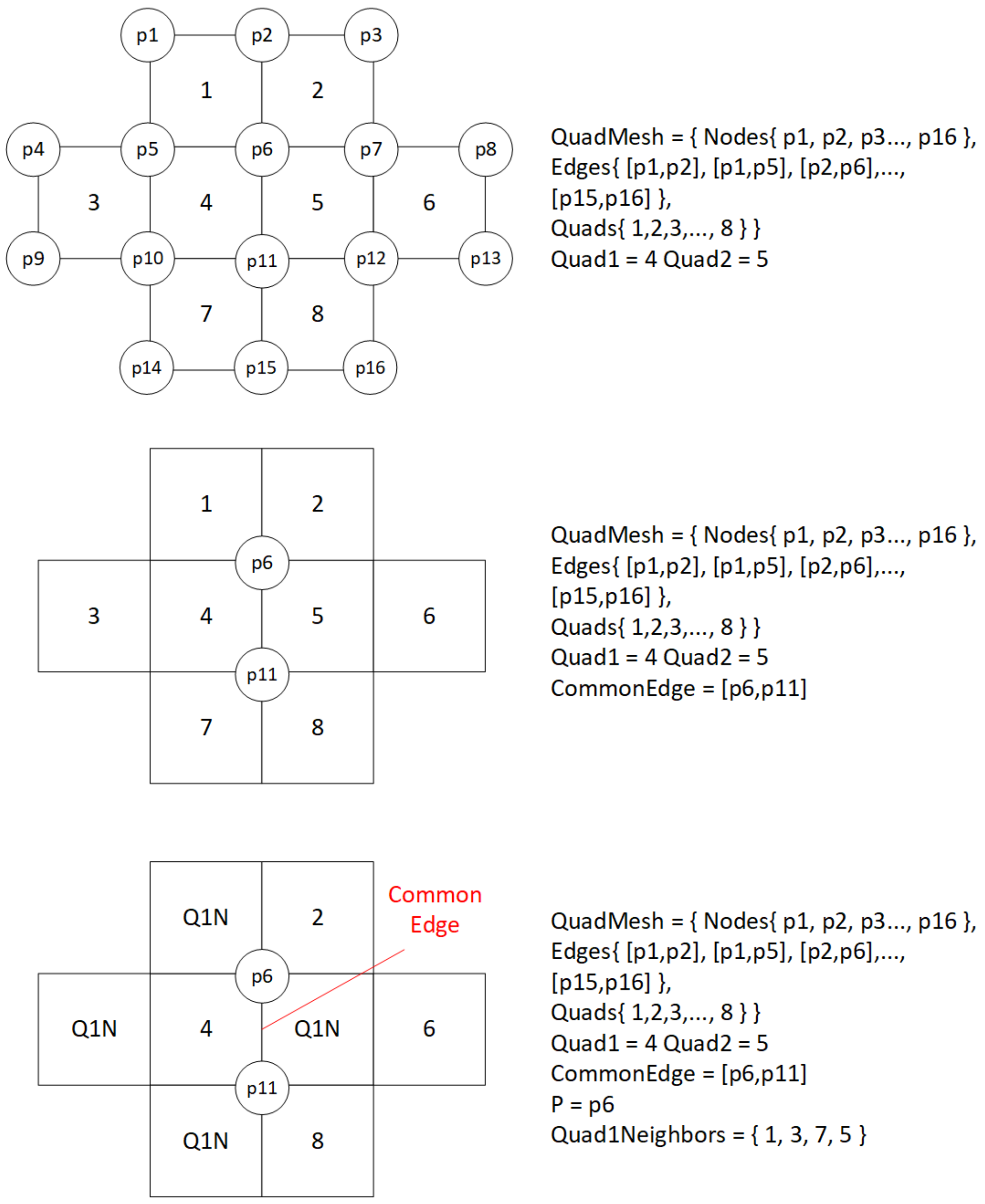
(c)

Figure 3.14: Execution steps of the INSERTBOWTIE algorithm (cont.)

Additionally, FINDBOWTIES algorithm uses GETADJACENTPAIRS algorithm for finding adjacent quad pairs to quad pairs created bow ties. Finding procedure is shown in Algorithm 3.3.4. This algorithm checks neighbor quads of the given quad pair and couples proper ones which can create new bow ties. Execution of this algorithm is shown in Figure 3.15.

Algorithm 3.3.4 GETADJACENTPAIRS(QuadMesh, Quad1, Quad2)

```
1: Pairs = empty set of pairs of quads
2: CommonEdge = common edge between Quad1 and Quad2
3: if CommonEdge exists then
4:     for each point $P \in$ CommonEdge do
5:         First = $null$
6:         Second = $null$
7:         Quad1Neighbors = quads having a common edge with Quad1
8:         for each neighbor quad $NQ \in$ Quad1Neighbors do
9:             if $NQ \neq$ Quad2 $and$ $P$ is a point of $NQ$ then
10:                First = $NQ$
11:                break
12:            end if
13:        end for
14:        Quad2Neighbors = quads having a common edge with Quad2
15:        for each neighbor quad $NQ \in$ Quad2Neighbors do
16:            if $NQ \neq$ Quad1 $and$ $P$ is a point of $NQ$ then
17:                Second = $NQ$
18:                break
19:            end if
20:        end for
21:        if First $\neq$ $null$ $and$ !First.IsMatched $and$ Second $\neq$ $null$ $and$ !Second.IsMatched then
22:            Add [First, Second] to Pairs
23:        end if
24:    end for
25: end if
26: return Pairs
```



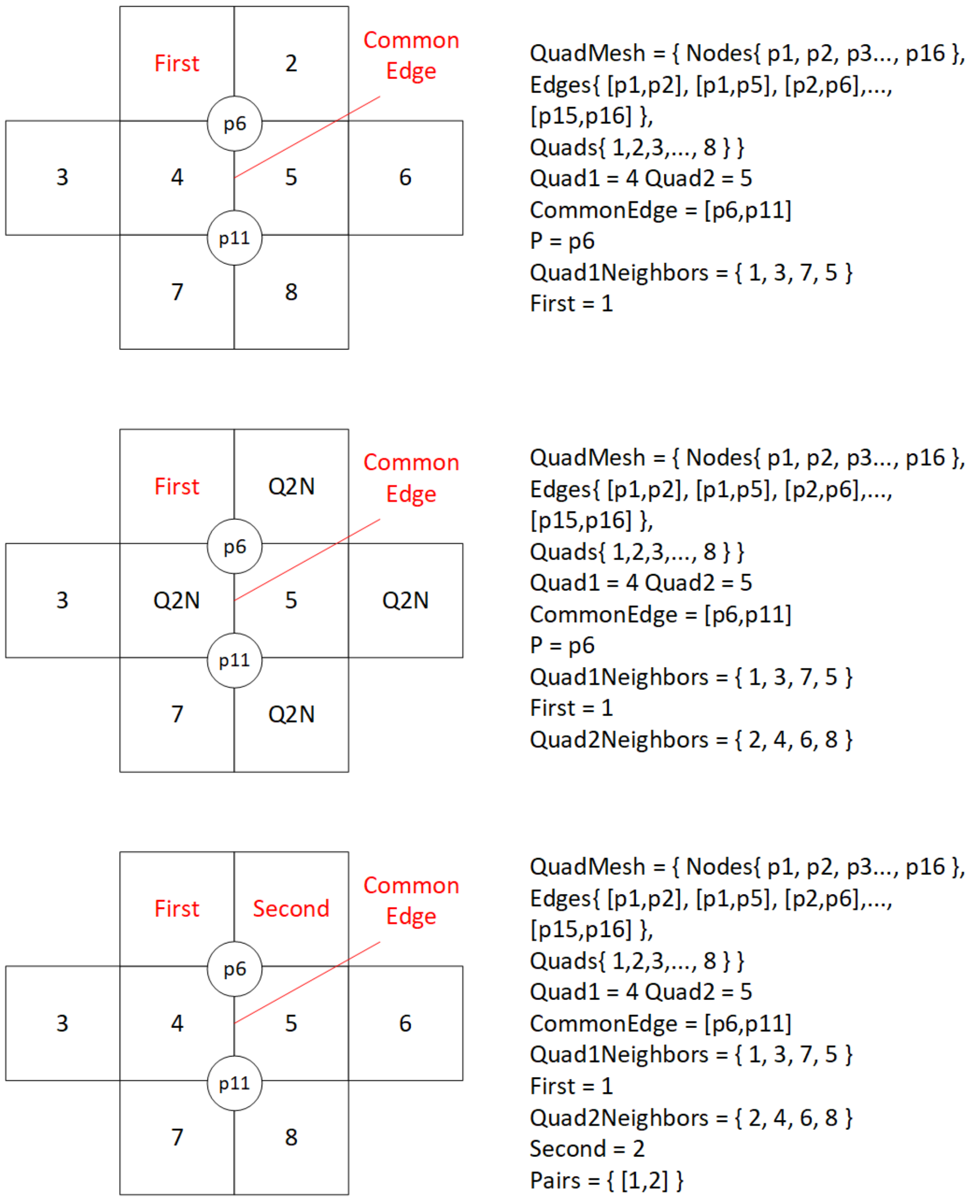
```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
```

```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
CommonEdge = [p6,p11]
```

```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
CommonEdge = [p6,p11]
P = p6
Quad1Neighbors = { 1, 3, 7, 5 }
```

(a)

Figure 3.15: Execution steps of the GETADJACENTPAIRS algorithm



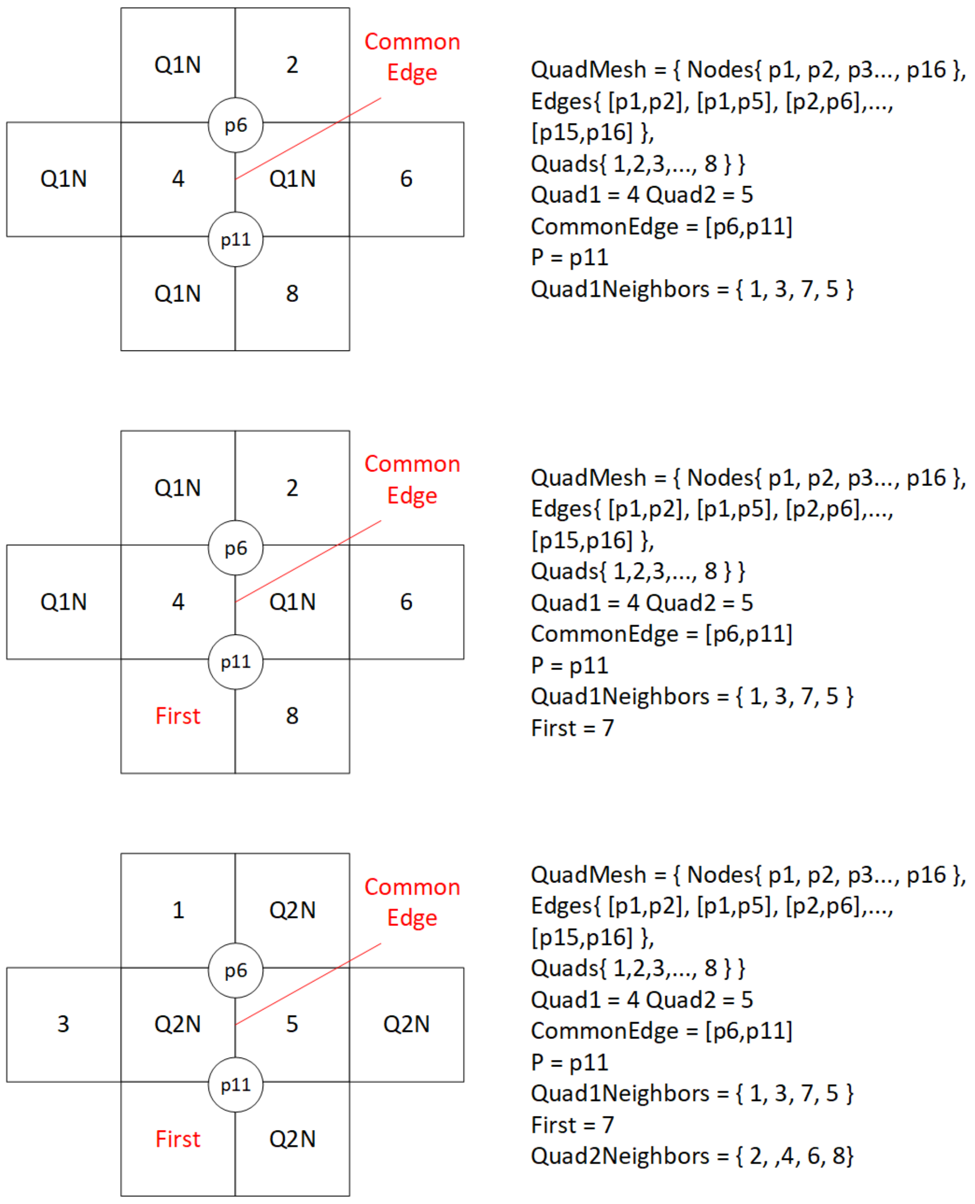
```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
CommonEdge = [p6,p11]
P = p6
Quad1Neighbors = { 1, 3, 7, 5 }
First = 1
```

```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
CommonEdge = [p6,p11]
P = p6
Quad1Neighbors = { 1, 3, 7, 5 }
First = 1
Quad2Neighbors = { 2, 4, 6, 8 }
```

```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
CommonEdge = [p6,p11]
Quad1Neighbors = { 1, 3, 7, 5 }
First = 1
Quad2Neighbors = { 2, 4, 6, 8 }
Second = 2
Pairs = { [1,2] }
```

(b)

Figure 3.15: Execution steps of the GETADJACENTPAIRS algorithm (cont.)



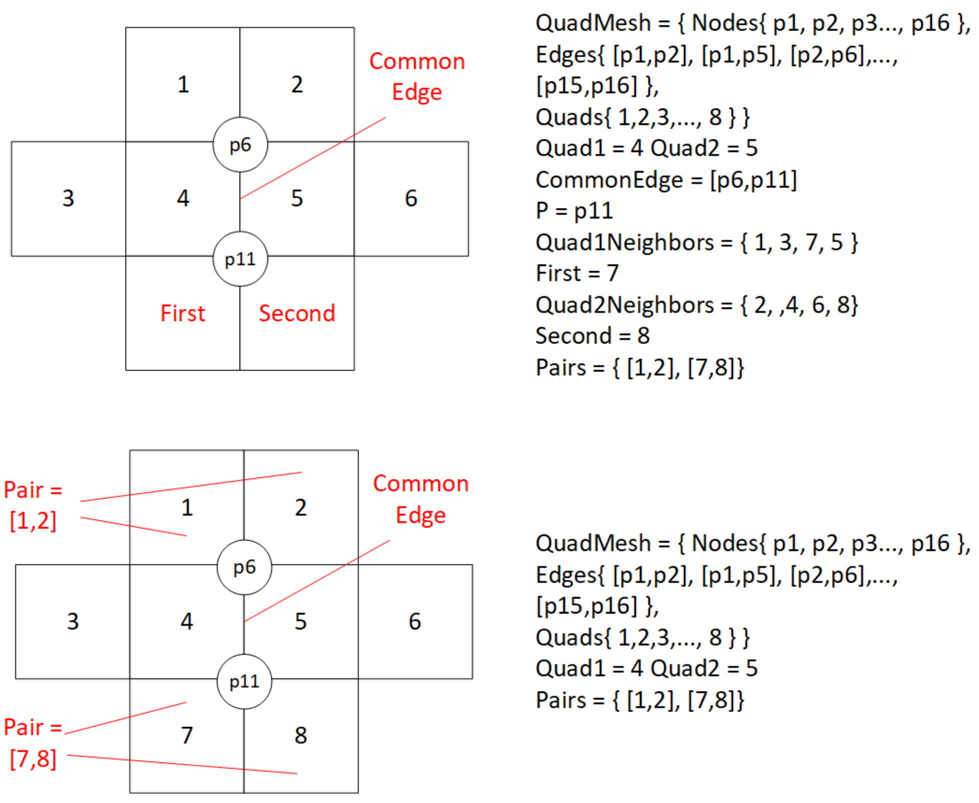
```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
CommonEdge = [p6,p11]
P = p11
Quad1Neighbors = { 1, 3, 7, 5 }
```

```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
CommonEdge = [p6,p11]
P = p11
Quad1Neighbors = { 1, 3, 7, 5 }
First = 7
```

```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
CommonEdge = [p6,p11]
P = p11
Quad1Neighbors = { 1, 3, 7, 5 }
First = 7
Quad2Neighbors = { 2, ,4, 6, 8}
```

(c)

Figure 3.15: Execution steps of the GETADJACENTPAIRS algorithm (cont.)



QuadMesh = { Nodes{ p1, p2, p3..., p16 }, Edges{ [p1,p2], [p1,p5], [p2,p6],..., [p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
CommonEdge = [p6,p11]
P = p11
Quad1Neighbors = { 1, 3, 7, 5 }
First = 7
Quad2Neighbors = { 2, ,4, 6, 8}
Second = 8
Pairs = { [1,2], [7,8]}

QuadMesh = { Nodes{ p1, p2, p3..., p16 }, Edges{ [p1,p2], [p1,p5], [p2,p6],..., [p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
Pairs = { [1,2], [7,8]}

(d)

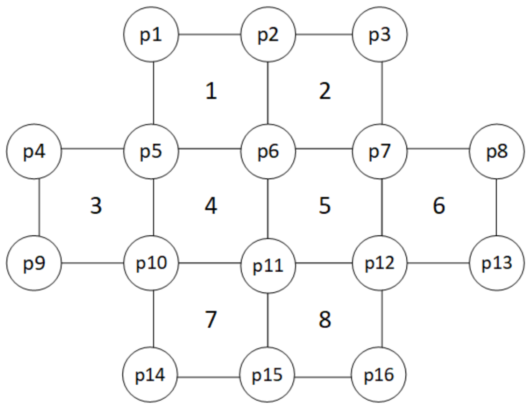
Figure 3.15: Execution steps of the GETADJACENTPAIRS algorithm (cont.)

Finally, FINDBOWTIES algorithm uses GETOPPOSITESIDENEIGHBOR algorithm in order to continue finding bow ties and inserting them into bow tie mesh structure, recursively. The algorithm GETOPPOSITESIDENEIGHBOR finds and returns neighbor quad of given quad and its already founded neighbor quad. In other words, this algorithm returns the neighbor standing the opposite side of given quad according to its neighbor quad. How this algorithm works is shown in Algorithm 3.3.5. Execution of the GETOPPOSITESIDENEIGHBOR algorithm is shown in Figure 3.16.

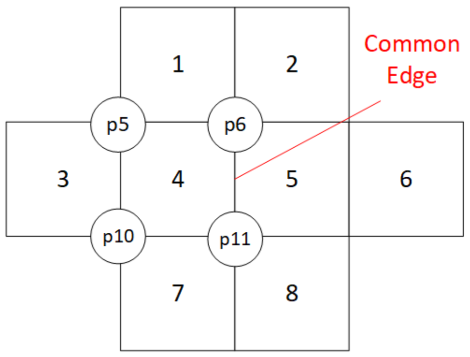
Besides, the algorithm GETOPPOSITESIDENEIGHBOR is also employed by SEARCH described in Algorithm 3.3.1. After finding bow ties process, SEARCH algorithm scans the quad mesh and detects the neighbor quads which does not participate in any bow tie of quads created a bow tie. When it finds such a quad, it gets this quad's neighbor quad which can create bow tie with this quad by using GETOPPOSITESIDENEIGHBOR algorithm.

Algorithm 3.3.5 GETOPPOSITESIDENEIGHBOR(QuadMesh, Quad1, Quad2)

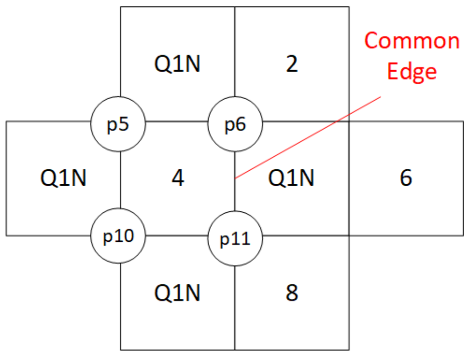
```
1: OppositeSideNeighbor = $null$
2: CommonEdge = common edge between Quad1 and Quad2
3: if CommonEdge exists then
4:     Quad1Neighbors = quads having a common edge with Quad1
5:     for each neighbor quad $NQ \in$ Quad1Neighbors do
6:         if $NQ \neq$ Quad2 $and\ !NQ.IsMatched$ then
7:             CommonEdgeWithNeighbor = common edge between Quad1 and $NQ$
8:             if CommonEdge and CommonEdgeWithNeighbor do not have any common point then
9:                 OppositeSideNeighbor = $NQ$
10:                break
11:            end if
12:        end if
13:    end for
14: end if
15: return OppositeSideNeighbor
```



```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
OppositeSideNeighbor = NULL
Quad1 = 4
Quad2 = 5
```



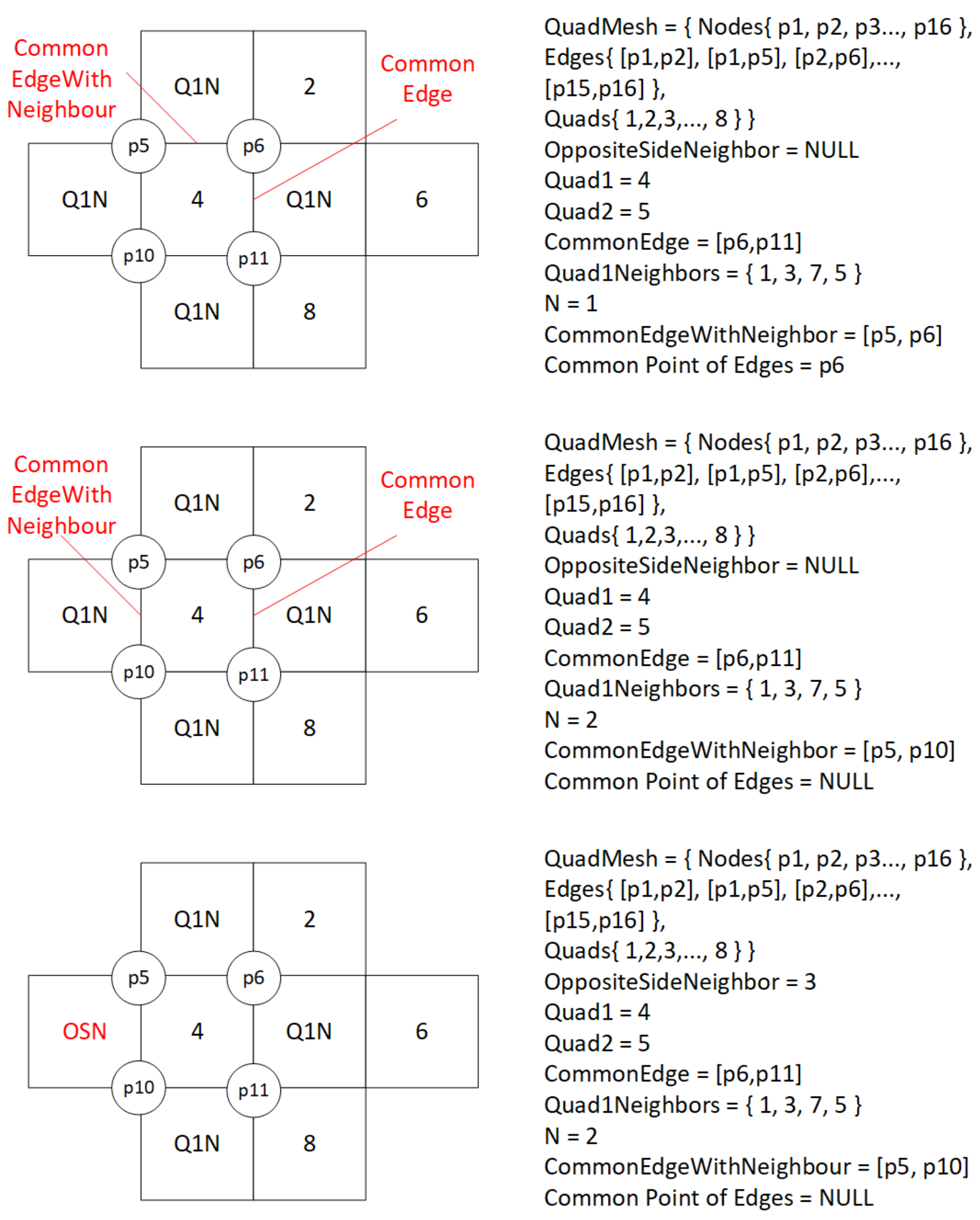
```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
OppositeSideNeighbor = NULL
Quad1 = 4
Quad2 = 5
CommonEdge = [p6,p11]
```



```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
OppositeSideNeighbor = NULL
Quad1 = 4
Quad2 = 5
CommonEdge = [p6,p11]
Quad1Neighbors = { 1, 3, 7, 5 }
```

(a)

Figure 3.16: Execution steps of the GETOPPOSITESIDENEIGHBOR algorithm



```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
OppositeSideNeighbor = NULL
Quad1 = 4
Quad2 = 5
CommonEdge = [p6,p11]
Quad1Neighbors = { 1, 3, 7, 5 }
N = 1
CommonEdgeWithNeighbor = [p5, p6]
Common Point of Edges = p6
```

```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
OppositeSideNeighbor = NULL
Quad1 = 4
Quad2 = 5
CommonEdge = [p6,p11]
Quad1Neighbors = { 1, 3, 7, 5 }
N = 2
CommonEdgeWithNeighbor = [p5, p10]
Common Point of Edges = NULL
```

```
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
OppositeSideNeighbor = 3
Quad1 = 4
Quad2 = 5
CommonEdge = [p6,p11]
Quad1Neighbors = { 1, 3, 7, 5 }
N = 2
CommonEdgeWithNeighbour = [p5, p10]
Common Point of Edges = NULL
```

(b)

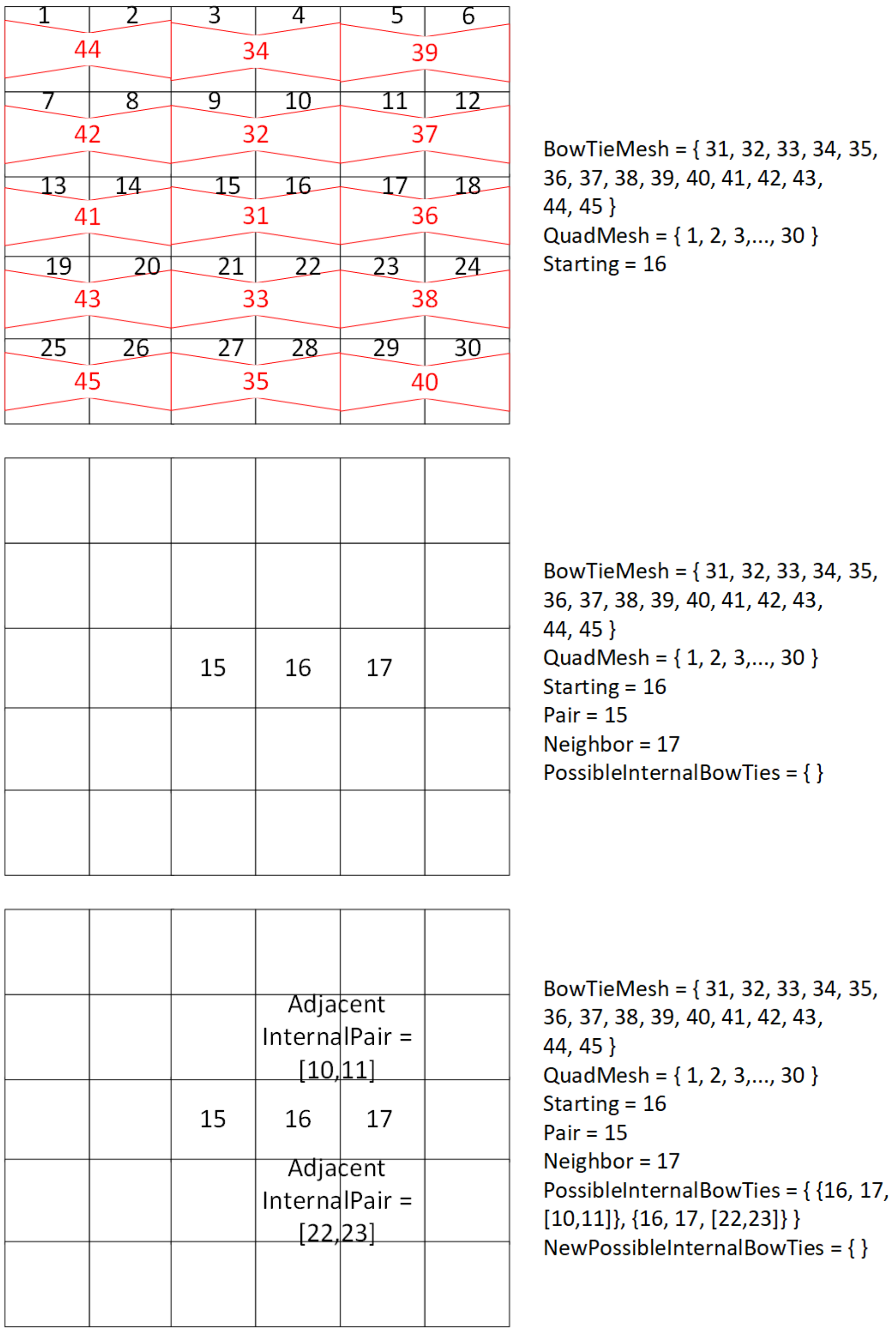
Figure 3.16: Execution steps of the GETOPPOSITESIDENEIGHBOR algorithm (cont.)

After the algorithm SEARCH described in Algorithm 3.3.1 completes the finding bow ties, then it starts a new finding operation to insert bow ties standing between founded bow ties, called internal bow ties. Internal bow ties cannot be inserted directly because founded bow ties generate them. If internal bow ties cannot be added into the bow tie mesh, gaps occur at the resulted bow tie mesh and these gaps cause disconnections. FINDINTERNALBOWTIES explained in Algorithm 3.3.6 is used for finding and inserting internal bow ties. This algorithm gets quad, checks bow ties around it for internal bow ties and inserts internal bow ties. After that, this algorithm jumps the neighbor quads of given quad and continues recursively. Execution of the FINDINTERNALBOWTIES algorithm is shown in Figure 3.17.

Algorithm 3.3.6 FINDINTERNALBOWTIES(BowTieMesh, QuadMesh, StartingQuad)

```
1:  if StartingQuad.IsMatched then
2:      PairQuad = StartingQuad.MatchedQuad
3:      NeighborQuad = GETOPPOSITESIDENEIGHBOR(QuadMesh, StartingQuad, PairQuad)
4:      if NeighborQuad $\neq null$ $and$ !NeighborQuad.IsInternalMatched $and$ !StartingQuad.IsInternalMatched
        then
5:          PossibleInternalBowTies = empty set of [Quad1, Quad2, QuadPair] creating possible inter-
            nal bow tie
6:          AdjacentInternalPairs = GETADJACENTINTERNALPAIRS(QuadMesh, StartingQuad, Neigh-
            bourQuad)
7:          for each adjacent internal pair $AIP \in$ AdjacentInternalPairs do
8:              Add [StartingQuad, NeighborQuad, $AIP$] to PossibleInternalBowTies
9:          end for
10:         while PossibleInternalBowTies $\neq empty$ do
11:             NewPossibleInternalBowTies = empty set of [quad, quad, pair of quads] creating possi-
                ble internal bow tie
12:             for each possible internal bow tie $PIB \in$ AdjacentInternalPairs do
13:                 INSERTINTERNALBOWTIE(BowTieMesh, QuadMesh, $PIB.Quad1$, $PIB.Quad2$,
                    $PIB.QuadPair$)
14:                 if bow tie was inserted from $PIB$ then
15:                     NewAdjacentInternalPairs = GETADJACENTINTERNALPAIRS(QuadMesh, $PIB.QuadPair.First$,
                        $PIB.QuadPair.Second$)
16:                     for each adjacent internal pair $AIP \in$ NewAdjacentInternalPairs do
17:                         Add [$PIB.QuadPair.First$, $PIB.QuadPair.Second$, $AIP$] to NewPossi-
```

```
                        bleInternalBowTies
                    **end for**
                **end if**
            **end for**PossibleInternalBowTies = NewPossibleInternalBowTies
        **end while**
        **if** NeighborQuad.IsMatched **then**
            NewStartingQuad = NeighborQuad.MatchedQuad
            FINDINTERNALBOWTIES(BowTieMesh, QuadMesh, NewStartingQuad)
        **end if**
        NewStartingQuad = PairQuad
        FINDINTERNALBOWTIES(BowTieMesh, QuadMesh, NewStartingQuad)
    **end if**
**end if**
```



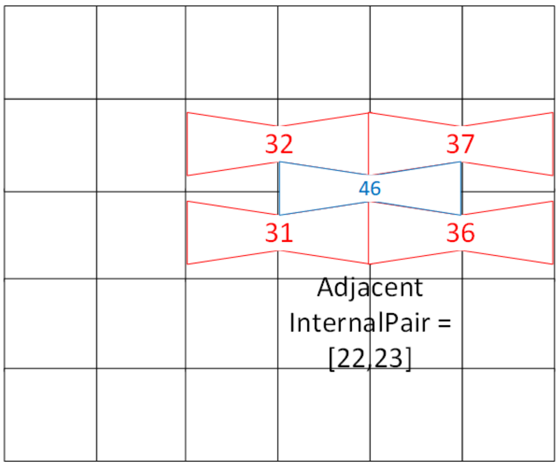
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16

BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { }

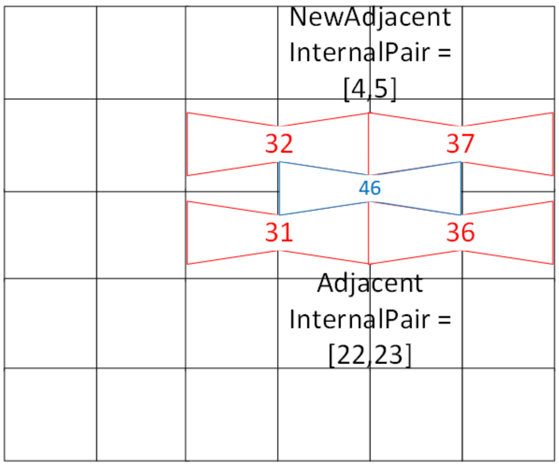
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {16, 17, [10,11]}, {16, 17, [22,23]} }
NewPossibleInternalBowTies = { }

(a)

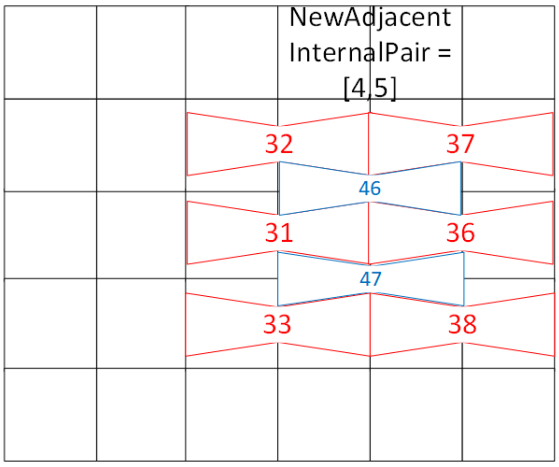
Figure 3.17: Execution steps of the FINDINTERNALBOWTIES algorithm



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {16, 17, [10,11]}, {16, 17, [22,23]} }
NewPossibleInternalBowTies = { }



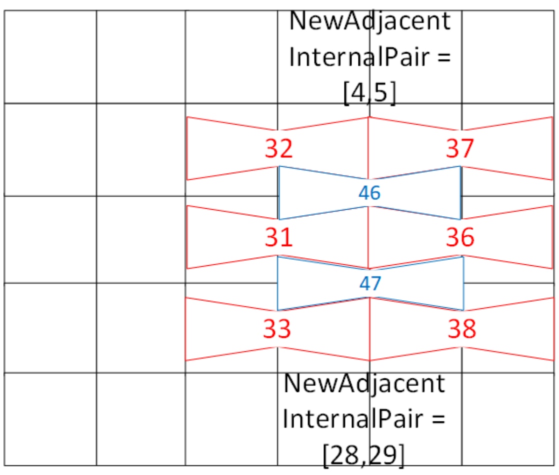
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {16, 17, [10,11]}, {16, 17, [22,23]} }
NewPossibleInternalBowTies = { {10, 11, [4,5]} }



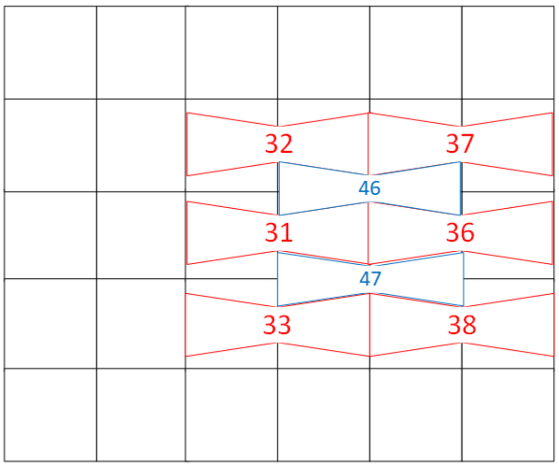
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {16, 17, [10,11]}, {16, 17, [22,23]} }
NewPossibleInternalBowTies = { {10, 11, [4,5]} }

(b)

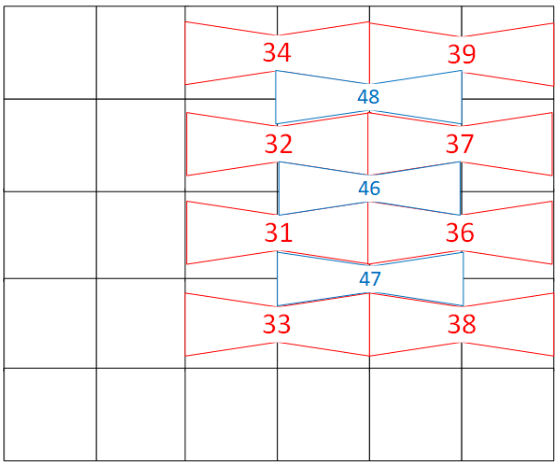
Figure 3.17: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)


NewAdjacent InternalPair = [4,5]
32
37
46
31
36
47
33
38
NewAdjacent InternalPair = [28,29]

BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {16, 17, [10,11]}, {16, 17, [22,23]} }
NewPossibleInternalBowTies = { {10, 11, [4,5]}, {22, 23, [28, 29]} }



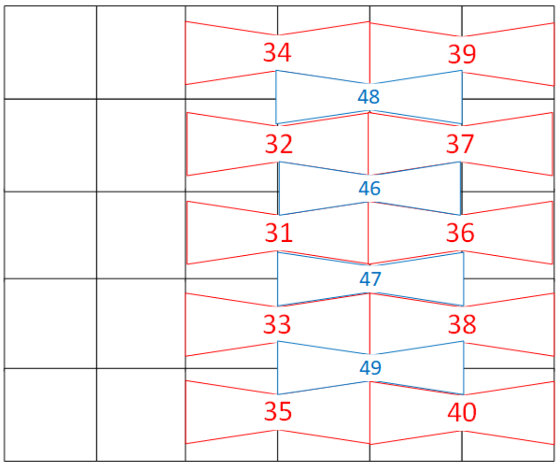
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {10, 11, [4,5]}, {22, 23, [28, 29]} }
NewPossibleInternalBowTies = { }



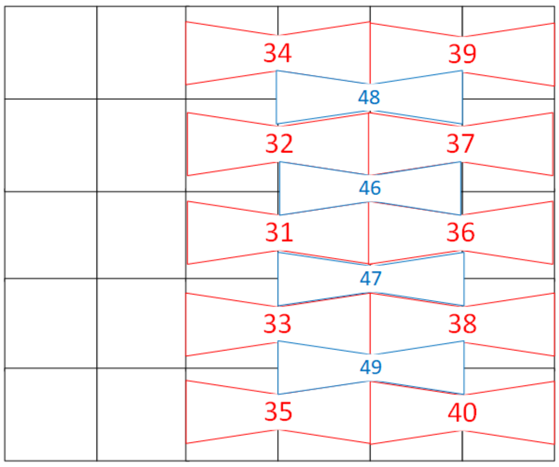
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {10, 11, [4,5]}, {22, 23, [28, 29]} }
NewPossibleInternalBowTies = { }

(c)

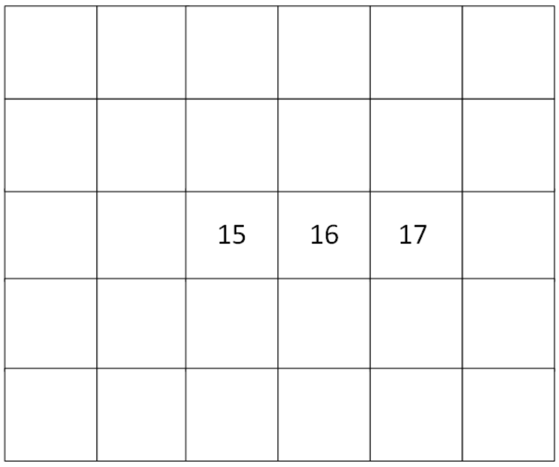
Figure 3.17: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {10, 11, [4,5]}, {22, 23, [28, 29]} }
NewPossibleInternalBowTies = { }



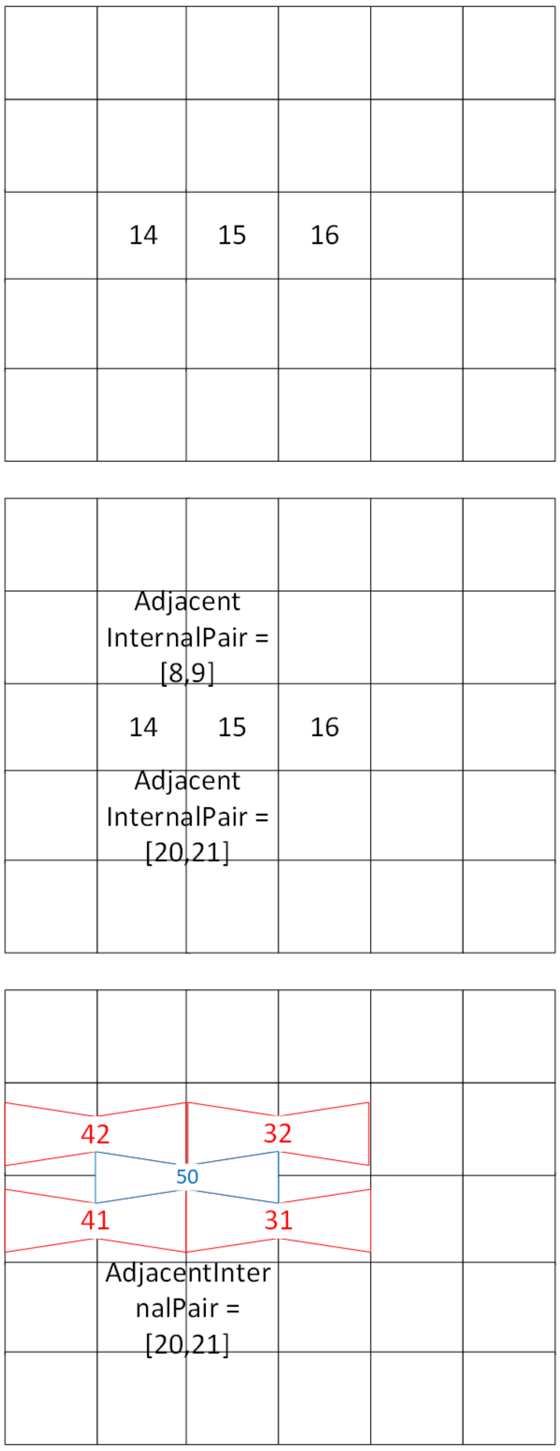
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { }



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { }
NewStarting = Pair

(d)

Figure 3.17: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)


14
15
16
Adjacent InternalPair = [8,9]
14
15
16
Adjacent InternalPair = [20,21]
42
32
50
41
31
AdjacentInternalPair = [20,21]

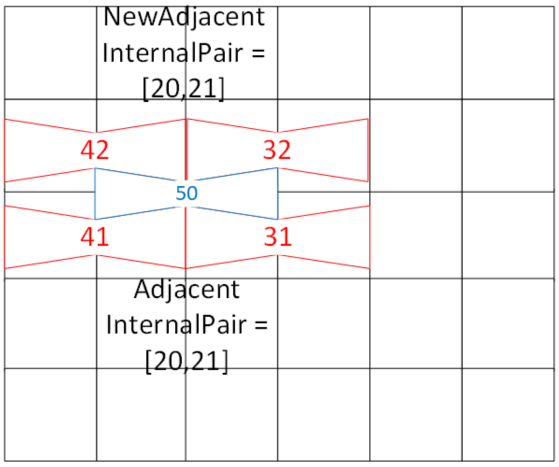
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { }

BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {15, 14, [8,9]}, {15, 14, [20,21]} }
NewPossibleInternalBowTies = { }

BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {15, 14, [8,9]}, {15, 14, [20,21]} }
NewPossibleInternalBowTies = { }

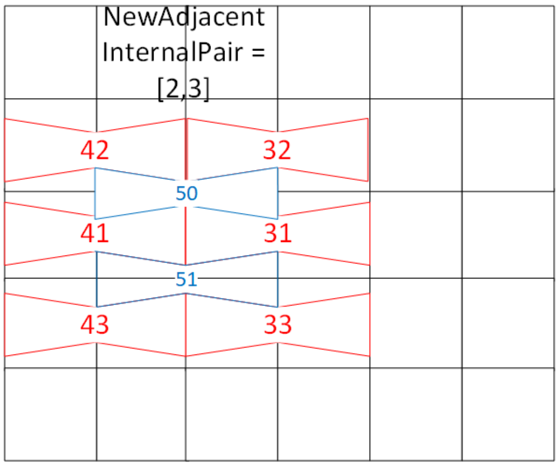
(e)

Figure 3.17: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)

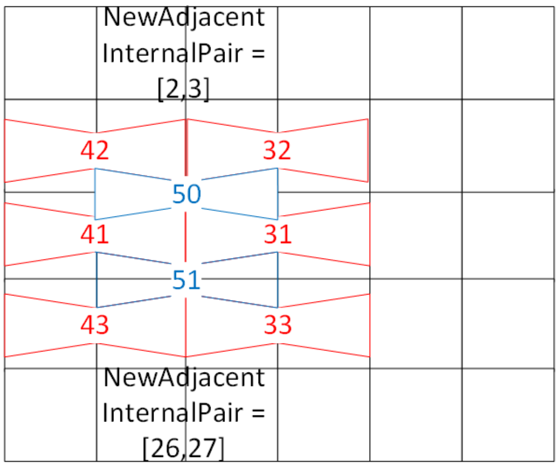


NewAdjacent InternalPair = [20,21]
42
32
50
41
31
Adjacent InternalPair = [20,21]

BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {15, 14, [8,9]}, {15, 14, [20,21]} }
NewPossibleInternalBowTies = { {8, 9, [20,21]} }



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {15, 14, [8,9]}, {15, 14, [20,21]} }
NewPossibleInternalBowTies = { {8, 9, [2,3]} }

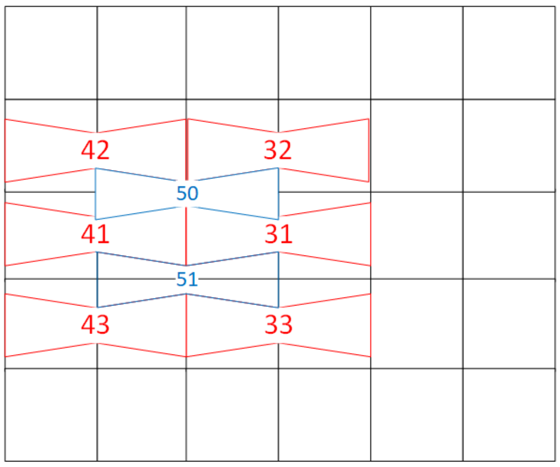


NewAdjacent InternalPair = [2,3]
42
32
50
41
31
51
43
33
NewAdjacent InternalPair = [26,27]

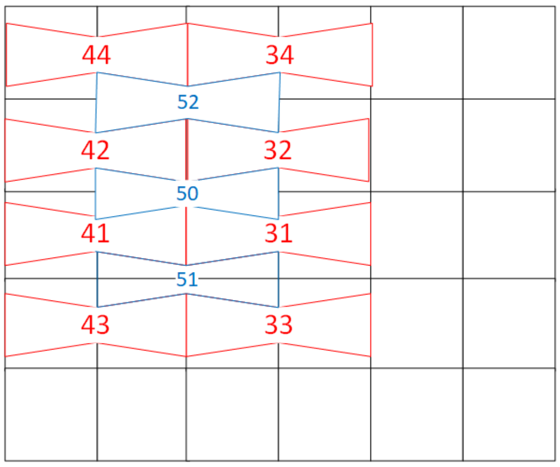
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {15, 14, [8,9]}, {15, 14, [20,21]} }
NewPossibleInternalBowTies = { {8, 9, [2,3]}, {20, 21, [26, 27]} }

(f)

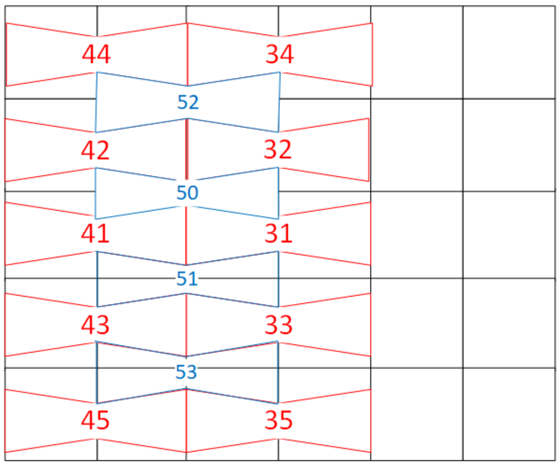
Figure 3.17: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {8, 9, [2,3]}, {20, 21, [26, 27]} }
NewPossibleInternalBowTies = { }



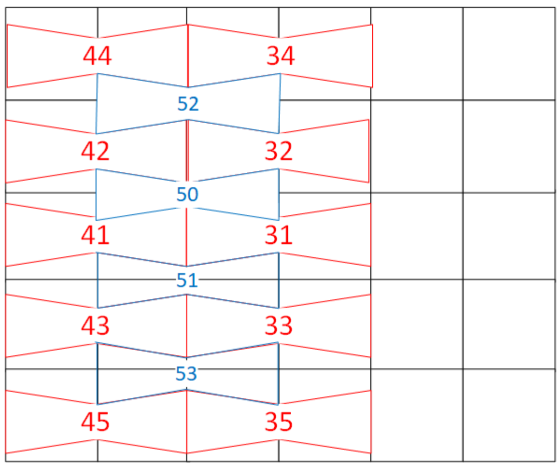
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {8, 9, [2,3]}, {20, 21, [26, 27]} }
NewPossibleInternalBowTies = { }



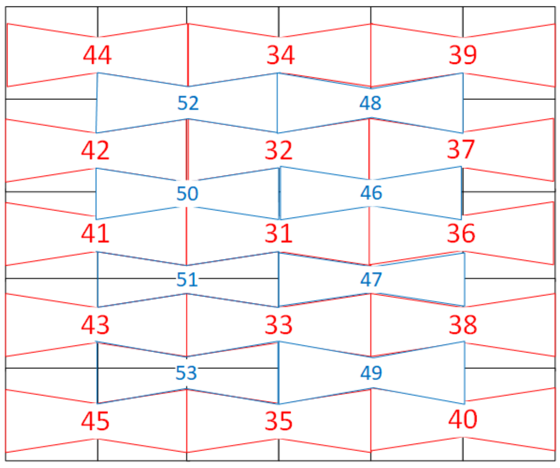
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {8, 9, [2,3]}, {20, 21, [26, 27]} }
NewPossibleInternalBowTies = { }

(g)

Figure 3.17: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { }



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { }
NewStarting = NULL

(h)

Figure 3.17: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)

Similar to algorithm FINDBOWTIES described in Algorithm 3.3.2, FINDINTERNALBOWTIES algorithm is also used GETOPPOSITESIDENEIGHBOR algorithm described in Algorithm 3.3.5 in order to find opposite side neighbor of the given quad.

The algorithm FINDINTERNALBOWTIES needs procedure getting adjacent internal pairs of quads of the given quad pair in order to create new internal bow ties. This procedure is very close the Algorithm 3.3.4 used by the Algorithm 3.3.2. Detail of this procedure is given in Algorithm 3.3.7. This algorithm, GETADJACENTINTERNALPAIRS, takes a pair of quads, checks their neighbors which created bow tie before and returns the suitable combination of them to create an internal bow tie with the given pair of quads. GETADJACENTINTERNALPAIRS algorithm is almost the same with the Algorithm 3.3.4 except being pair criteria therefore the same Figure 3.15 is valid for its execution.

Algorithm 3.3.7 GETADJACENTINTERNALPAIRS(QuadMesh, Quad1, Quad2)

```
1:  Pairs = empty set of pairs of quads
2:  CommonEdge = common edge between Quad1 and Quad2
3:  if CommonEdge exists then
4:      for each point $P \in$ CommonEdge do
5:          First = $null$
6:          Second = $null$
7:          Quad1Neighbors = quads having a common edge with Quad1
8:          for each neighbor quad $NQ \in$ Quad1Neighbors do
9:              if $NQ \neq$ Quad2 $and$ $P$ is a point of $NQ$ then
10:                 First = $NQ$
11:                 break
12:             end if
13:         end for
14:         Quad2Neighbors = quads having a common edge with Quad2
15:         for each neighbor quad $NQ \in$ Quad2Neighbors do
16:             if $NQ \neq$ Quad1 $and$ $P$ is a point of $NQ$ then
17:                 Second = $NQ$
18:                 break
19:             end if
20:         end for
21:         if First $\neq$ $null$ $and$ Second $\neq$ $null$ $and$ First $\neq$ Second $and$ !First.IsInternalMatched $and$
```

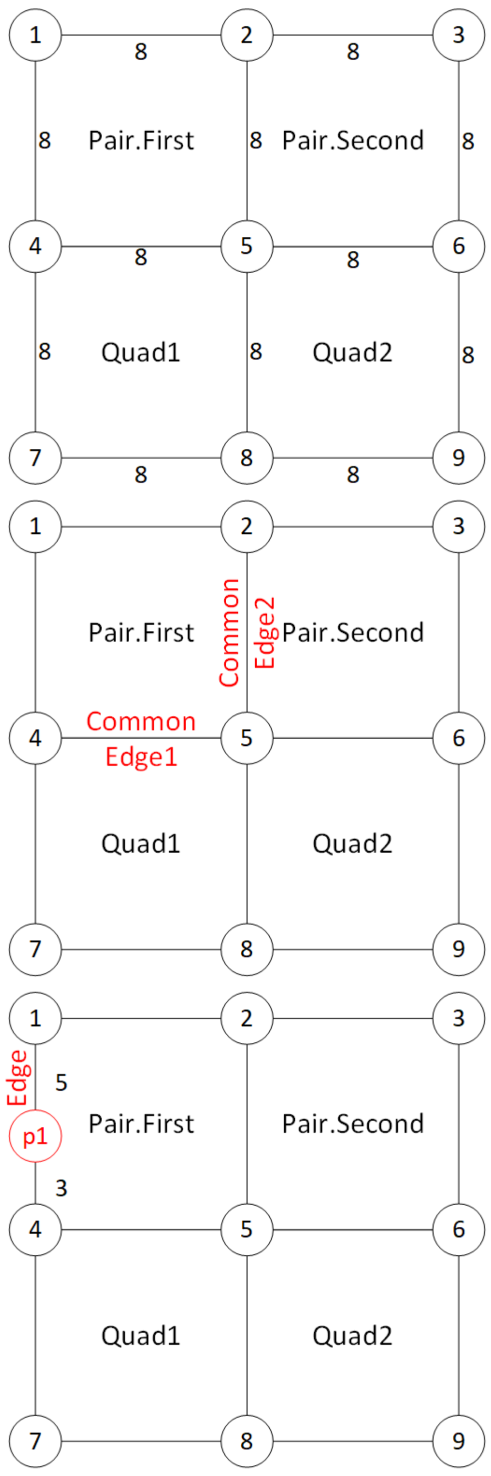
```
        !Second.IsInternalMatched $and$ First.IsMatched $and$ Second.IsMatched $and$ First.MatchedQuad
        $\neq$ Second $and$ Second.MatchedQuad $\neq$ First **then**
22:         Add [First, Second] to Pairs
23:       end if
24:     end for
25: end if
26: return Pairs
```

FINDINTERNALBOWTIES algorithm needs a different method from the INSERTBOWTIE described in Algorithm 3.3.3 to create and insert internal bow ties because bow ties consist of quads while internal bow ties consist of the other bow ties. This method is named INSERTINTERNALBOWTIE and described in Algorithm 3.3.8. INSERTINTERNALBOWTIE algorithm takes four quads which created bow tie before in doubles. This algorithm calculates coordinates of points creating the internal bow tie and orders these points to create related internal bow tie. Execution of this algorithm is shown in Figure 3.18.

Algorithm 3.3.8 INSERTINTERNALBOWTIE(BowTieMesh, QuadMesh, Quad1, Quad2, QuadPair)

```
 1: CommonEdge1 = common edge between Quad1 and QuadPair.First
 2: CommonEdge2 = common edge between QuadPair.First and QuadPair.Second
 3: if CommonEdge1 exists $and$ CommonEdge2 exists then
 4:     CommonPoint = common point of CommonEdge1, CommonEdge2
 5:     if CommonPoint exists then
 6:         OtherPoint1 = CommonEdge1's other point than CommonPoint
 7:         OtherPoint2 = CommonEdge2's other point than CommonPoint
 8:         Edge = QuadPair.First's edge containing OtherPoint1 and not containing CommonPoint
 9:         p1 = OtherPoint1 + (length of Edge) * 3/8
10:         Edge = QuadPair.First's edge containing CommonPoint and not containing OtherPoint1
11:         p2 = CommonPoint + (length of Edge) * 1/8
12:         Edge = Pair.Second's edge not containing CommonPoint and not containing OtherPoint2
13:         Point = Edge's point creating an edge with CommonPoint
14:         p3 = Point + (length of Edge) * 1/8
15:         Edge = Quad2's edge containing Point and not containing CommonPoint
16:         p4 = Point + (length of Edge) * 1/8
17:         Edge = Quad2's edge containing CommonPoint and not containing Point
```

```
18:         p5 = CommonPoint + (length of Edge) * 3/8
19:         Edge = Quad1's edge containing OtherPoint1 and not containing CommonPoint
20:         p6 = OtherPoint1 + (length of Edge) * 3/8
21:         Add points p1, p2, p3, p4, p5, p6 to BowTieMesh if they are not added before
22:         BowTie = empty set of ordered points creating bow tie
23:         Add ordered points p1, p2, p3, p4, p5, p6 to BowTie
24:         **if** Quad1's normal vector and BowTie's normal vector are in the opposite directions **then**
25:             Reverse points of BowTie
26:         **end if**
27:         Add BowTie to BowTieMesh
28:         Quad1.IsInternalMatched = $true$
29:         Quad2.IsInternalMatched = $true$
30:         QuadPair.First.IsInternalMatched = $true$
31:         QuadPair.Second.IsInternalMatched = $true$
32:     **end if**
33: **else**
34:     INSERTHALFBOWTIE(BowTieMesh, QuadMesh, Quad1, Quad2, QuadPair)
35: **end if**
```



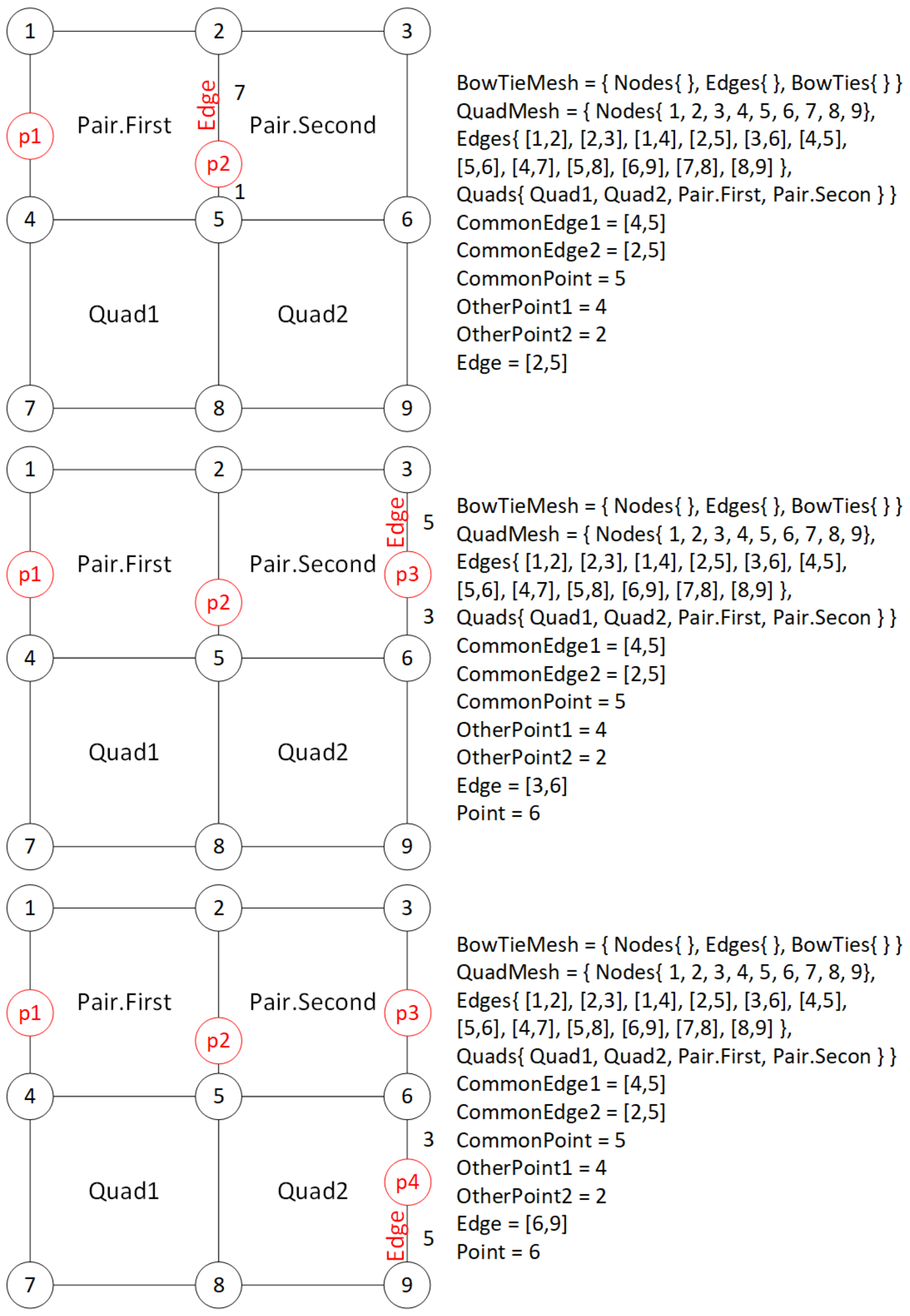
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9},
Edges{ [1,2], [2,3], [1,4], [2,5], [3,6], [4,5],
[5,6], [4,7], [5,8], [6,9], [7,8], [8,9] },
Quads{ Quad1, Quad2, Pair.First, Pair.Secon } }

BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9},
Edges{ [1,2], [2,3], [1,4], [2,5], [3,6], [4,5],
[5,6], [4,7], [5,8], [6,9], [7,8], [8,9] },
Quads{ Quad1, Quad2, Pair.First, Pair.Secon } }
CommonEdge1 = [4,5]
CommonEdge2 = [2,5]
CommonPoint = 5
OtherPoint1 = 4
OtherPoint2 = 2

BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9},
Edges{ [1,2], [2,3], [1,4], [2,5], [3,6], [4,5],
[5,6], [4,7], [5,8], [6,9], [7,8], [8,9] },
Quads{ Quad1, Quad2, Pair.First, Pair.Secon } }
CommonEdge1 = [4,5]
CommonEdge2 = [2,5]
CommonPoint = 5
OtherPoint1 = 4
OtherPoint2 = 2
Edge = [1,4]

(a)

Figure 3.18: Execution steps of the INSERTINTERNALBOWTIE algorithm



BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9},
Edges{ [1,2], [2,3], [1,4], [2,5], [3,6], [4,5],
[5,6], [4,7], [5,8], [6,9], [7,8], [8,9] },
Quads{ Quad1, Quad2, Pair.First, Pair.Secon } }
CommonEdge1 = [4,5]
CommonEdge2 = [2,5]
CommonPoint = 5
OtherPoint1 = 4
OtherPoint2 = 2
Edge = [2,5]

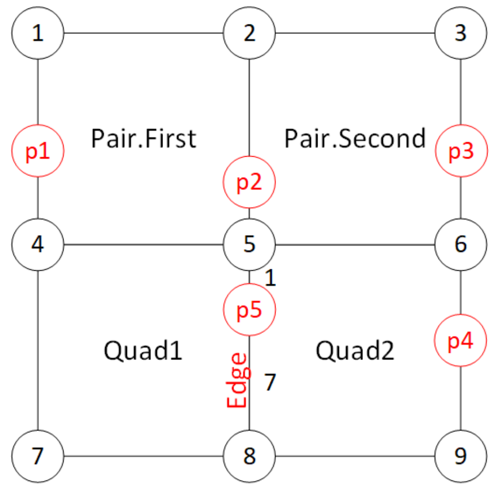
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9},
Edges{ [1,2], [2,3], [1,4], [2,5], [3,6], [4,5],
[5,6], [4,7], [5,8], [6,9], [7,8], [8,9] },
Quads{ Quad1, Quad2, Pair.First, Pair.Secon } }
CommonEdge1 = [4,5]
CommonEdge2 = [2,5]
CommonPoint = 5
OtherPoint1 = 4
OtherPoint2 = 2
Edge = [3,6]
Point = 6

BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9},
Edges{ [1,2], [2,3], [1,4], [2,5], [3,6], [4,5],
[5,6], [4,7], [5,8], [6,9], [7,8], [8,9] },
Quads{ Quad1, Quad2, Pair.First, Pair.Secon } }
CommonEdge1 = [4,5]
CommonEdge2 = [2,5]
CommonPoint = 5
OtherPoint1 = 4
OtherPoint2 = 2
Edge = [6,9]
Point = 6

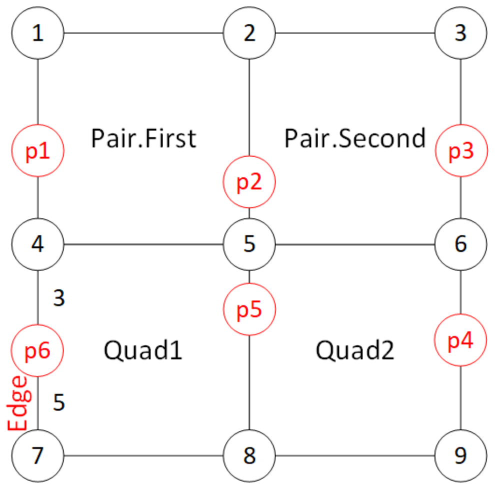
(b)

Figure 3.18: Execution steps of the INSERTINTERNALBOWTIE algorithm (cont.)

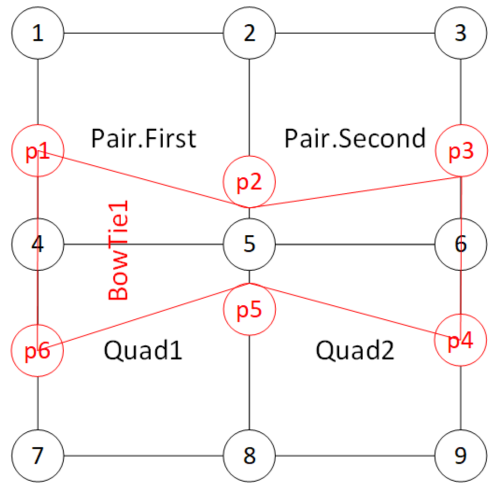
"



BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9},
Edges{ [1,2], [2,3], [1,4], [2,5], [3,6], [4,5],
[5,6], [4,7], [5,8], [6,9], [7,8], [8,9] },
Quads{ Quad1, Quad2, Pair.First, Pair.Secon } }
CommonEdge1 = [4,5]
CommonEdge2 = [2,5]
CommonPoint = 5
OtherPoint1 = 4
OtherPoint2 = 2
Edge = [5,8]
Point = 6



BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9},
Edges{ [1,2], [2,3], [1,4], [2,5], [3,6], [4,5],
[5,6], [4,7], [5,8], [6,9], [7,8], [8,9] },
Quads{ Quad1, Quad2, Pair.First, Pair.Secon } }
CommonEdge1 = [4,5]
CommonEdge2 = [2,5]
CommonPoint = 5
OtherPoint1 = 4
OtherPoint2 = 2
Edge = [4,7]
Point = 6



BowTieMesh = { Nodes{p1, p2, p3, p4, p5, p6 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p5], [p5,p6],
[p6,p1] },
BowTies{ BowTie1} }

QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9},
Edges{ [1,2], [2,3], [1,4], [2,5], [3,6], [4,5],
[5,6], [4,7], [5,8], [6,9], [7,8], [8,9] },
Quads{ Quad1, Quad2, Pair.First, Pair.Secon } }

(c)

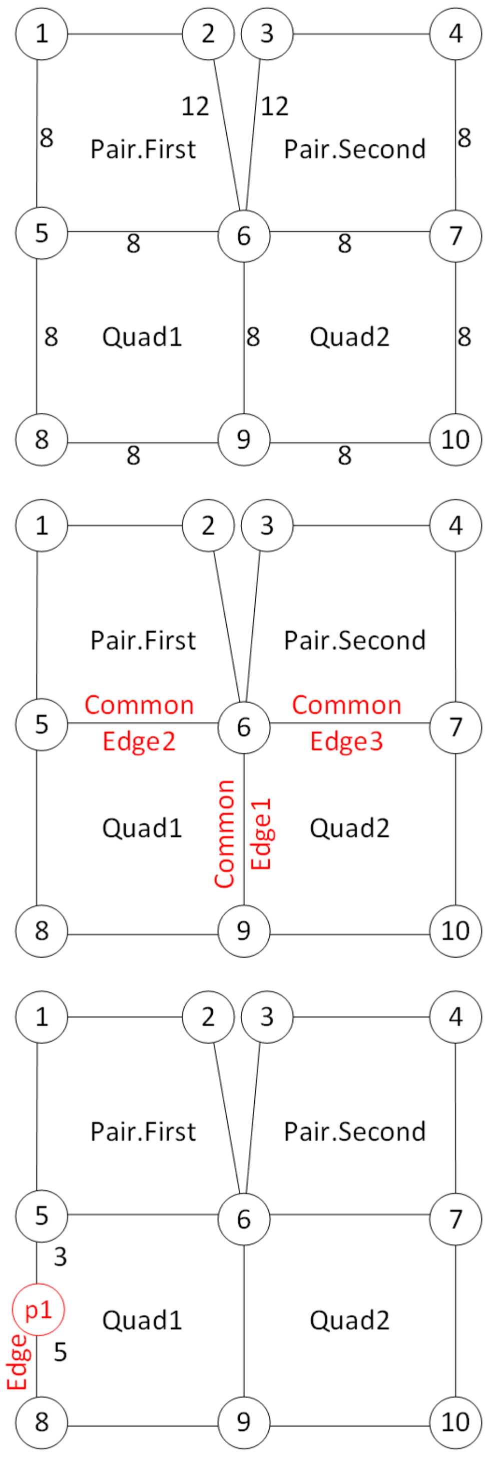
Figure 3.18: Execution steps of the INSERTINTERNALBOWTIE algorithm (cont.)

If the algorithm INSERTINTERNALBOWTIE cannot achieve to create complete bow tie due to lack of common edges, it applies to INSERTHALFBOWTIE described in Algorithm 3.3.9. This algorithm tries to create one or two half bow tie with the given quads and inserts what it founded in the bow tie mesh. Half bow ties are very important to keep resulted bow tie mesh together. Insertion steps of half bow ties are shown in Figure 3.19.

Algorithm 3.3.9 INSERTHALFBOWTIE(BowTieMesh, QuadMesh, Quad1, Quad2, QuadPair)

```
 1: CommonEdge1 = common edge between Quad1 and Quad2
 2: CommonEdge1 = common edge between Quad1 and QuadPair.First
 3: CommonEdge2 = common edge between Quad2 and QuadPair.Second
 4: if CommonEdge1 exists $and$ CommonEdge2 exists then
 5:     CommonPoint = common point of CommonEdge1, CommonEdge2
 6:     if CommonPoint exists then
 7:         OtherPoint = CommonEdge2's other point than CommonPoint
 8:         Edge = Quad1's edge containing OtherPoint and not containing CommonPoint
 9:         p1 = OtherPoint + (length of Edge) * 3/8
10:         Edge = Quad1's edge containing CommonPoint and not containing OtherPoint
11:         p2 = CommonPoint + (length of Edge) * 1/8
12:         Edge = Pair.First's edge containing CommonPoint and not containing OtherPoint
13:         p3 = CommonPoint + (length of Edge) * 1/8
14:         Edge = Pair.First's edge containing OtherPoint and not containin CommonPoint
15:         p4 = OtherPoint + (length of Edge) * 3/8
16:         Add points p1, p2, p3, p4 to BowTieMesh if they are not added before
17:         HalfBowTie = empty set of ordered points creating bow tie
18:         Add ordered points p1, p2, p3, p4 to HalfBowTie
19:         if Quad1's normal vector and HalfBowTie's normal vector are in the opposite directions
            then
20:             Reverse points of HalfBowTie
21:         end if
22:         Add HalfBowTie to BowTieMesh
23:     end if
24: end if
25: if CommonEdge1 exists $and$ CommonEdge3 exists then
26:     CommonPoint = common point of CommonEdge1, CommonEdge3
27:     if CommonPoint exists then
```

```
28:         OtherPoint = CommonEdge3's other point than CommonPoint
29:         Edge = Quad2's edge containing OtherPoint and not containing CommonPoint
30:         p5 = OtherPoint + (length of Edge) * 3/8
31:         Edge = Quad2's edge containing CommonPoint and not containing OtherPoint
32:         p6 = CommonPoint + (length of Edge) * 1/8
33:         Edge = Pair.Seconds's edge containing CommonPoint and not containing OtherPoint
34:         p7 = CommonPoint + (length of Edge) * 1/8
35:         Edge = Pair.Seconds's edge containing OtherPoint and not containin CommonPoint
36:         p8 = OtherPoint + (length of Edge) * 3/8
37:         Add points p5, p6, p7, p8 to BowTieMesh if they are not added before
38:         HalfBowTie = empty set of ordered points creating bow tie
39:         Add ordered points p5, p6, p7, p8 to HalfBowTie
40:         if Quad2's normal vector and HalfBowTie's normal vector are in the opposite directions
            then
41:             Reverse points of HalfBowTie
42:         end if
43:         Add HalfBowTie to BowTieMesh
44:     end if
45: end if
```



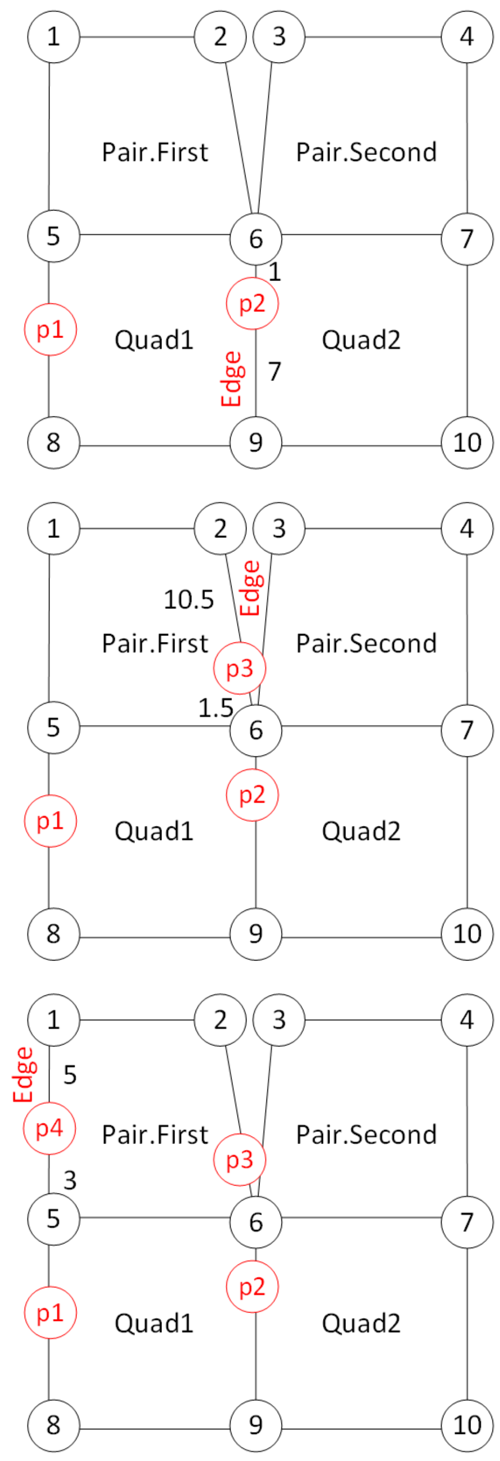
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }

BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 5

BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 5
Edge = [5,8]

(a)

Figure 3.19: Execution steps of the INSERTHALFBOWTIE algorithm



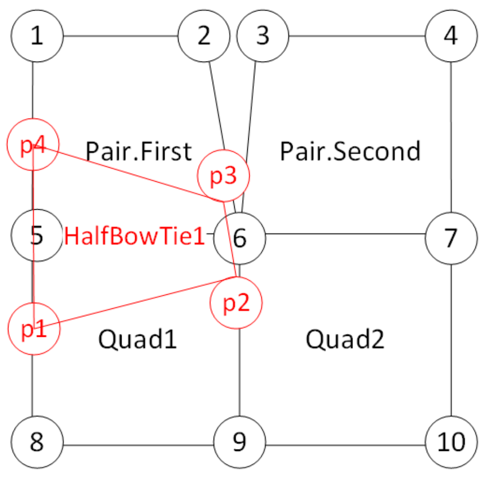
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 5
Edge = [6,9]

BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 5
Edge = [2,6]

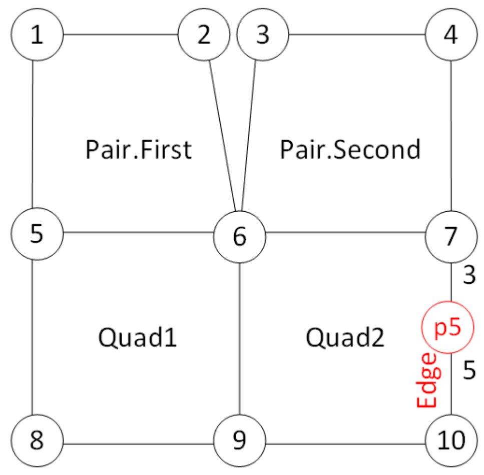
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 5
Edge = [1,5]

(b)

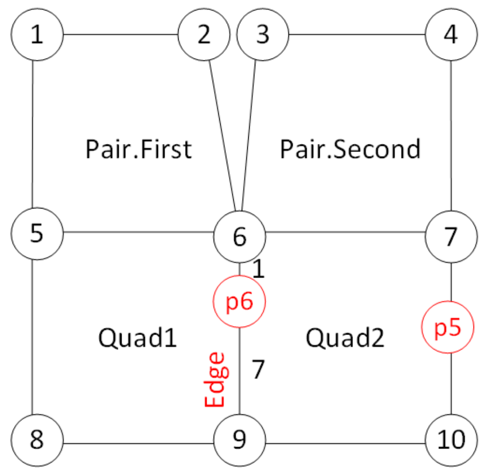
Figure 3.19: Execution steps of the INSERTHALFBOWTIE algorithm (cont.)



BowTieMesh = { Nodes{ p1, p2, p3, p4 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p1] },
BowTies{ HalfBowTie1 } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 5



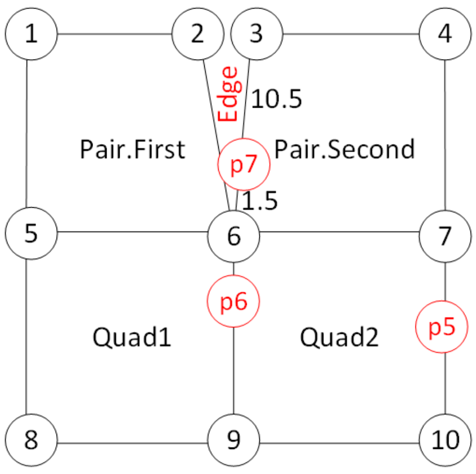
BowTieMesh = { Nodes{ p1, p2, p3, p4 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p1] },
BowTies{ HalfBowTie1 } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 7
Edge = [7,10]



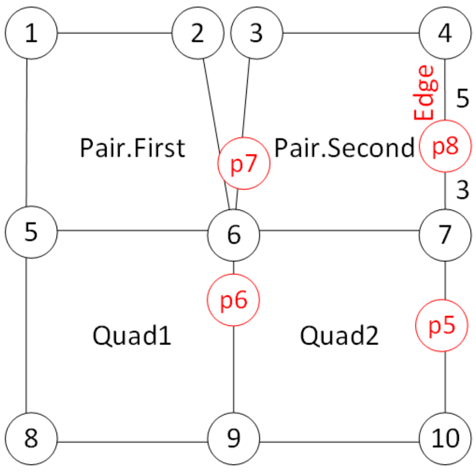
BowTieMesh = { Nodes{ p1, p2, p3, p4 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p1] },
BowTies{ HalfBowTie1 } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 7
Edge = [6,9]

(c)

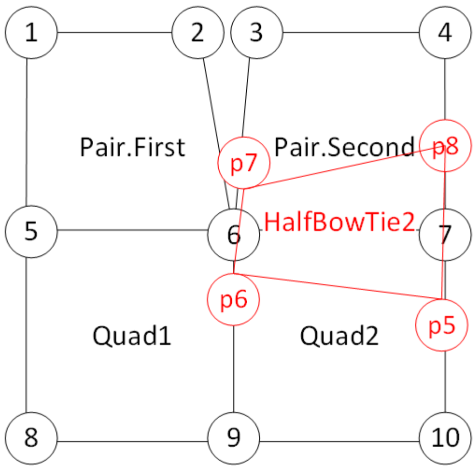
Figure 3.19: Execution steps of the INSERTHALFBOWTIE algorithm (cont.)



BowTieMesh = { Nodes{ p1, p2, p3, p4 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p1] },
BowTies{ HalfBowTie1 } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 7
Edge = [3,6]



BowTieMesh = { Nodes{ p1, p2, p3, p4 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p1] },
BowTies{ HalfBowTie1 } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 7
Edge = [4,7]



BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6,
p7, p8 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p1],
[p5,p6], [p6,p7], [p7,p8], [p8,p4], },
BowTies{ HalfBowTie1, HalfBowTie2 } }

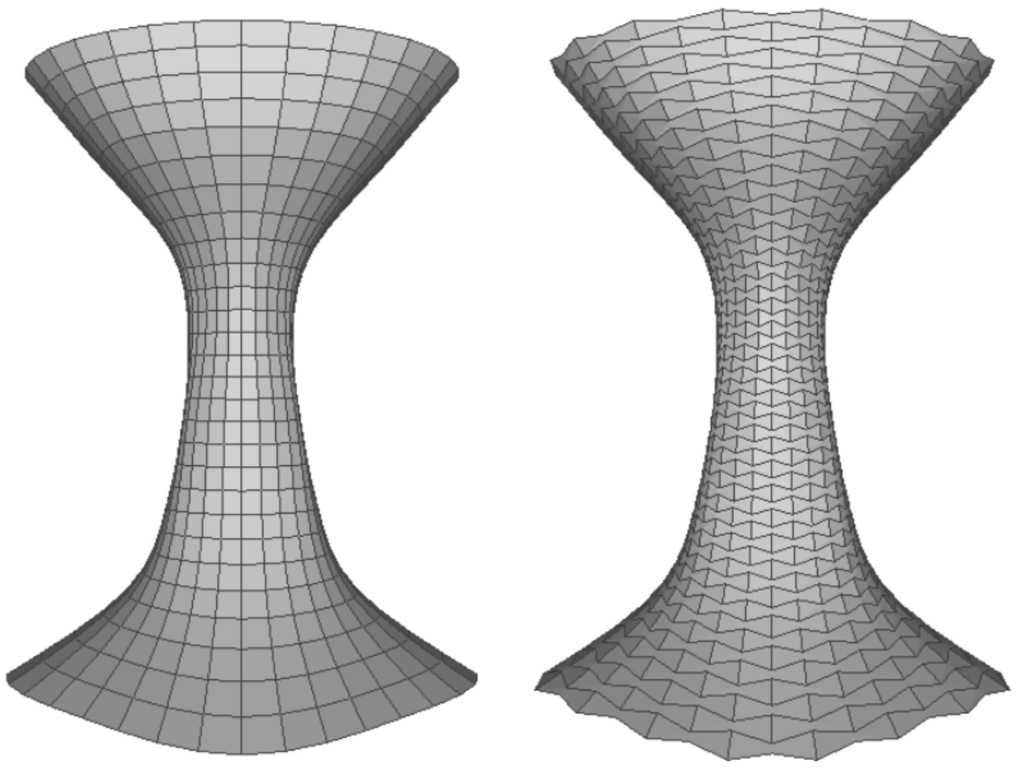
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }

(d)

Figure 3.19: Execution steps of the INSERTHALFBOWTIE algorithm (cont.)

3.4 Preparing Extracted Auxetic Patterns For 3D Printing

In Figure 3.20a, an example quad mesh goblet digital model which is the input of auxetic pattern extraction algorithm described in Section 3.3. After the algorithm works on it, obtained bow tie mesh is shown in Figure 3.20b. Quads were turned into hexagons.



(a) Quad mesh

(b) Bow tie mesh

Figure 3.20: Goblet model

After auxetic patterns were extracted, faces must be removed and remaining edges must be thickened for 3D printing. At this stage, we get help from blender [30], an open source 3D graphics designing software. This software presents implementations that can be used for modifying meshes. We use blender's wireframe and skin modifiers in order to get rid of faces created by hexagons and thicken the edges of hexagons. The bow tie mesh in Figure 3.20b was processed in blender and desired structure which is ready for 3D printing was acquired. Extracted re-entrant honeycomb patterns without faces and with thickened edges are shown in Figure 3.21.

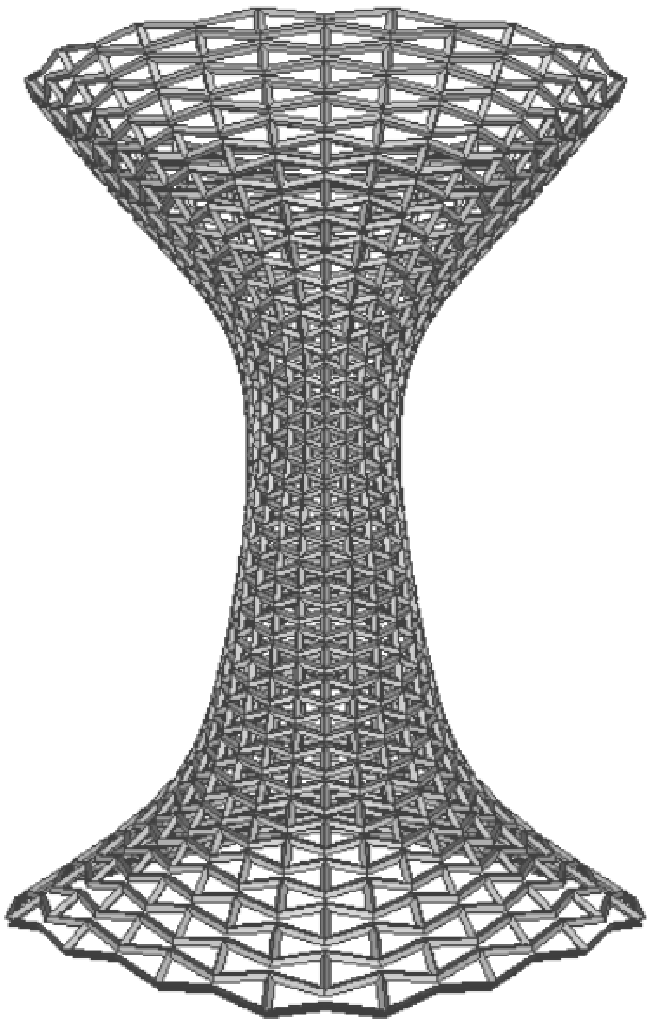


Figure 3.21: Extracted auxetic patterns prepared for 3D printing

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Advantages Of Extracting Auxetic Patterns For 3D Printing

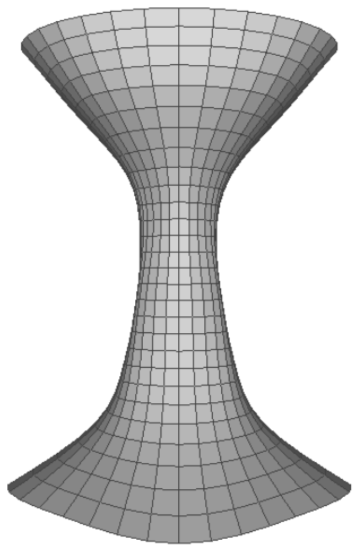
Extracting auxetic patterns from digital designs and 3D printing them instead of the original digital designs provides some benefits. These benefits are addressed in this section.

In Figure 4.1a the digital design of a goblet model is shown. The digital design is a quad mesh. The goblet model does not have bottom and top bases; therefore, this model is not a closed structure. Because of being not a closed structure, the goblet is hollow. In Figure 4.1b auxetic pattern extracted from the goblet model is shown. This auxetic pattern structure is a wireframe bow tie mesh.

In Figure 4.2a the digital design of a vase model is shown. The digital design is quad mesh. The vase model has only bottom base; therefore, this model is a half closed structure. The vase is also hollow. In Figure 4.2b auxetic pattern extracted from the vase model is shown. This auxetic pattern structure is a wireframe bow tie mesh.

In Figure 4.3a the digital design of a glove model is shown. The digital design is quad mesh. The glove model is complete closed structure. Despite being completely closed, inside of the glove is empty. Therefore, the glove is also hollow. In Figure 4.3b auxetic pattern extracted from the vase model is shown. This auxetic pattern structure is a wireframe bow tie mesh.

When we want to 3D print both the original digital models and extracted auxetic patterns from them shown in Figures 4.1, 4.2, 4.3 the numbers about 3D printing processes are shown in table 4.1. These numbers obtained from the 3D printing software,


(a) Original digital design

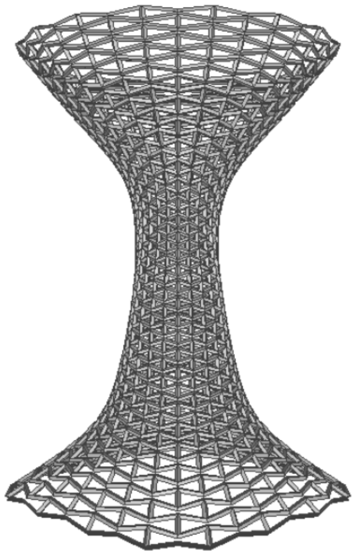
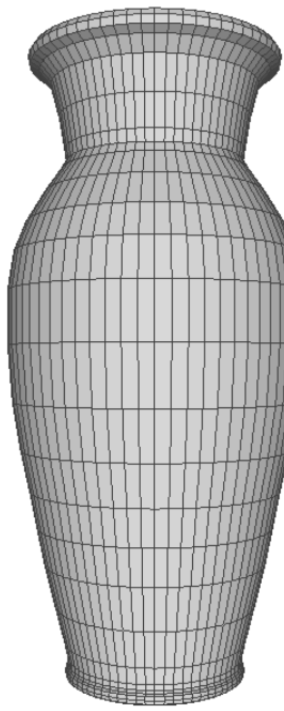

(b) Extracted auxetic pattern

Figure 4.1: Goblet model


(a) Original digital design

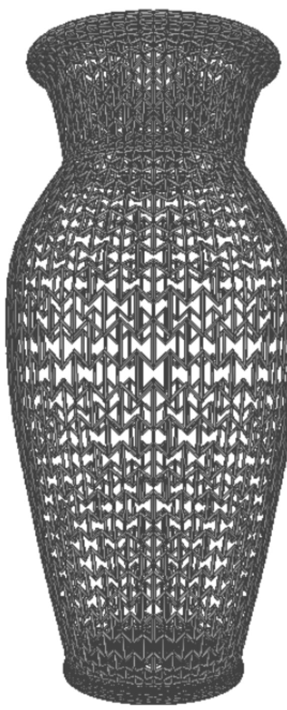

(b) Extracted auxetic pattern

Figure 4.2: Vase model

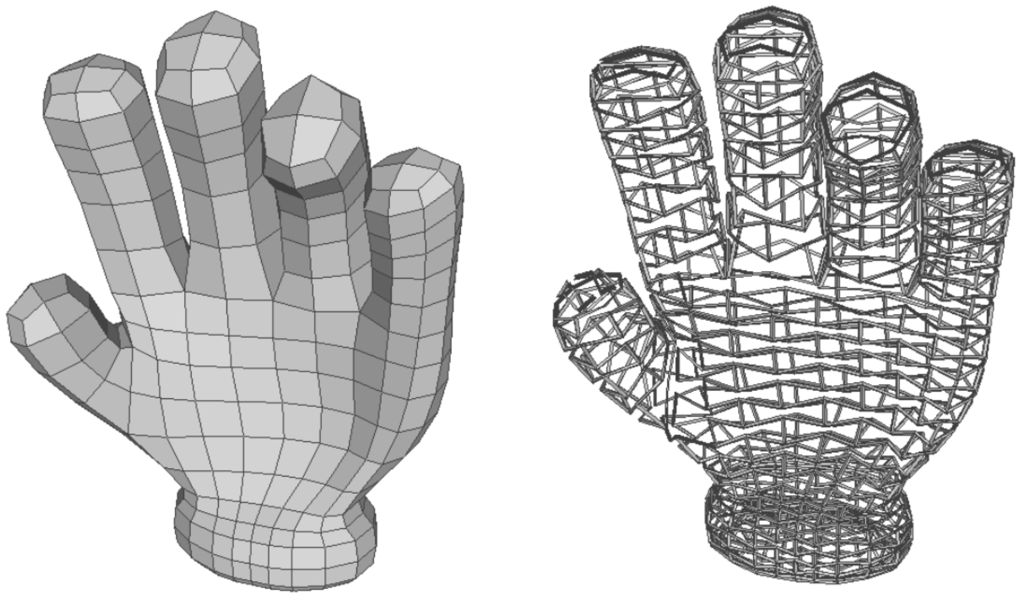

(a) Original digital design (b) Extracted auxetic pattern

Figure 4.3: Glove model

Cura [31] for 3D printer Ultimaker 2+. In all cases, original models and extracted auxetic patterns have the same dimensions. Identical parameters(material, nozzle, layer height, print speed) are used for both the original models and extracted auxetic patterns. Besides, supports are not added and infill is not enabled in the 3D printing software in any case for fair comparison.

Table 4.1: 3D Printing Requirements Of Different Models

Target Model	**Required Time (minute)**	**Required Material Quantity (meter / gram)**
Goblet	372	3.05 / 24
Extracted Auxetic Patterns From Goblet	156	0.63 / 4
Vase	529	5.28 / 41
Extracted Auxetic Patterns From Vase	303	0.66 / 5
Glove	947	8.67 / 68
Extracted Auxetic Patterns From Glove	360	1.05 / 8

Printing the extracted auxetic patterns reduces the time spent for 3D printing. Reduced 3D printing time means that 3D printer works less and energy consumption of it also reduces. Therefore, printing cost also becomes cheaper. Additionally, printing the extracted auxetic patterns substantially reduces the material usage which constitutes the essential cost of 3D printing. Wang et al. [24] point out that fabricating wireframe meshes of objects is a cost-effective method in 3D printing world because of the low material consumption.

Another benefit provided from reduced 3D printing time is quickly validating the digital designs. Wireframe bow tie meshes geometrically approximate to original meshes. In other words, wireframe bow ties replace the surface of the original meshes. 3D printing the wireframe bow tie meshes requires shorter time; however, obtained 3D objects resemble the 3D objects printed from the original meshes. Fast 3D printing enables the designers quickly review their designs' appearances. Mueller et al. [1] also emphasize the importance of fast prototyping for designers.

4.2 Challenges Of 3D Printing Extracted Auxetic Patterns

Although 3D printing of extracted auxetic patterns from digital models provides some benefits described in Section 4.1, it is not easy to print them. Challenges of 3D printing extracted auxetic patterns and what can be done against them are addressed in this section.

4.2.1 Increasing Weigth Challenge

Original digital models have solid structures. There are no gaps on their surfaces shown as Figures 4.1a, 4.2a, 4.3a. 3D printing surfaces without gaps is not so hard. While 3D printing, upper layers lean on bottom layers thus, the stability of the upper layers guaranteed by bottom layers because there is not any gap between these layers.

However, auxetic patterns extracted from digital models have gaps on their surfaces shown as Figures 4.1b, 4.2b, 4.3b. Because of existing gaps on the surfaces, extracted auxetic patterns are not as durable as the original digital models. Thus, 3D printing extracted auxetic patterns having gaps on their surfaces is harder than 3D printing digital models without having gaps on their surfaces. Bottom layers of extracted auxetic patterns are as not solid as original digital models. Therefore, sometimes bottom layers cannot resist increasing weight of upper layers. This situation causes downfalls during 3D printing operation. Such a case is encountered during 3D printing of extracted auxetic patterns shown in Figure 4.3b. This case is shown in Figure 4.4.

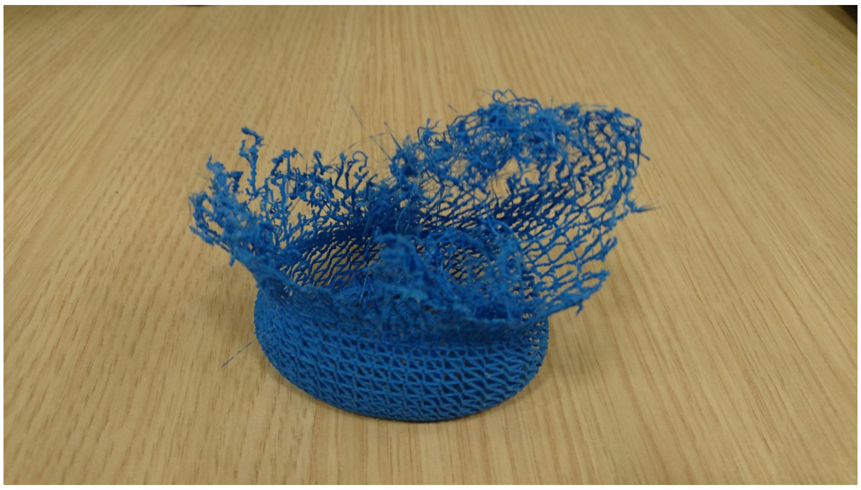


Figure 4.4: Downfall happened while 3D printing with Ultimaker 2+

If any downfall happens while 3D printing, the operation must be canceled immediately because 3D printers cannot realize downfalls. Because of collapsed bottom layers, they print upper layers on the air. They continue to print as if nothing has happened and that means wasting of time and material.

Although it is harder to 3D print extracted auxetic patterns, it is not impossible. If the structure (surface slope, balance) of the digital model is proper and thickness is sufficient, auxetic patterns extracted from the digital model can be 3D printed. Succeeded 3D printing of extracted auxetic patterns shown in Figure 4.1b can be seen in Figure 4.5.

We also tried different 3D printer which is also based on different technology, named DLP. DLP 3D printers can 3D print the entire layer at once. Layers for DLP 3D printers are images. In each image, parts desired to be 3D printed are white pixels and rest of the pixels are black. DLP 3D printers locate an image under the liquid material and flash the image. Black pixels in the image block the light. Light only gets through white pixels and solidates the liquid material at desired parts. DLP is much better for 3D printing intricate shapes like our extracted auxetic patterns.

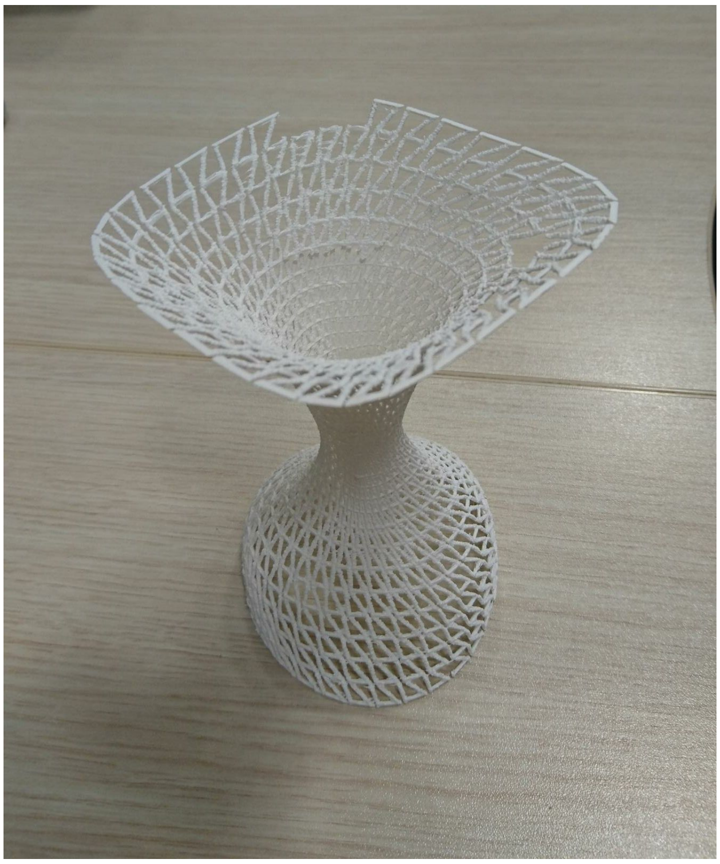


Figure 4.5: Successfully 3D printed auxetic patterns with Ultimaker 2+

We separated extracted auxetic patterns show in Figure 4.1b into layers. Each layer is consisting of points belonging to edges of extracted auxetic patterns and when these layers are added up target shape is formed. These layers can be seen in Figure 4.6. Then, we created related images from layers. Example images are shown in Figure 4.7. We fed the DLP 3D printer with these images and DLP 3D printer created the shape shown in Figure 4.8.

However, increasing weight problem is also valid for DLP 3D printers in a different way. Top layers, because of their increasing weight, pulls bottom layers down during the 3D printing process. Therefore, sometimes bottom layers cannot resist gravitational force causing the weight of top layers and connections between bottom and top layers can be broken.

Mueller et al. [1] suggest a method, called WirePrint, which is more appropriate to 3D print our extracted auxetic patterns. They developed a software which arranges the 3D

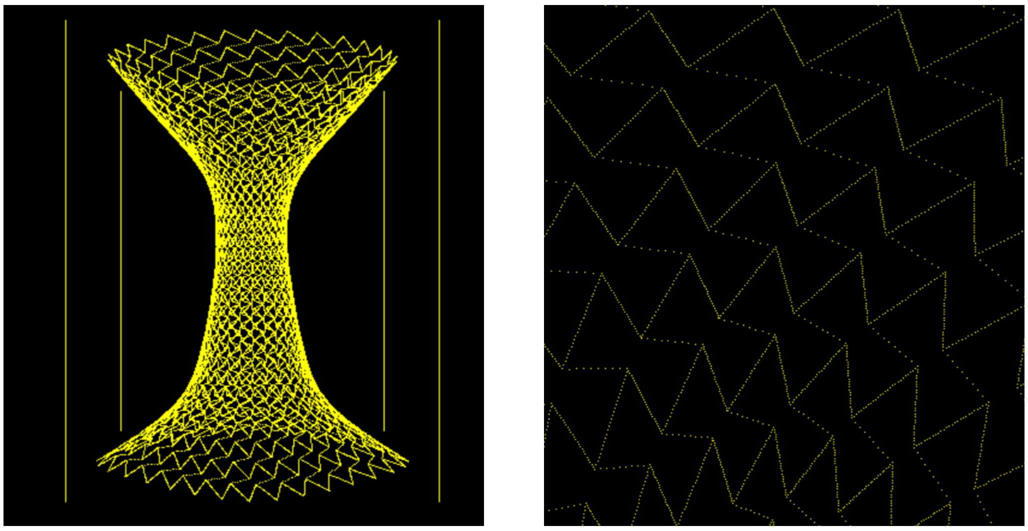
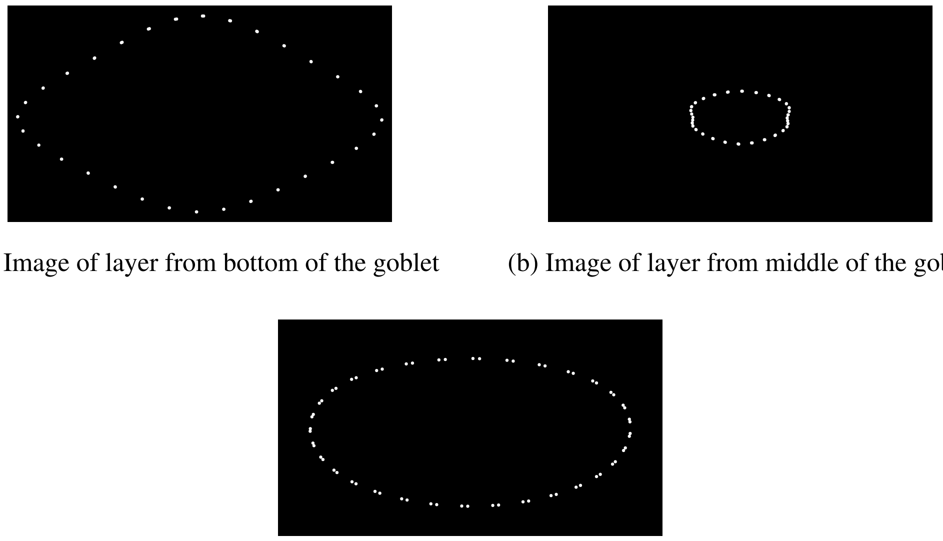

(a) All layers creating the goblet (b) Zoomed layers

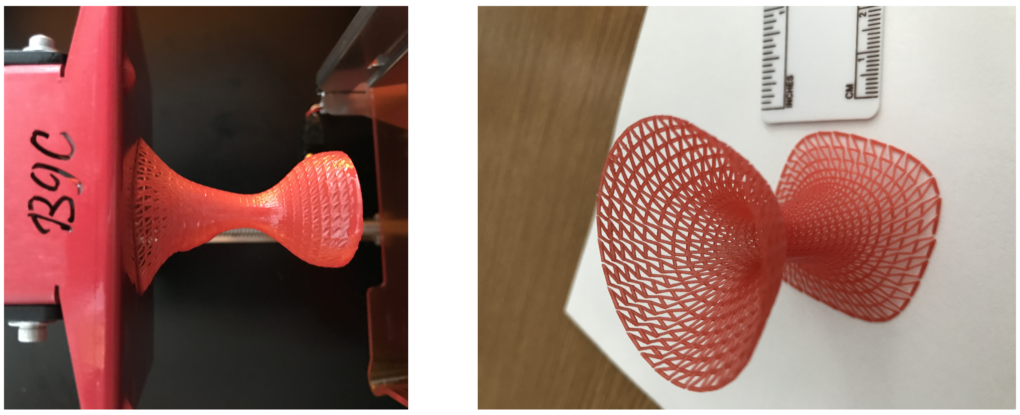
Figure 4.6: Layered goblet for DLP 3D printing


(a) Image of layer from bottom of the goblet (b) Image of layer from middle of the goblet

(c) Image of layer from top of the goblet

Figure 4.7: Images of goblet layers

printers' print head moves for WirePrint. In other words, they transform traditional 3D printers to print not layer by layer but edge by edge. However, arranging print head moves is not enough, they also use extra cooling mechanisms integrated with the print head to strengthen edges during the edge by edge print process. Their printer cools



(a) (b)

Figure 4.8: 3D printed goblet with B9 Creator 1.2

the edges immediately after they printed. Therefore, printed edges are more strong and their resistance to increasing weight of upper layers is better. Their modified printer and printed object with this printer are shown in Figure 4.9. If we have the opportunity of using such a 3D printer, we can 3D print our extracted auxetic patterns easily.

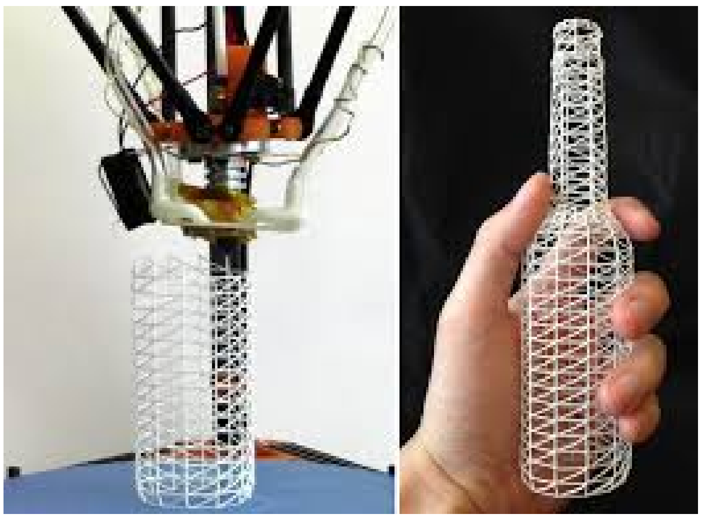
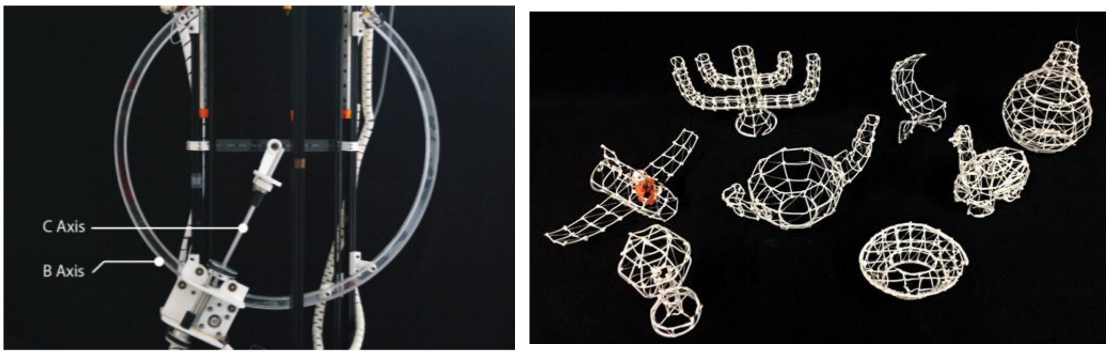


Figure 4.9: WirePrint [1]

Additionally, there is another modified 3D printer to wireframe printing developed by Peng et al. [2], 5DOF Wireframe Printer. This 3D printing technique improved on the method designed by Mueller et al. [1]. 5DOF Wireframe Printer has a rotating platform additional to print head with extra cooling mechanisms. The platform rotates already printed part of the object during 3D print operation in order to print new edges smoothly. When the weight on the bottom layers increased, already printed bottom layers can be rotated and printing can go on horizontally with the 5DOF WireFrame Printer. There could be collisions between already printed parts and print head moves; fortunately, this problem is solved before by Wu et al. [26]. 5 DOF WireFrame Printer and its products are shown in Figure 4.10. 5 DOF WireFrame Printer seems to be great for 3D printing our extracted auxetic patterns.



(a) (b)

Figure 4.10: 5 DOF WireFrame Printer and its products [2]

4.2.2 3D Printing Flattened Auxetic Patterns

After we saw difficulties of 3D printing wireframe meshes, our extracted auxetic patterns, with ordinary 3D printers; we come up with the idea that 3D printing flattened auxetic patterns. It is not hard to 3D print flattened shapes because they have almost two dimensions. Our 3D printed flattened auxetic patterns are shown in Figures 4.11 and 4.12.

We apply 3D mesh parametrization methods in order to transform 3D extracted auxetic patterns to 2D flattened auxetic patterns. However, after applying parametrization operations to 3D extracted auxetic patterns, they would turn into 2D surfaces by definition of the parametrization. Therefore, we decide to apply parametrization methods to 3D meshes, turn them into 2D meshes and extract the auxetic patterns on

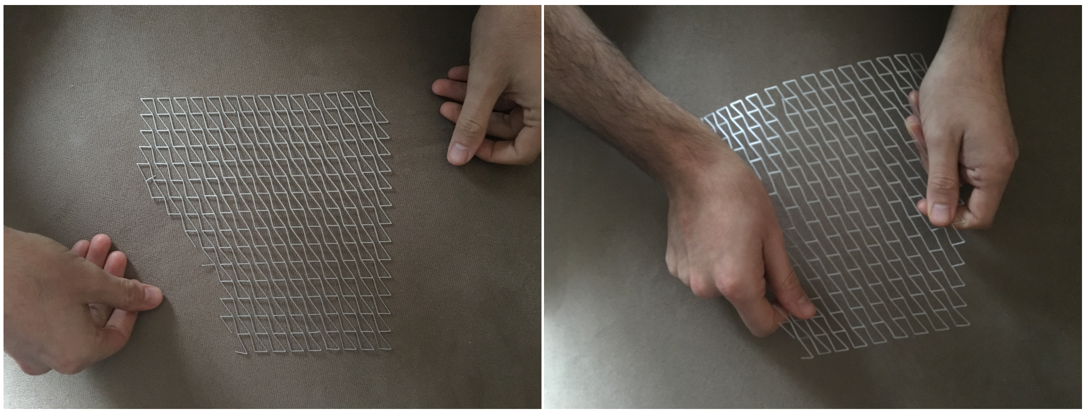

(a) Normal (b) Stretched

Figure 4.11: 3D printed flattened large auxetic patterns with Ultimaker 2+

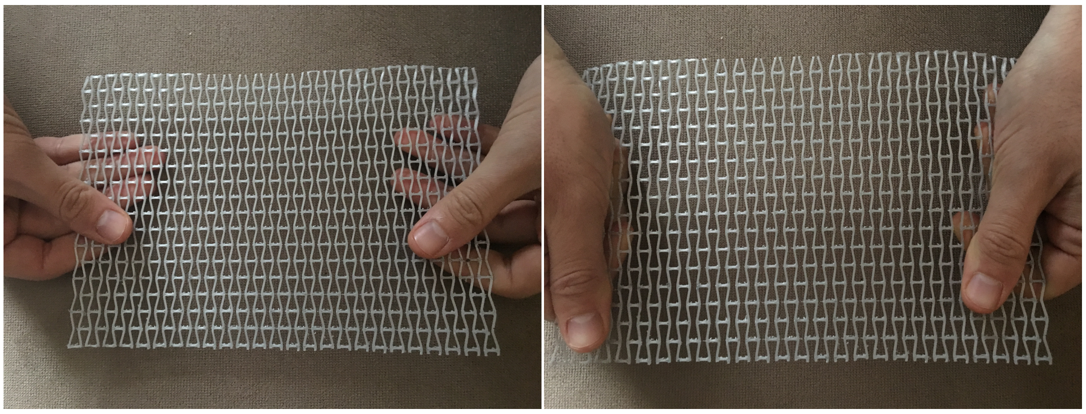

(a) Normal (b) Stretched

Figure 4.12: 3D printed flattened narrow auxetic patterns with Ultimaker 2+

2D meshes.

We utilize an open source geometry processing library, libigl [32]. The libigl provides three different parametrization method implementation: harmonic [33], least squares conformal maps [34] and as-rigid-as possible [35]. Harmonic parametrization detects the given shapes' boundaries, map these boundary points to a circle, then place the other points inside the circle. Least square conformal maps parametrization aims to minimize angular deformity while as-rigid-as possible approach tries to conserve not only angles but also distances.

In Figure 4.13, a triangle mesh male head model can be seen in different perspectives. We applied parametrization methods from libigl to this head model. Results are shown in 4.14.

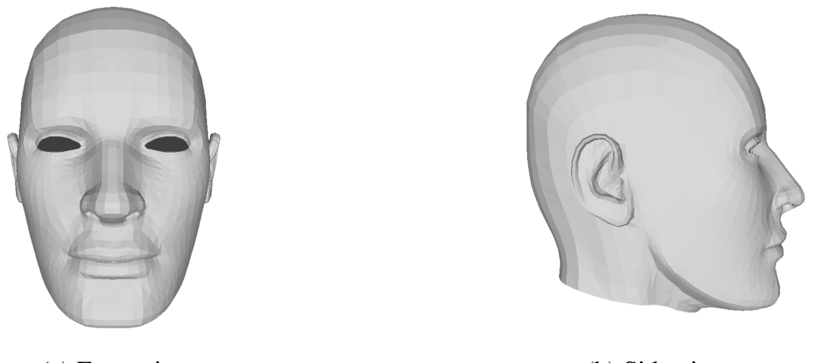
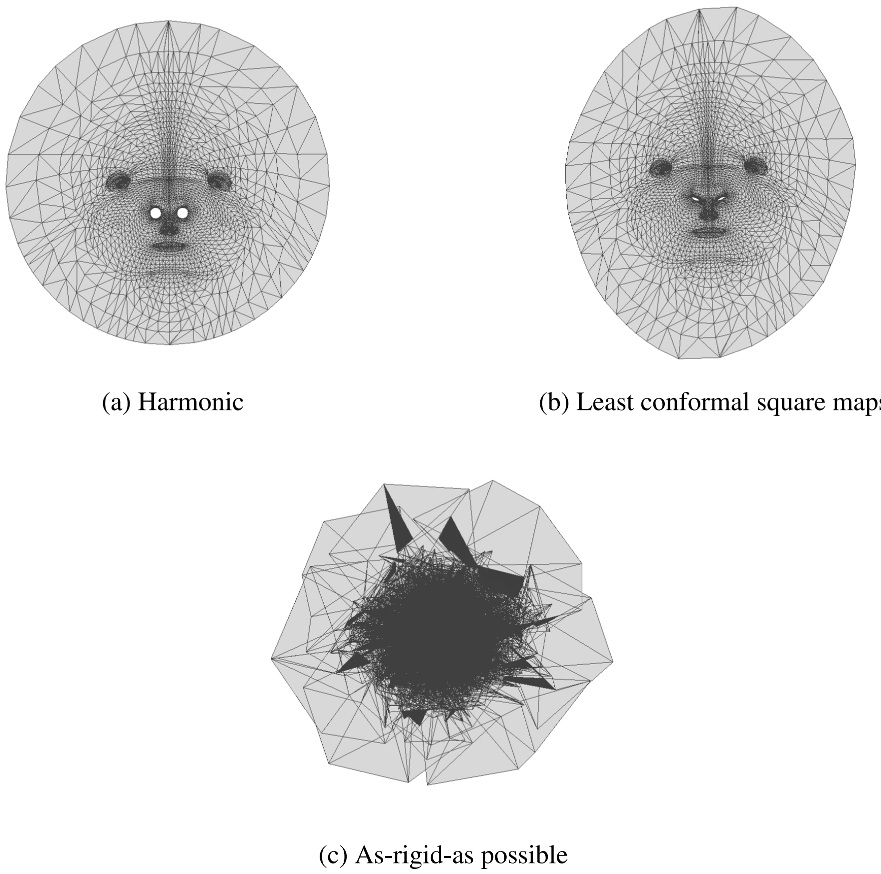

(a) Front view (b) Side view

Figure 4.13: Male head model


(a) Harmonic (b) Least conformal square maps

(c) As-rigid-as possible

Figure 4.14: Parametrization results of male head model

These results except shown in Figure 4.14c do not seem bad. If we extract auxetic patterns from them, then we can 3D print the extracted auxetic patterns. After that, we can fold the 3D printed auxetic patterns on the original model shown in Figure 4.13. However, our extraction algorithms described in Chapter 3 are not proper for these resulted meshes due to the limitations referred in Section 4.3. In order to extract auxetic patterns from these resulted meshes, developing new algorithms is required.

4.2.3 Folding Flattened Auxetic Patterns

Because of cannot extracting auxetic patterns from parameterized meshes, we decided to generate flattened simple shapes consisting of auxetic patterns. After 3D printing these flattened simple shapes, we would fold them and create target shapes. For example, we generated a flattened hexagonal prism shown in Figure 4.15. This hexagonal prism was 3D printed. We scraped off the print from the table immediately after 3D printing operation completed because we wanted to fold it. The only way to this, fold it while it is hot and not solidified completely. However, distortions occurred while scraping off the print. These distortions can be seen in Figure 4.16. Although these distortions, we tried to fold the print. Folding steps are shown in Figure 4.17.

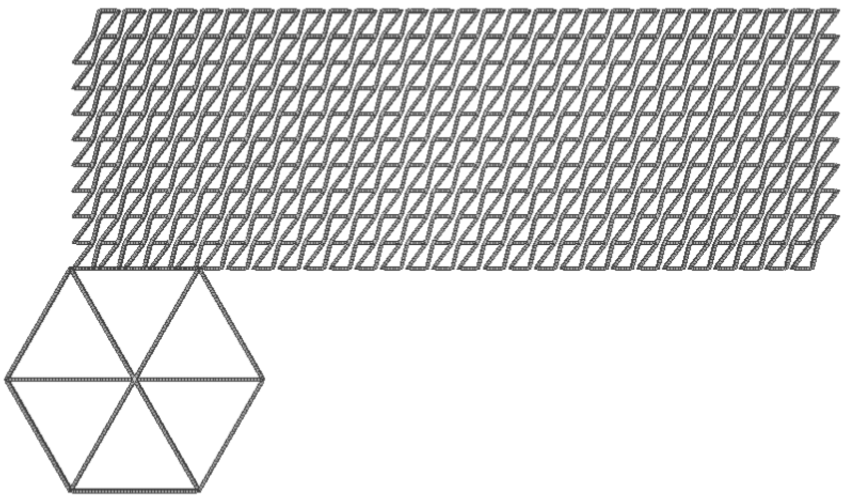


Figure 4.15: Flattened hexagonal prism

We deduced that our elementary folding approach is not successful. Scraping off the print while it is still hot is a problem and folding the print is another problem. Even if

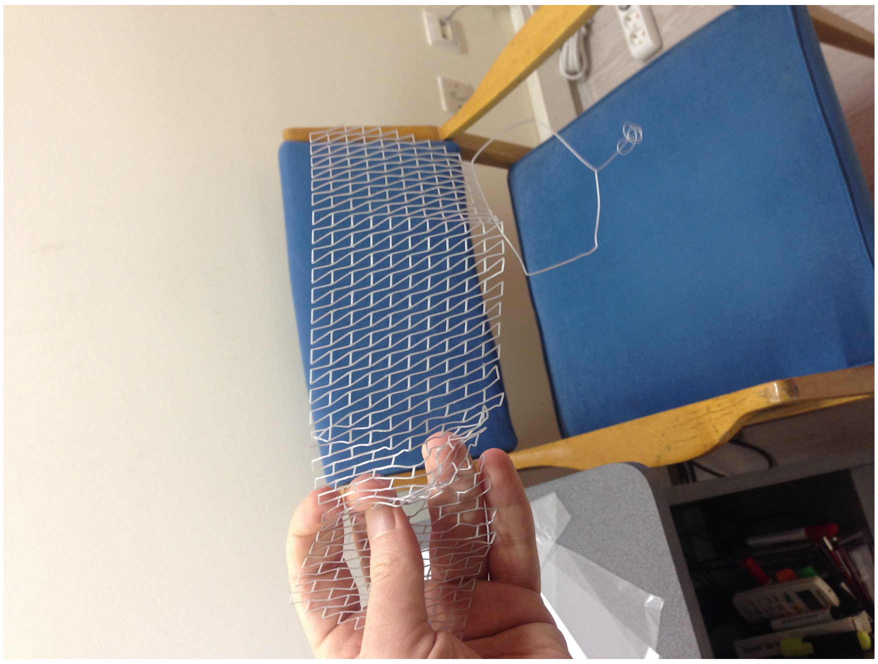


Figure 4.16: Distortions occurred on hexagonal prism 3D printed with Ultimaker 2+

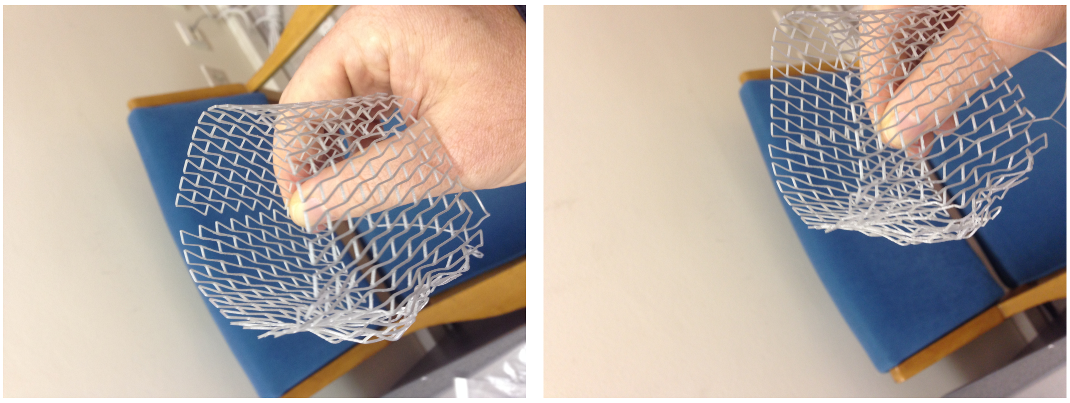

(a) (b)

Figure 4.17: Steps of folding 3D printed flattened hexagonal prism

we scrape off and fold the print successfully, we have to paste and dry it and it raises a new trouble. When we paste and dry the print, resulted shape would not be flexible and durable.

Unlike us, Guseinov et al. [3] have a study based on shaping objects from flat plates, named CurveUps, similar to our folding approach. Their method is very advanced when compared to ours. There are lots of step in their method, thus their method is a little bit tedious. However, they achieved to generate some shapes shown in Figure 4.18. Konakovic et al. [4] also managed to cover shapes with single sheet fabricated material. They analyze the target shape and then generates combinations of auxetic patterns special to this shape. Their example can be seen in Figure 4.19.

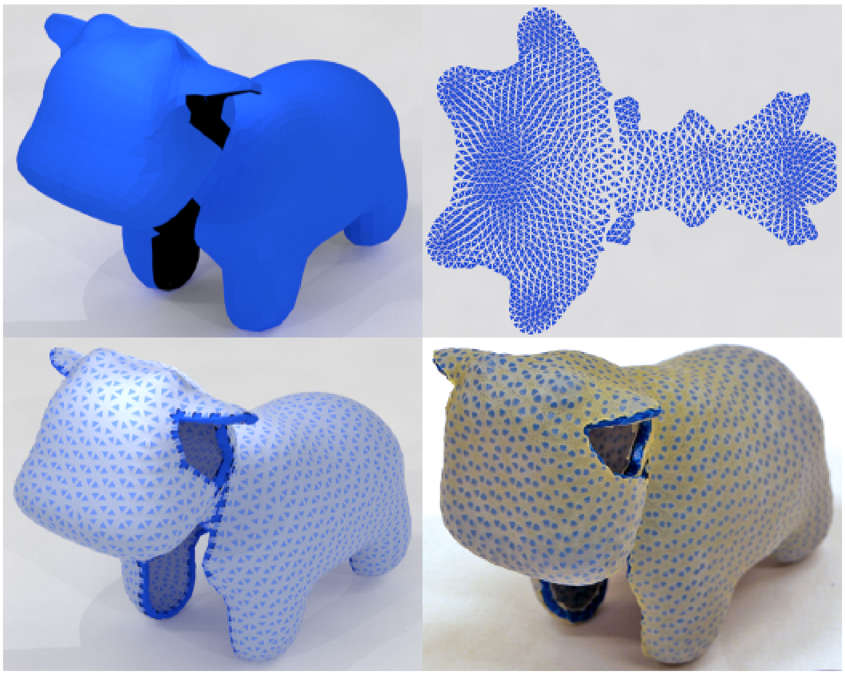


Figure 4.18: Spot generated by [3]

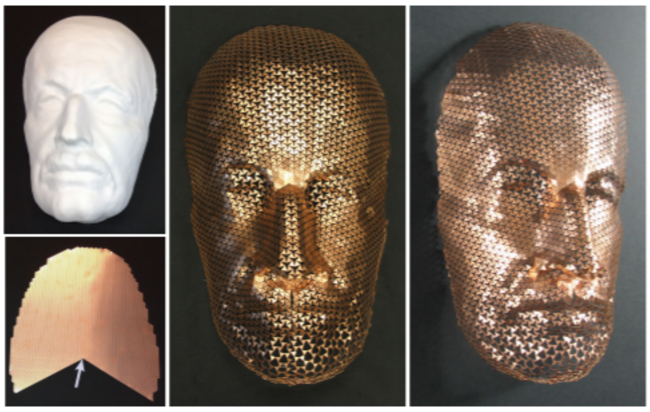


Figure 4.19: Max Planck generated by [4]

We also tried to cover complex shapes with 3D printed flattened auxetic patterns. Experiments about covering 3D printed male head model (Figure 4.13) are shown in Figure 4.20. In Figure 4.20a, large auxetic patterns shown in Figure 4.11b and in Figure 4.20b narrow auxetic patterns shown in Figure 4.12b are used. Because of these auxetic patterns are not generated particularly for the male head model, we failed.

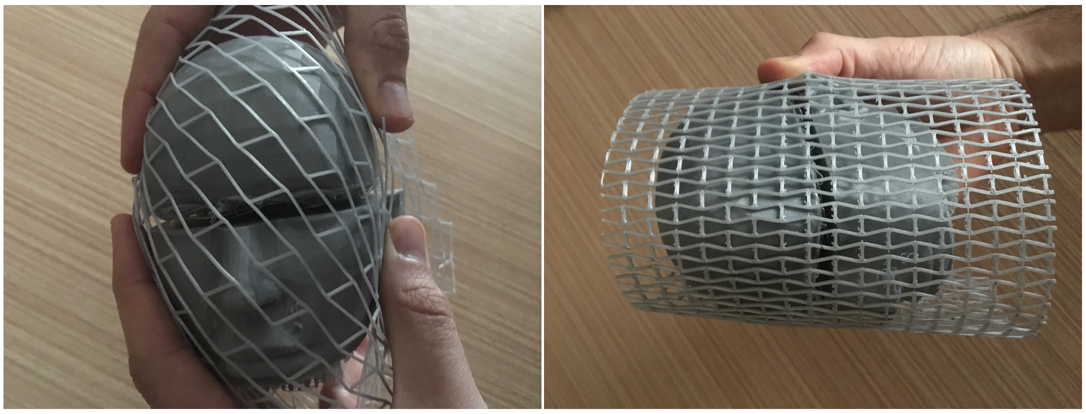
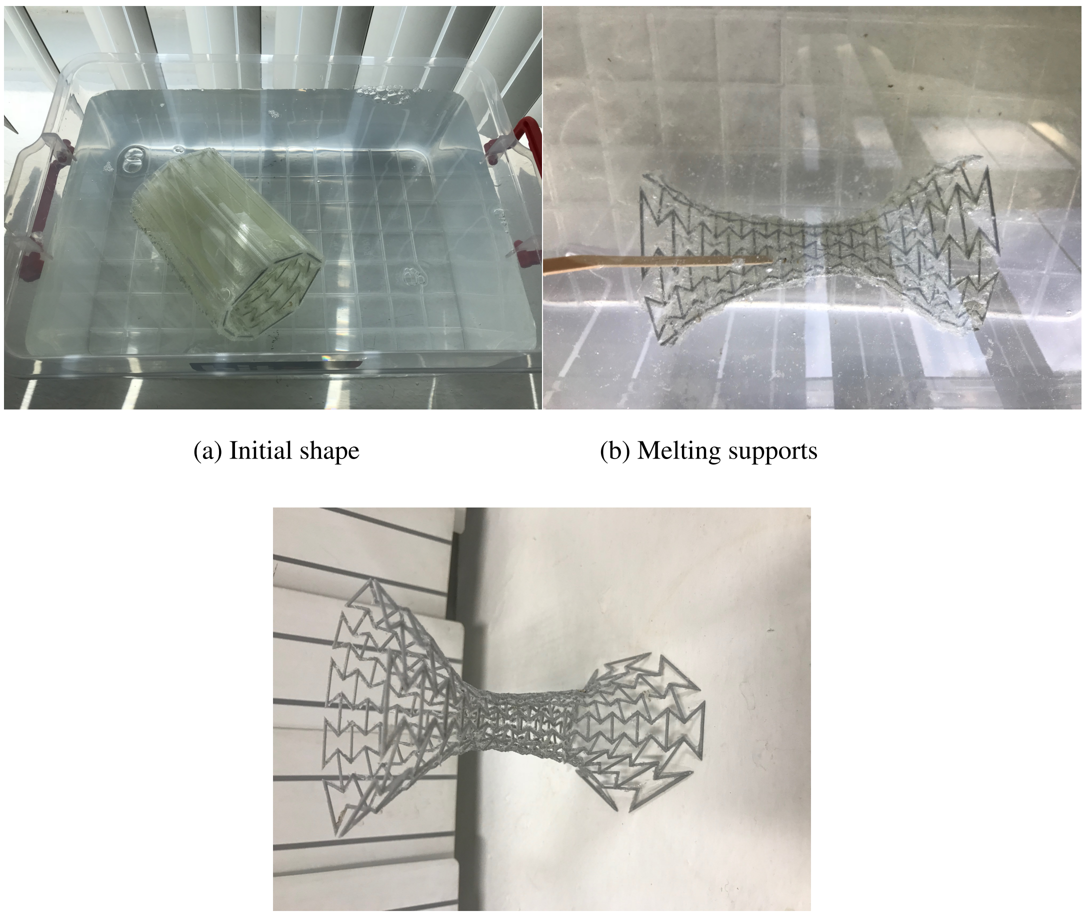

(a) With large auxetic patterns (b) With narrow auxetic patterns

Figure 4.20: Covering the male head model with flattened auxetic patterns

4.2.4 Using Meltable Support Structures

Finally, we updated our 3D printing technology and we started to proceed in a different way. The essential problem of 3D printing extracted auxetic patterns is increasing weight causing downfalls on the bottom layers described in Section 4.2.1. This problem can be eliminated by using support structures which can be provided any 3D printing software; for instance, Cura [31]. However, added support structures distorts the desired shapes. If support structures used during the 3D printing operation, 3D printed shape must be purified from these after 3D printing.

We employed a 3D printer using 2 different materials; one material to print parts of actual shape and another material to print parts of external supports. The material used for supports is water-soluble. We 3D printed extracted auxetic patterns from the goblet model shown in Figure 4.1b with this printer. Melting phases of supports and the resulted shape shown in Figure 4.21.


(a) Initial shape (b) Melting supports

(c) Final shape

Figure 4.21: 3D printed auxetic patterns with Ultimaker 3 Extended

4.3 Limitations Of Auxetic Patterns Extraction Methods

Like all algorithms, our auxetic patterns extraction algorithms described in Sections 3.2 and 3.3 have also some restrictions. They work under certain conditions. In this section, limitations of our algorithms are indicated clearly in Subsections 4.3.1 and 4.3.2.

4.3.1 Auxetic Patterns Extraction Algorithm For Triangle Meshes

The steps of what algorithm does in order to match triangles for creating bow tie:

- Gets a target triangle to find its pair.
- Clusters the triangles having a common vertex with target triangle.
- Eliminates triangles having also common edge with target triangle among this cluster.
- Eliminates triangles having a common edge with triangles eliminated at previous step among this cluster.
- Remaining triangles in the cluster are candidate triangles for creating bow tie together with target triangle.
- Algorithm expects just one triangle for each vertex of target triangle in the remaining set.
- Chooses a triangle from the remaining set considering already created bow ties' orientations.

Consider triangle T shown in Figure 4.22. Vertices of this triangle are v1, v2, v3 and all these vertices' valences are six. Triangles can be matched with T also shown in the same figure. This is the desired and proper case for the algorithm.

If the algorithm encounters a triangle having a vertex with valence value different from six during its running, it can find that triangle's match candidates. However, it cannot decide to choose which of them. Such an exceptional case for the algorithm shown in Figure 4.23. Triangle T's vertices v1, v2, v3 have valence values different from six. The algorithm does not find any match candidates for vertices v2, v3 and finds three match candidates for vertex v1. This is inconvenient for the algorithm.

The algorithm requires a triangle mesh which its vertices valence values are six mostly. Triangles having vertices with valence values different from six are discarded by the algorithm. The more triangles having vertices with valence values six the more bow tie can be founded. If the input triangle mesh would be a six-regular mesh (all vertex valence values are six), the algorithm can completely cover the input mesh with bow ties.

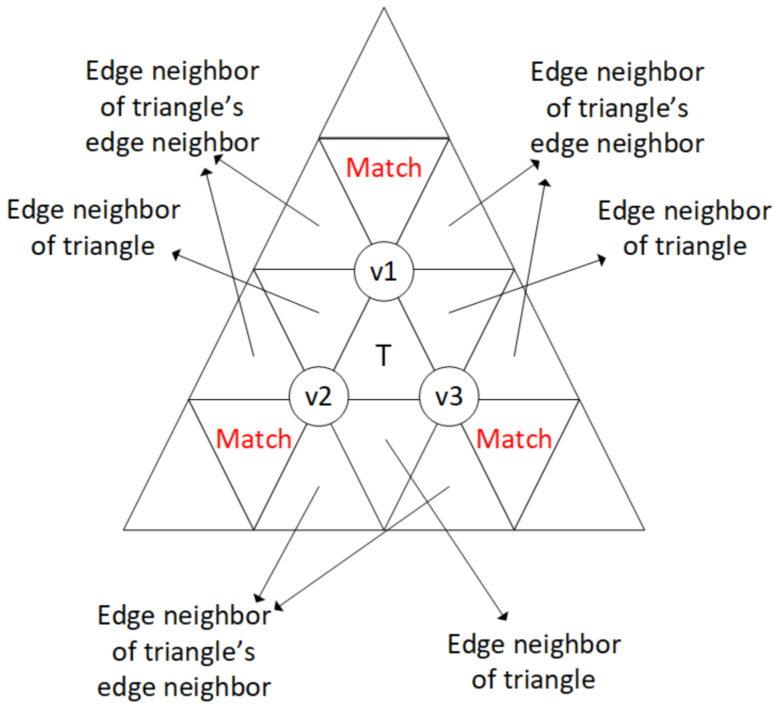

Edge neighbor of triangle's edge neighbor
Edge neighbor of triangle's edge neighbor
Edge neighbor of triangle
Edge neighbor of triangle
Match
v1
T
v2
v3
Match
Match
Edge neighbor of triangle's edge neighbor
Edge neighbor of triangle

Figure 4.22: Appropriate case for triangle mesh algorithm

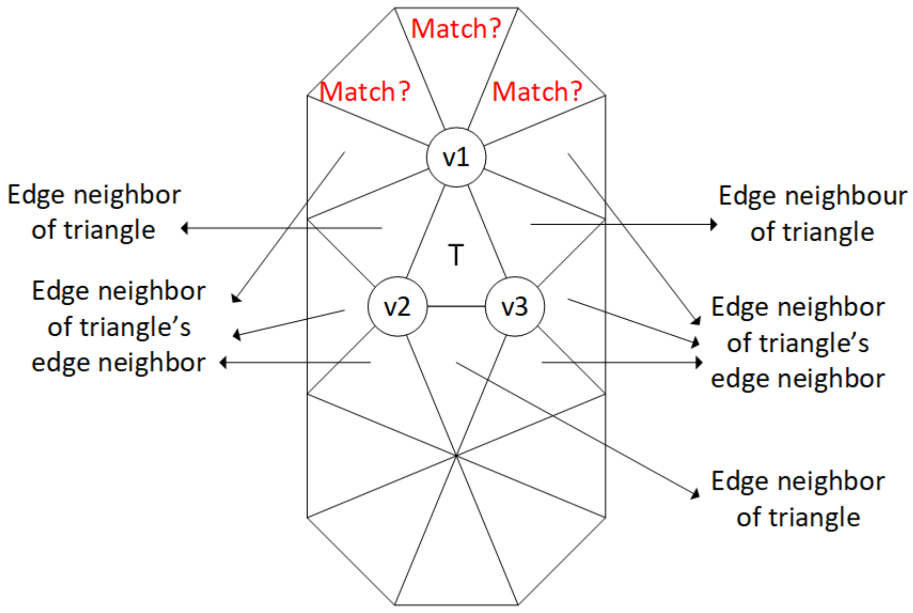

Match?
Match?
Match?
v1
Edge neighbor of triangle
Edge neighbour of triangle
T
Edge neighbor of triangle's edge neighbor
v2
v3
Edge neighbor of triangle's edge neighbor
Edge neighbor of triangle

Figure 4.23: Inappropriate case for triangle mesh algorithm

4.3.2 Auxetic Patterns Extraction Algorithm For Quad Meshes

The steps of what algorithm does in order to match quads for creating bow tie:

- Gets a target quad to find its pair.
- Clusters the quads having just one common complete edge with target quad.
- Chooses a quad from this cluster considering already created bow ties' orientations.

The proper case for the algorithm is shown in Figure 4.24. Quad Q's match candidates can be detected easily by the algorithm.

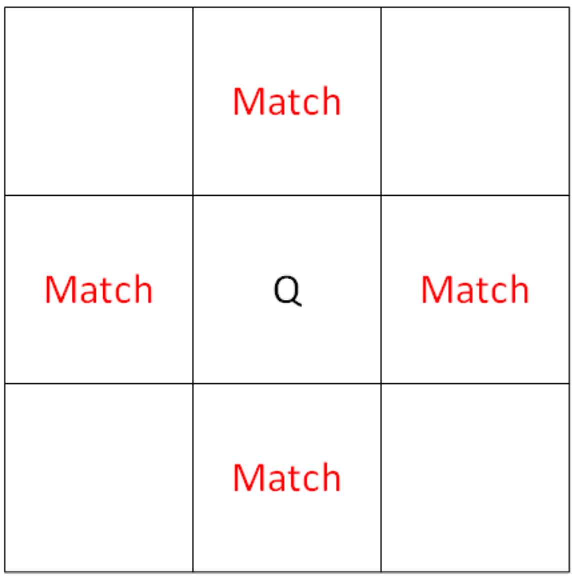


Figure 4.24: Appropriate case for quad mesh algorithm

If the algorithm encounters a quad and cannot find its edge neighbor quads during its running, it also cannot find this quad's match candidates. As a result, this quad cannot be matched. Such a case is shown in Figure 4.25. Quad Q does not have any edge neighbor having the complete common edge with it. All edges of Q involves adjacent quads' edges. There is not any complete edge between Q and its adjacent quads.

Consequently, the algorithm requires a quad mesh which its internal vertices are mostly regular (valance values are four) and its edges do not involve each other as possible as. Quads having edge involving another quad's edge are discarded by the algorithm. Valence semi-regular quad meshes classified by Bommes et al. [36] are ideally suited as inputs for our algorithm.

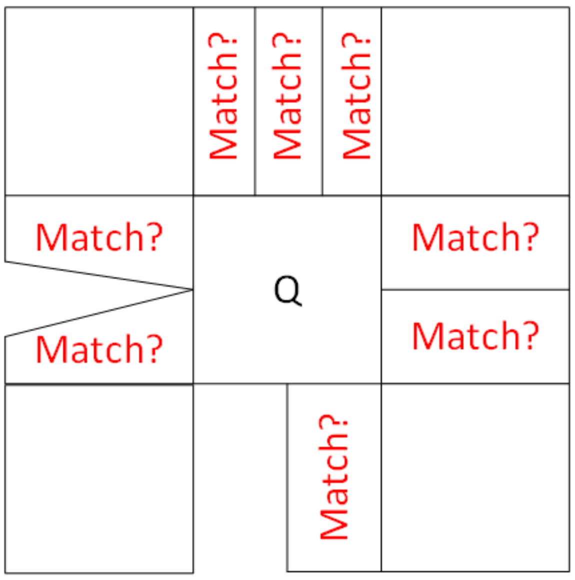


Figure 4.25: Inappropriate case for quad mesh algorithm

4.4 Performances Of Auxetic Patterns Extraction Methods

Algorithms described in Sections 3.2 and 3.3 have similar working principals. Their complexities are $O(n^2)$. Because of having similar working principals, only analyze results of algorithm designed for quad meshes described in Section 3.3 are presented.

The algorithm for quad meshes was run on three different inputs with different sizes and shapes shown in Figures 4.1a (goblet), 4.2a (vase) and 4.3a (glove). The numbers about quad meshes and running processes are exhibited in table 4.2.

Table 4.2: Running Results Of Quad Mesh Auxetic Patterns Extraction Algorithm For Different Inputs

Target Model	**Number Of Quads**	**Number Of Paired Quads**	**Number Of Internally Paired Quads**	**Running Time (milisecond)**
Goblet	896	896	896	2132
Vase	2784	2784	2752	7197
Glove	536	536	504	1582

The first model, goblet, is very suitable for the algorithm. All bow ties and internal bow ties between these bow ties are founded from quads. There is no gap on the goblet's surface shown in Figure 4.26a.

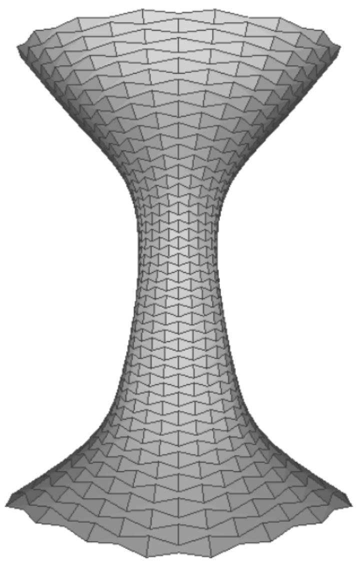
Second and third models, vase and glove are almost suitable for the algorithm. All bow ties are founded from quads; however, some internal bow ties cannot be founded due to the shapes of these models. There are some gaps on their surfaces shown in Figures 4.26b and 4.26c.

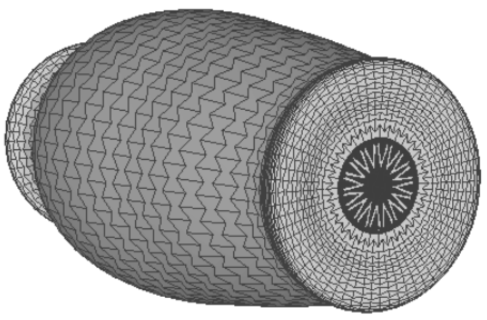
There are also models which not much suitable for the algorithm. In Figure 4.27a, a famous model in computer graphics world is shown, Stanford bunny. Although this Stanford bunny quad mesh model was regularized by Peng et al. [37], it still contains irregular vertices. We tried our algorithm on this quad mesh. The numbers about this quad mesh and running process are exhibited in table 4.3. Additionally, the result of the algorithm can be seen in Figure 4.27b. There are some lanes consisting of gaps and interlaced bow ties shown in Figures 4.27c and 4.27d. The algorithm needs to be improved.

Table 4.3: Running Results Of Quad Mesh Auxetic Patterns Extraction Algorithm For Stanford Bunny Model

Target Model	**Number Of Quads**	**Number Of Paired Quads**	**Number Of Internally Paired Quads**	**Running Time (milisecond)**
Bunny	12087	11856	11004	351665

The algorithm was run multiple times for each quad meshes and average running time was calculated and presented as running time. Change on running times of algorithm for different quad meshes is as expected. Increasing quad counts in meshes also increases the running times with respect to the complexity of the algorithm which is $O(n^2)$.


(a) Goblet


(b) Vase

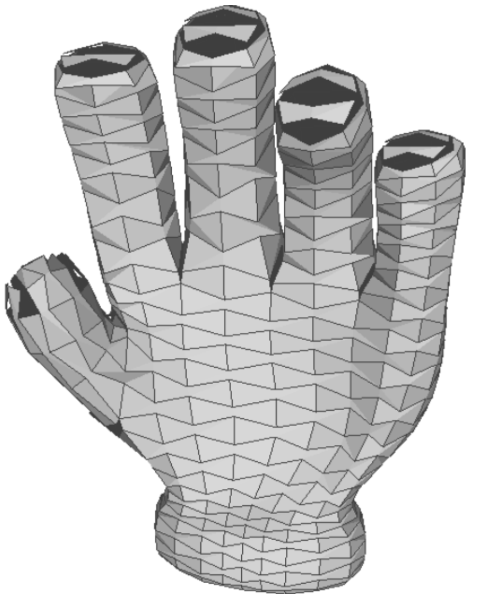
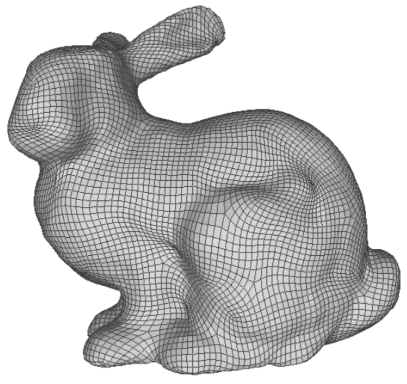
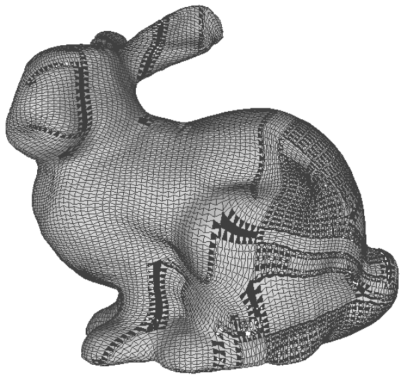
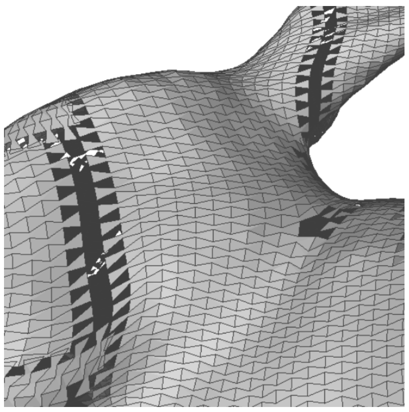

(c) Glove

Figure 4.26: Surfaces of different models after quad mesh auxetic patterns extraction algorithm run


(a) Quad mesh Stanford bunny model [37]


(b) Final result obtained by the algoritm


(c) Zoomed to the lanes consisting of gaps

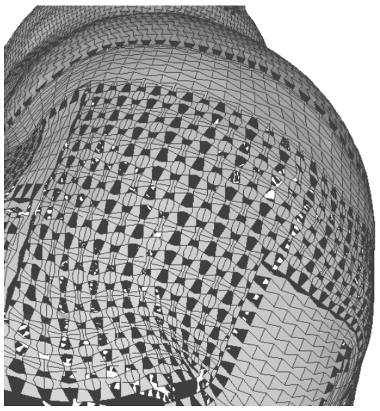

(d) Zoomed to the interlaced bow ties

Figure 4.27: Failure case of the quad mesh auxetic patterns extraction algorithm on Stanford bunny

CHAPTER 5

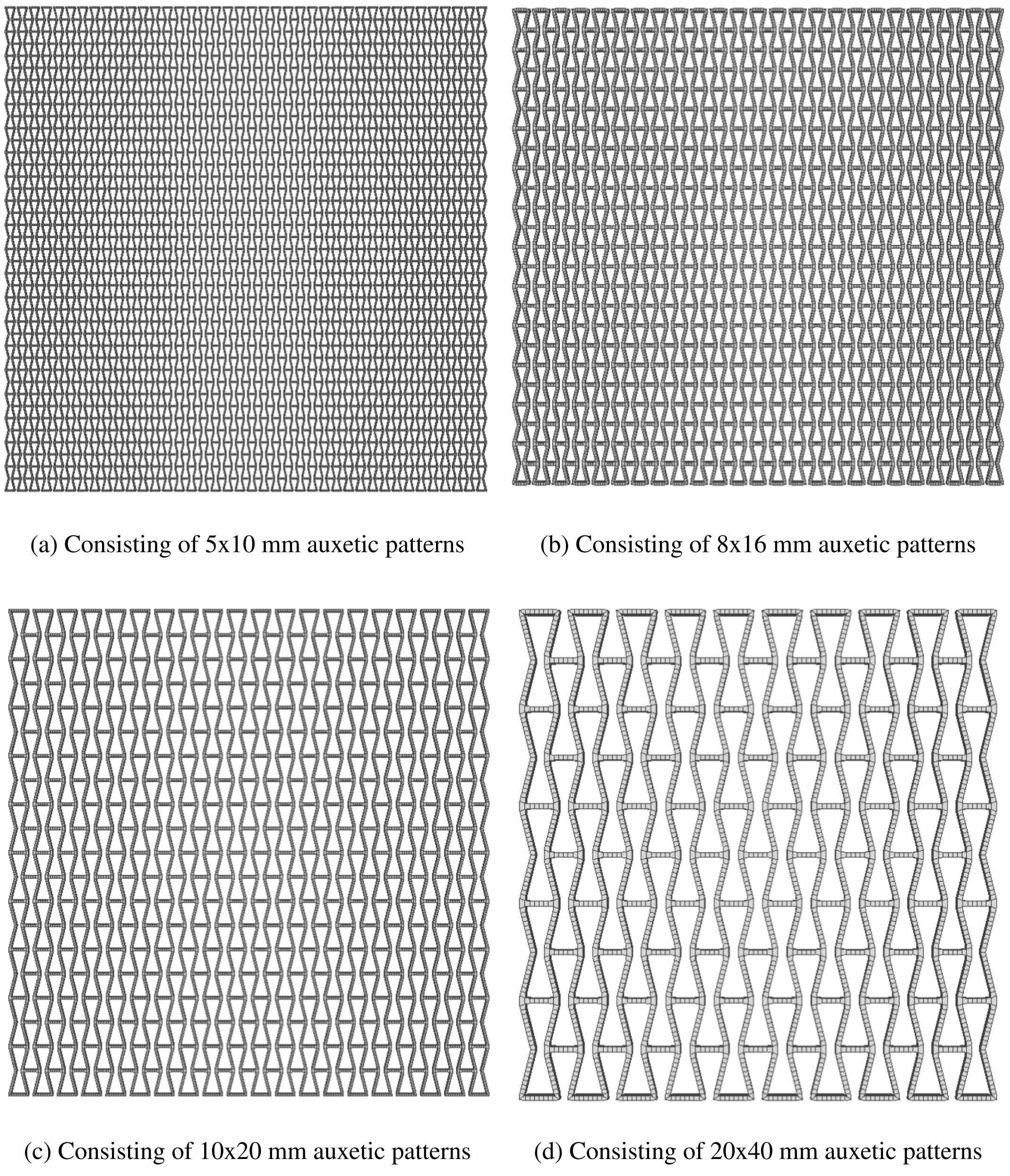
CONCLUSION

In this thesis, we developed novel methods for extracting auxetic patterns from 3D printable digital designs by modifying the existing mesh primitives directly and fully-automatically. Then, we used our methods and obtained extracted auxetic patterns which can be 3D printed. We 3D printed some of them and observed the auxetic behaviors on outputs. During 3D printing processes, we encountered problems caused by the difficulties of fabricating auxetic patterns. We tried different methods to fabricate auxetic patterns and create objects consisting of auxetic patterns; however, our tries generates other problems. We strived to solve these problems in order to obtain convenient 3D printed objects having auxetic behavior.

To conclude, although we managed to 3D print some extracted auxetic patterns successfully, we do not have perfect 3D printed objects having auxetic behavior. Additionally, we cannot extract auxetics patterns for arbitrary digital designs. Our methods have some constraints. Moreover, we cannot 3D print successfully all auxetic patterns extracted with our methods by 3D printers we already have. We created flattened digital designs consisting of auxetic patterns which can be 3D printed easily. Then, we tried to fold them to generate objects having auxetic behavior but we failed. However, our works are encouraging and 3D printing auxetic objects is still popular in the worlds of computer graphics and digital fabrication.

In the future, we plan to improve our algorithms in order to extract auxetic patterns from more complex digital models having nontrivial geometric features. This requires a multi-resolution approach that employs auxetic patters of varying sizes on the same mesh (Figure 5.1). Establishing smooth transition between such multi-resolution patterns is expected to alleviate the gapping and interlacing problems demonstrated in

Figure 4.27. Such an approach will also enable varying elasticity over the fabricated object, where the places tiled with smaller patterns will be less elastic than the ones tiled with larger patterns. Finally, we aim to perform fabrication using more sophisticated 3D printers than the ones we employed.


(a) Consisting of 5x10 mm auxetic patterns (b) Consisting of 8x16 mm auxetic patterns

(c) Consisting of 10x20 mm auxetic patterns (d) Consisting of 20x40 mm auxetic patterns

Figure 5.1: Auxetic planes having dimensions 200x200 mm

REFERENCES

[1] S. Mueller, S. Im, S. Gurevich, A. Teibrich, L. Pfisterer, F. Guimbretière, and P. Baudisch, “Wireprint: 3d printed previews for fast prototyping,” in *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pp. 273–280, ACM, 2014.

[2] H. Peng, R. Wu, S. Marschner, and F. Guimbretière, “On-the-fly print: Incremental printing while modelling,” in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pp. 887–896, ACM, 2016.

[3] R. Guseinov, E. Miguel, and B. Bickel, “Curveups: shaping objects from flat plates with tension-actuated curvature,” *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 64, 2017.

[4] M. Konaković, K. Crane, B. Deng, S. Bouaziz, D. Piker, and M. Pauly, “Beyond developable: computational design and fabrication with auxetic materials,” *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 89, 2016.

[5] L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk, “State of the art in example-based texture synthesis,” in *Eurographics 2009, State of the Art Report, EG-STAR*, pp. 93–117, Eurographics Association, 2009.

[6] E. Praun, A. Finkelstein, and H. Hoppe, “Lapped textures,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 465–470, ACM Press/Addison-Wesley Publishing Co., 2000.

[7] S. Lefebvre and H. Hoppe, “Appearance-space texture synthesis,” in *ACM Transactions on Graphics (TOG)*, vol. 25, pp. 541–548, ACM, 2006.

[8] K. Zhou, X. Huang, X. Wang, Y. Tong, M. Desbrun, B. Guo, and H.-Y. Shum, “Mesh quilting for geometric texture synthesis,” in *ACM Transactions on Graphics (TOG)*, vol. 25, pp. 690–697, ACM, 2006.

[9] C. Ma, L.-Y. Wei, and X. Tong, "Discrete element textures," in *ACM Transactions on Graphics (TOG)*, vol. 30, p. 62, ACM, 2011.

[10] A. Garg, A. O. Sageman-Furnas, B. Deng, Y. Yue, E. Grinspun, M. Pauly, and M. Wardetzky, "Wire mesh design.," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 66–1, 2014.

[11] C. Torres, T. Campbell, N. Kumar, and E. Paulos, "Hapticprint: Designing feel aesthetics for digital fabrication," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pp. 583–591, ACM, 2015.

[12] J. Dumas, A. Lu, S. Lefebvre, J. Wu, and C. Dick, "By-example synthesis of structurally sound patterns," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 137, 2015.

[13] W. Chen, X. Zhang, S. Xin, Y. Xia, S. Lefebvre, and W. Wang, "Synthesis of filigrees for digital fabrication," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 98, 2016.

[14] C. Schumacher, B. Thomaszewski, and M. Gross, "Stenciling: Designing structurally-sound surfaces with decorative patterns," in *Computer Graphics Forum*, vol. 35, pp. 101–110, Wiley Online Library, 2016.

[15] J. Zehnder, S. Coros, and B. Thomaszewski, "Designing structurally-sound ornamental curve networks," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 99, 2016.

[16] H. Zhao, F. Gu, Q.-X. Huang, J. Garcia, Y. Chen, C. Tu, B. Benes, H. Zhang, D. Cohen-Or, and B. Chen, "Connected fermat spirals for layered fabrication," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 100, 2016.

[17] A. Jacobson, I. Baran, J. Popovic, and O. Sorkine, "Bounded biharmonic weights for real-time deformation.," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 78–1, 2011.

[18] Y. Sahillioğlu and Y. Yemez, "Minimum-distortion isometric shape correspondence using em algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2203–2215, 2012.

[19] J. Spillmann and M. Teschner, "Cosserat nets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 2, pp. 325–338, 2009.

[20] J. Pérez, B. Thomaszewski, S. Coros, B. Bickel, J. A. Canabal, R. Sumner, and M. A. Otaduy, "Design and fabrication of flexible rod meshes," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 138, 2015.

[21] J. Panetta, Q. Zhou, L. Malomo, N. Pietroni, P. Cignoni, and D. Zorin, "Elastic textures for additive fabrication," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 135, 2015.

[22] C. Schumacher, B. Bickel, J. Rys, S. Marschner, C. Daraio, and M. Gross, "Microstructures to control elasticity in 3d printing," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 136, 2015.

[23] J. Martínez, J. Dumas, and S. Lefebvre, "Procedural voronoi foams for additive manufacturing," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 44, 2016.

[24] W. Wang, T. Y. Wang, Z. Yang, L. Liu, X. Tong, W. Tong, J. Deng, F. Chen, and X. Liu, "Cost-effective printing of 3d objects with skin-frame structures," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, p. 177, 2013.

[25] L. Lu, A. Sharf, H. Zhao, Y. Wei, Q. Fan, X. Chen, Y. Savoye, C. Tu, D. Cohen-Or, and B. Chen, "Build-to-last: strength to weight 3d printed objects," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 97, 2014.

[26] R. Wu, H. Peng, F. Guimbretière, and S. Marschner, "Printing arbitrary meshes with a 5dof wireframe printer," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 101, 2016.

[27] Y. Liu and H. Hu, "A review on auxetic structures and polymeric materials," *Scientific Research and Essays*, vol. 5, no. 10, pp. 1052–1063, 2010.

[28] R. Underhill, "Defense applications of auxetic materials," *Defense Systems Information Analysis Center*, 07 2014.

[29] G. Imbalzano, J. P. Tran, T. Ngo, and P. Lee, "Three-dimensional modelling of

auxetic sandwich panels for localised impact resistance," vol. 19, pp. 291–316, 12 2015.

[30] *blender - Free and Open Source Creation Suit*, 2017 (last released Sep 12, 2017). available: `https://www.blender.org/`.

[31] *Ultimaker Cura Software*, 2017 (last released May 12, 2017). available: `https://ultimaker.com/en/products/ultimaker-cura-software`.

[32] *libigl - A simple C++ geometry processing library*, 2018 (last commited Jan 11, 2018). available: `https://github.com/libigl/libigl/`.

[33] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution analysis of arbitrary meshes," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 173–182, ACM, 1995.

[34] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, "Least squares conformal maps for automatic texture atlas generation," in *ACM transactions on graphics (TOG)*, vol. 21, pp. 362–371, ACM, 2002.

[35] P. Mullen, Y. Tong, P. Alliez, and M. Desbrun, "Spectral conformal parameterization," in *Computer Graphics Forum*, vol. 27, pp. 1487–1494, Wiley Online Library, 2008.

[36] D. Bommes, B. Lévy, N. Pietroni, C. Silva, M. Tarini, and D. Zorin, "State of the art in quad meshing," 2012.

[37] C.-H. Peng, E. Zhang, Y. Kobayashi, and P. Wonka, "Connectivity editing for quadrilateral meshes," in *ACM Transactions on Graphics (TOG)*, vol. 30, p. 141, ACM, 2011.