TRIANGULATION GUIDED HIGH CLEARANCE COLLISION-FREE PATHS

By

Sandeep Maharjan

Bachelor of Engineering in Computer Engineering
Tribhuvan University
2018

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2024

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

April 5, 2024

This thesis prepared by

Sandeep Maharjan

entitled

Triangulation Guided High Clearance Collision-Free Paths

is approved in partial fulfillment of the requirements for the degree of

Master of Science – Computer Science
Department of Computer Science

Laxmi Gewali, Ph.D.
Examination Committee Chair

Mingon Kang, Ph.D.
Examination Committee Member

John Minor, Ph.D.
Examination Committee Member

Henry Selvaraj, Ph.D.
Graduate College Faculty Representative

Alyssa Crittenden, Ph.D.
Vice Provost for Graduate Education & Dean of the Graduate College

Abstract

Algorithms dealing with the construction of high clearance collision-free paths in the presence of polygonal obstacles is an important problem in robotics and transportation engineering. Method of extracting collision-free paths guided by triangulation of free space is examined. Two algorithms for improving the standard triangulation guided algorithms are presented. The time complexities of the presented algorithms are analysed. Finally, further applications of the proposed techniques are discussed.

Acknowledgements

I want to thank my advisor, Dr. Laxmi Gewali, above all for his guidance, patience, and support while I was preparing my thesis. His vast expertise and understanding in computational geometry greatly aided me in finishing the thesis, for which I am immensely grateful.

Secondly, I would like to express my appreciation to the members of my thesis committee for their commitment to serving on the committee and for their ongoing helpful criticism and suggestions meant to improve the thesis.

In addition, I am grateful to my girlfriend Binela Dangol for her encouragement and moral support during my graduate studies, which culminated in this thesis. Her review, proofreading, and comments on my thesis are really appreciated.

Lastly, I am really grateful to my roommates and friends, Mr. Shrijan Shrestha and Mr. Aavash Adhikari, for always keeping me motivated and upbeat. I also sincerely appreciate my family's unfluctuating support and well wishes for the future.

Sandeep Maharjan

University of Nevada, Las Vegas

May 2024

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Issues pertaining to point site distribution in the plane have been considered extensively in computational geometry [O'R98]. Computing the closest pair among point sites, finding nearest neighbor for each point site have many applications. Such problems are commonly known as proximity problems. Geometric structures for computing proximity relationships between point sites is elegantly captured by a structure known as the Voronoi Diagram [O'R98]. Once we have the Voronoi diagram of points distributed in the plane, it is relatively easy to extract triangulation, Euclidean minimum spanning tree, convex hull, and closest neighbors to each site.

A triangulation of point sites is extensively used in finite element analysis which in turn is used for generating approximate solution for heat flow problems [PAZ03].

The dual of the Voronoi diagram is an special kind of triangulation called Delaunay triangulation. Several intriguing features are satisfied by the Delaunay triangulation, and these include:

i) In-circle property which states that the circle through vertices of the triangle does not include any other point sites.

ii) Point sites' convex hull forms the border of the Delaunay triangulation's external face.

iii) Euclidean minimum spanning tree of point sites is included in the Delaunay triangulation.

iv) Consider the two nodes, v_i and v_j, the shortest path among them in Delaunay triangulation is not very far from the straight line separation between the nodes v_i and v_j. The shortest distance among the two nodes v_i and v_j in Delaunay triangulation is no more than 5.08 times the straight line distance [DFS90].

In this thesis, we examine the quality of shortest collision-free paths guided by triangulation. In chapter 2, we review important existing algorithms for constructing collision-free paths in the presence of polygonal obstacles. In chapter 3, we describe the instances where the standard method of constructing collision-free paths guided by triangulation of free-space may generate invalid paths. We then propose two algorithms to modify the triangulation of free-space so that the path obtained from the dual of the triangulation is valid. The first algorithm, called 'flipping method', is based on flipping carefully chosen edge sharing triangle pairs. This algorithm has the time complexity of $O(n^2)$. The second algorithm we propose is based on constructing envelopes around obstacles and constructing triangulation on the free space not including the envelop strips. This algorithm has the time complexity of $O(n^2)$. In chapter 4, we discuss the limitation of the proposed algorithms and approaches for further improvement and extensions. We discuss the viability of using cluster of points rather than the individual points as the nodes for triangulation. We also discuss the development of the methods for constructing collision-free paths that do not have sharp turns in addition to having high clearance.

Chapter 2

Review of Collision-free Paths Algorithms

In this chapter, we present a brief review of the algorithms for the collision-free paths with high clearance.

2.1 Voronoi Diagram and High Clearance Path

The Voronoi diagram formed by a set of n point sites $p_1, p_2, ..., p_n$ in the plane is the decomposition of the plane into convex regions. An example of the Voronoi diagram formed by 15 point sites is shown in figure 2.1. Voronoi diagrams satisfy certain useful proximity properties for point sites. Some of the properties of Voronoi diagrams used for developing efficient proximity algorithms can be listed as follow [O'R98]:

a) Every region on the Voronoi Diagram is convex; some are bounded, while others are unbounded. The faces corresponding to the points inside the convex hull are bounded. The faces corresponding to the sites on the convex hull are unbounded. In Figure 2.1, there are seven unbounded faces corresponding to point sites $p_2, p_4, p_6, p_3, p_9, p_5$ and p_7. The faces corresponding to interior point sites are bounded.

b) The closest neighbor of point site p_i is one of the sites corresponding to the faces adjacent to the faces $f(i)$, the face for p_i.

c) The dual of the Voronoi diagram of point sites $p_1, p_2, ..., p_n$ gives the triangulation of the point sites. Such a triangulation is called the Delaunay triangulation. An example of dual and the

Delaunay triangulation is shown in the figure 2.2.

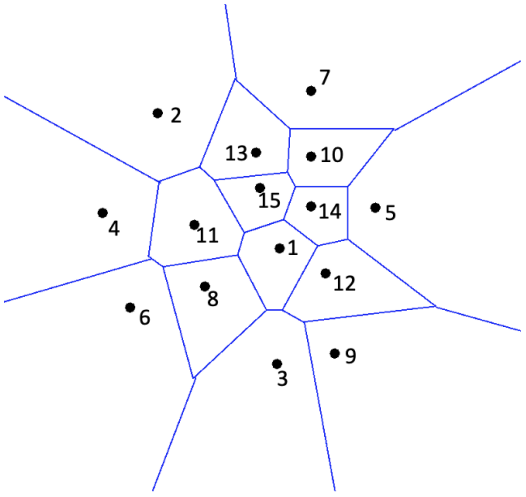


Figure 2.1: Voronoi Diagram of 15 Point Sites

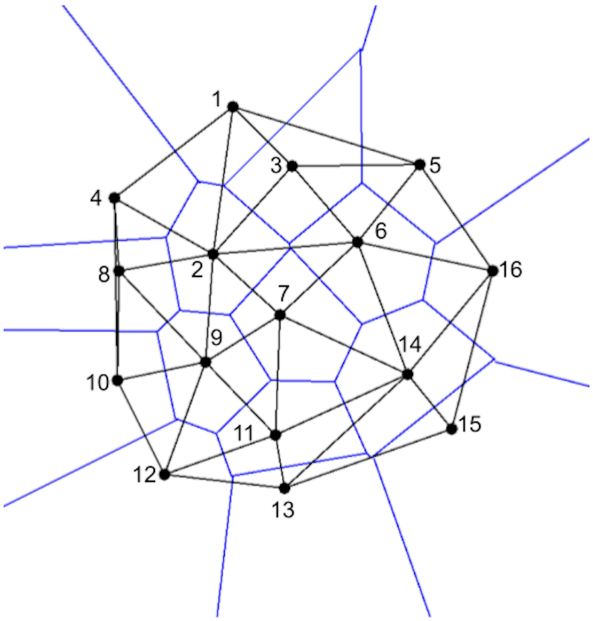


Figure 2.2: Delaunay Triangulation of 16 Point Sites

2.2 Review of Triangulation Algorithm

It is a partitioning of the polygon into triangles whose vertices are also the vertices of the polygon. Figure 2.3 and 2.4 depicts valid and invalid triangulated polygon respectively.

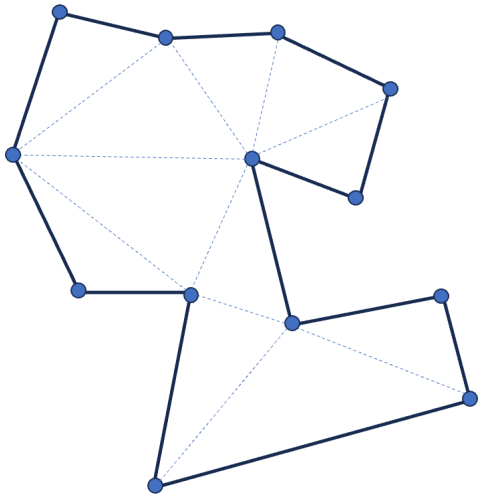


Figure 2.3: Valid

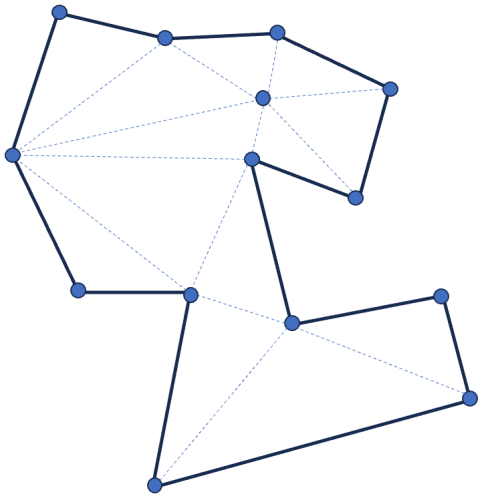


Figure 2.4: Invalid

In most triangulation studies, valid triangulations are considered [O'R98]. It is known that in any simple polygon of$n$ vertices, it admits $(n-2)$ triangles. The vertices v_{i-1}, v_i and v_{i+1} are the three consecutive vertices of the polygon which forms an 'ear' if v_{i-1}, v_{i+1} is an internal diagonal. In the figure 2.5(a), v_3, v_1 and v_2 forms an ear whereas, v_7, v_8, v_9 and v_6, v_7, v_8 do not form ears.

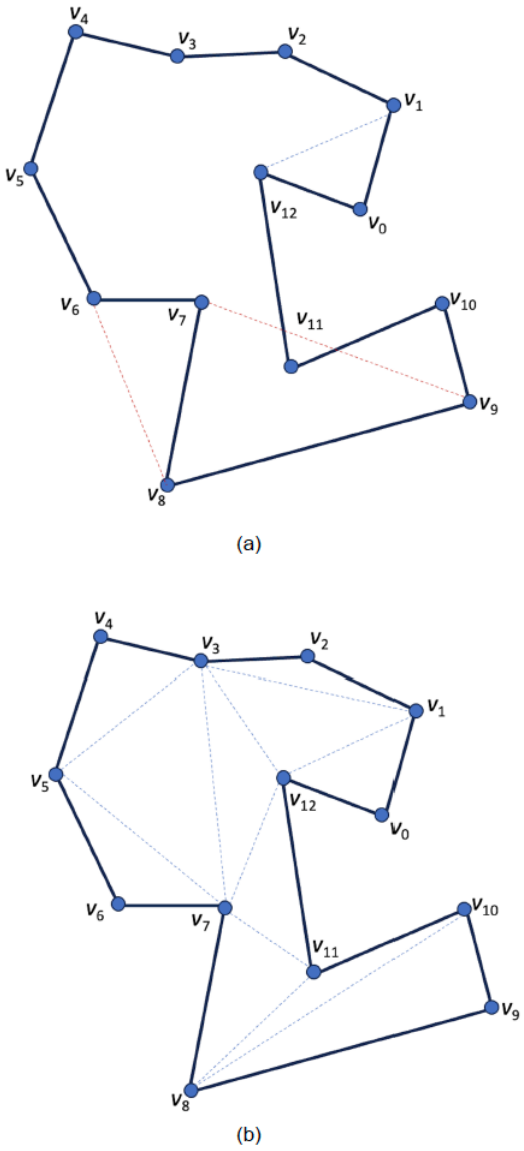


Figure 2.5: Triangulation Using Ear Removal

A very simple algorithm for triangulating a polygon is based on repeatedly removing ears. When a ear is removed from a polygon of n vertices, remaining part is a polygon of $n-1$ vertices. This process of repeatedly removing the ear generates a triangulation. For example in figure 2.5(a), when ears are repeatedly removed from the polygon, it results a triangulation of the polygon as shown in the figure 2.5(b). However, the time complexity of this algorithm is $O(n^2)$. Finding an efficient algorithm for triangulating a polygon is a very important problem.

Computing collision free path based on by the placement of random nodes in free region has been considered by Barun Thapa. The detail are available in [Tha23].

Another algorithm to triangulate a polygon is based on decomposing a simple polygon into

monotone parts. It is known that in linear time, a monotone polygon can be triangulated [dBvKOS00]. Figure 2.6 illustrates the decomposition of s simple polygon into three components. The time complexity of such an algorithm is $O(n \log n)$ [dBvKOS00].

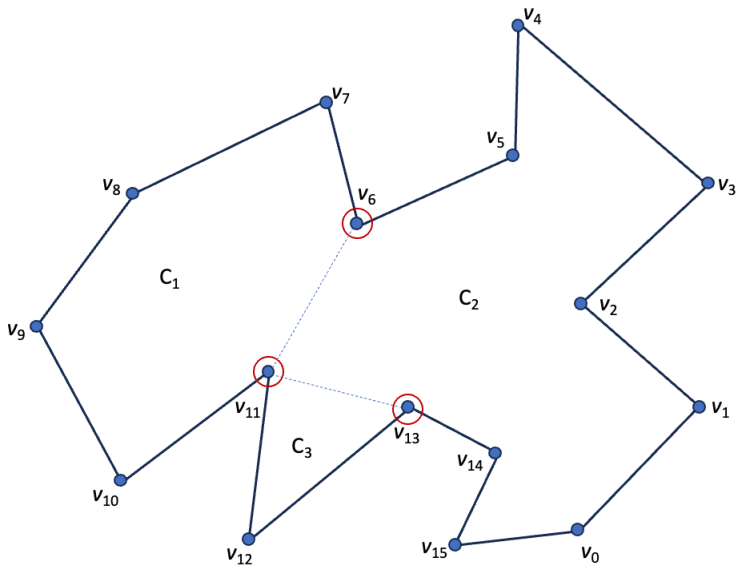


Figure 2.6: Monotone Components

Triangulating a simple polygon in linear time was considered by several researchers. For this issue, Bernard Chazelle [Cha91] developed a linear time algorithm. The algorithm is very complicated and its implementation needs sophisticated data structures.

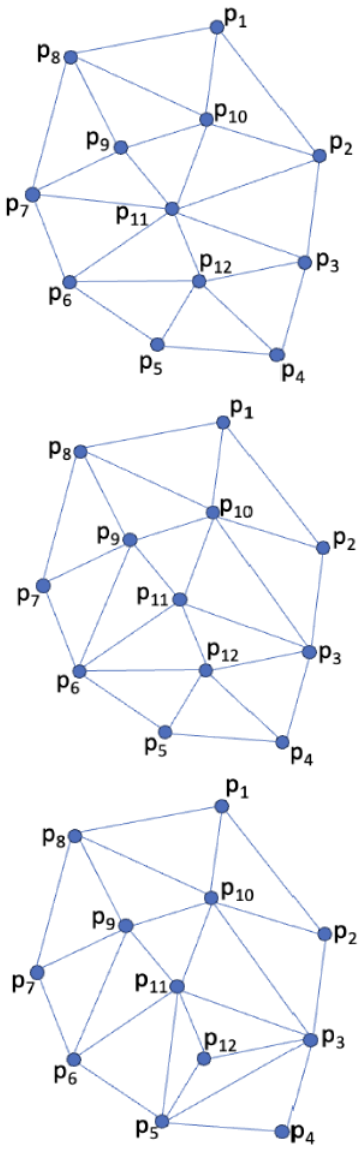


Figure 2.7: Triangulation of Same Point Set

Triangulating a set of points in two dimensions has many practical applications. In 2D, there are numerous methods for triangulating a set of points. Figure 2.7 illustrates several triangulations of the same point set. It has been proven, in fact, that there are exponentially many ways to triangulate a point set [O'R98]. The problem of triangulating a set of points has a lower bound of $\Omega(n \log n)$.

2.3 Triangulation by Segment Rotation

A straightforward method for triangulating the free-space in the presence of polygon obstacles is to insert non-intersecting diagonals incrementally. The approach is to examine all candidate diagonals formed by the vertices of polygon obstacles. If the candidate diagonal does not intersect with the already selected diagonals then it is taken as the diagonal of the triangulation. This process of examining/selecting candidate diagonals is repeated until the free-space is completely triangulated. There are $O(n^2)$ candidate diagonals. Each candidate diagonal needs to be checked for possible intersection which has execution time of $O(n)$. This straightforward approach takes $O(n^3)$ time and is a very slow algorithm.

A faster algorithm can be developed by considering rotating segment from a given vertex. Imagine a line segment originating from a vertex a_3 and extending to meet a vertex q_{10} on the boundary wall as shown in figure 2.8. This segment (a_3, q_{10}) intersects with four edges (c_2, c_3), (c_4, c_5), (d_2, d_3), (d_3, d_1). In a balanced search tree, these intersecting edges are maintained in the priority of distance from origin a_3. The search tree is updated as the segment rotates in clockwise direction. The updating is done when the rotating segment passes through vertices as shown in figure 2.9. The detail of this method is available in [Lee78]. The rotation procedure is done by selecting all vertices as the origin. The time complexity of the rotating method is $O(n^2 \log n)$.

The set of visibility segments in the presence of polygonal obstacles gives a structure called Visibility Graph [O'R98]. Visibility graph contains edges of any triangulation. Hence a Visibility Graph can be used to extract desired triangulation edges. It is established that the time required to compute a visibility graph is proportional to its size [GM87]. Hence a triangulation algorithm based on the visibility graph takes time $O(n^2)$.

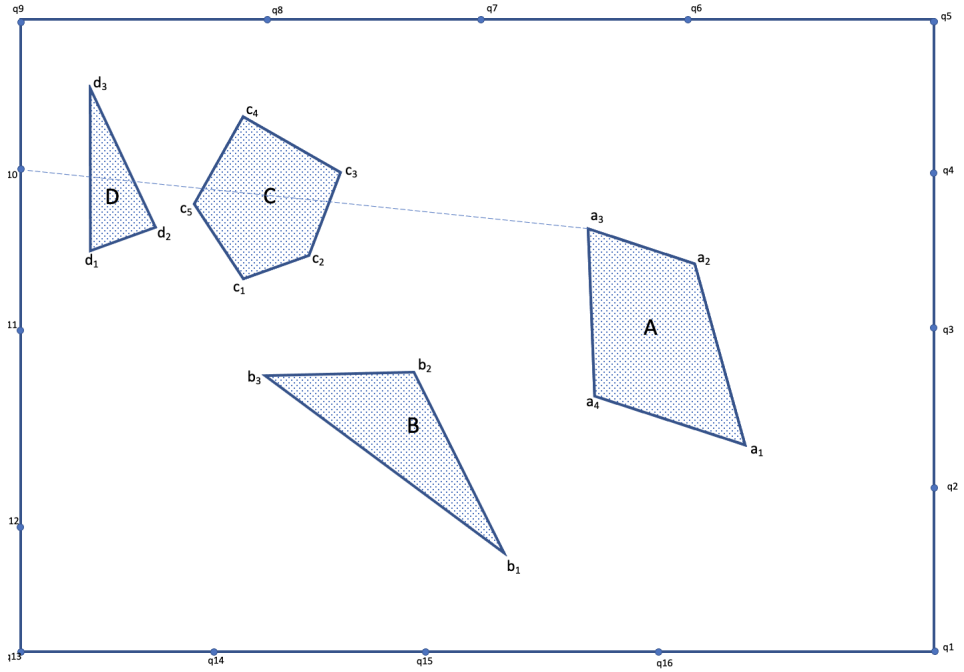


Figure 2.8: Rotating 1

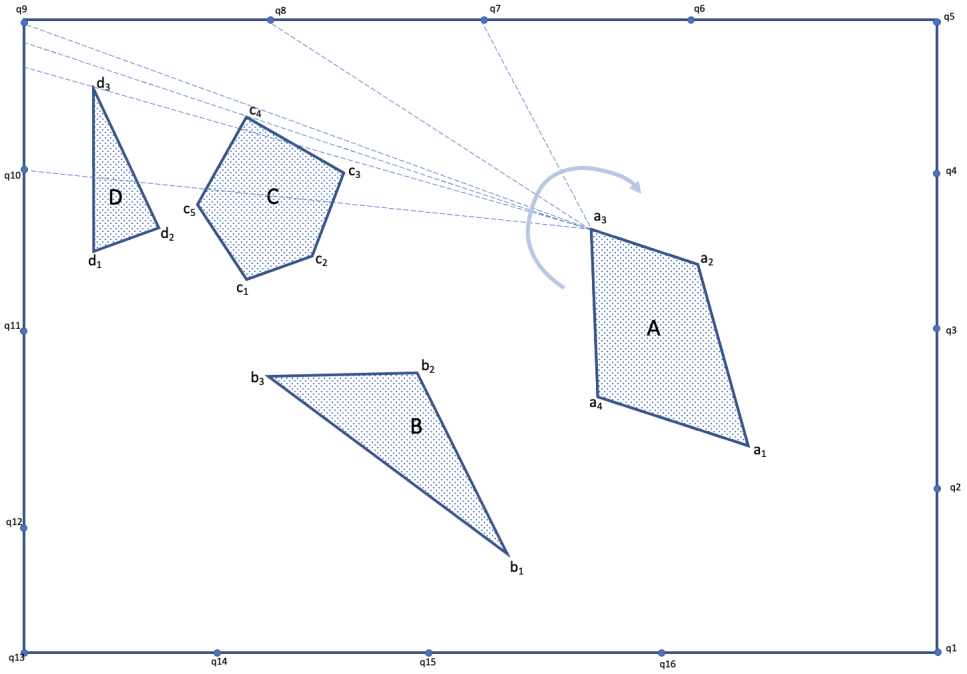


Figure 2.9: Rotating 2

2.4 Doubly Connected Edge List Data Structure (DCEL)

A DCEL Data Structure is used to represent and store planar graphs. A triangulation of a set of points is a restricted planar graph and hence DCEL data structure is used to store the triangulation efficiently. We briefly describe below the DCEL data structure.

A planar graph consists of three entities - Faces, Edges and Vertices. Let us use the figures 2.10 and 2.11 to understand the detail of DECL properties.

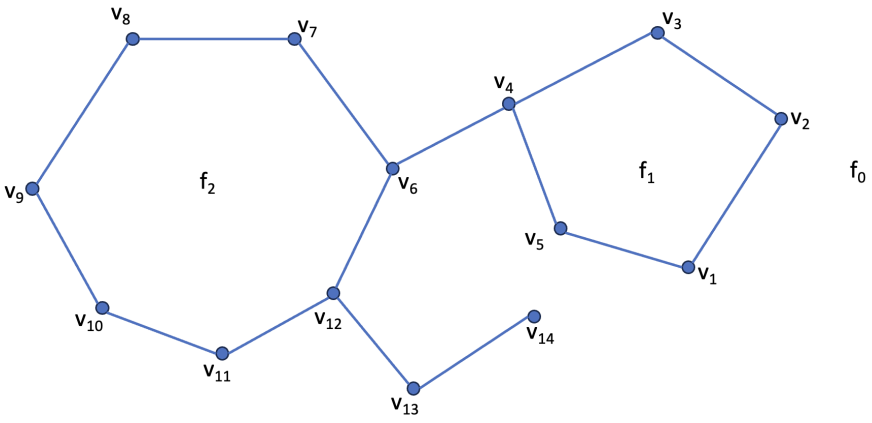


Figure 2.10: Planar Graph Showing Faces and Vertices

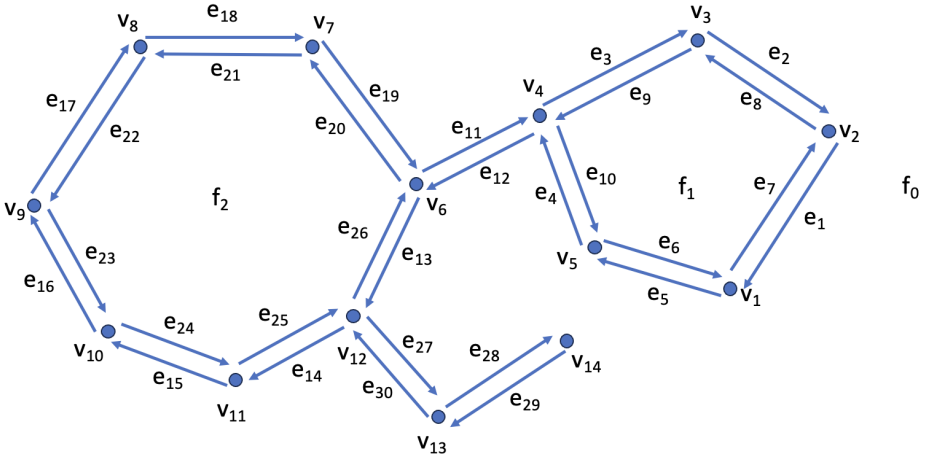


Figure 2.11: Planar Graph Showing Faces, Edges and Vertices

Each edges is represented by a pair of directed half edges e_1 and e_2, directed in opposite direction. The details are available in [dBvKOS00]. Table 2.1 demonstrates how to store the half edges record in a table.

Half Edge	Twin	Next	Previous	Incident Vertex	Incident Face
e_1	e_7	e_5	e_2	v_2	f_0
e_2	e_8	e_1	e_3	v_3	f_0
e_3	e_9	e_2	e_{11}	v_4	f_0
e_4	e_{10}	e_{12}	e_5	v_5	f_0
e_5	e_6	e_4	e_1	v_1	f_0
e_6	e_5	e_7	e_{10}	v_5	f_1
e_7	e_1	e_8	e_6	v_1	f_1
e_8	e_2	e_9	e_7	v_2	f_1
e_9	e_3	e_{10}	e_8	v_3	f_1
e_{10}	e_4	e_6	e_9	v_4	f_1
e_{11}	e_{12}	e_3	e_{19}	v_6	f_0
e_{12}	e_{11}	e_{13}	e_4	v_4	f_0
e_{13}	e_{26}	e_{27}	e_{12}	v_6	f_0
e_{14}	e_{25}	e_{15}	e_{30}	v_{12}	f_0
e_{15}	e_{24}	e_{16}	e_{14}	v_{11}	f_0
e_{16}	e_{23}	e_{17}	e_{15}	v_{10}	f_0
e_{17}	e_{22}	e_{18}	e_{16}	v_9	f_0
e_{18}	e_{21}	e_{19}	e_{17}	v_8	f_0
e_{19}	e_{20}	e_{11}	e_{18}	v_7	f_0
e_{20}	e_{19}	e_{21}	e_{26}	v_6	f_2
e_{21}	e_{18}	e_{22}	e_{20}	v_7	f_2
e_{22}	e_{17}	e_{23}	e_{21}	v_8	f_2
e_{23}	e_{16}	e_{24}	e_{22}	v_9	f_2
e_{24}	e_{15}	e_{25}	e_{23}	v_{10}	f_2
e_{25}	e_{14}	e_{26}	e_{24}	v_{11}	f_2
e_{26}	e_{13}	e_{20}	e_{25}	v_{12}	f_2
e_{27}	e_{30}	e_{28}	e_{13}	v_{12}	f_0
e_{28}	e_{29}	e_{29}	e_{27}	v_{13}	f_0
e_{29}	e_{28}	e_{30}	e_{28}	v_{14}	f_0
e_{30}	e_{27}	e_{14}	e_{29}	v_{13}	f_0

Table 2.1: A Table Showing the Half Record of Edges

If a planar graph G is available in a DECL representation, it can be navigated very efficiently. It is convenient to have classes for vertices, faces, and half-edges. Figure 2.12 depicts useful methods for class HalfEdge.

```
e1.getTwin()         /* return the twin half edge of e1*/
e1.getNext()         /* return the next half edge of e1*/
e1.getPrev()         /* return the prev half edge of e1*/
e1.getIncFace()      /* return the face bounded by e1 */
                     etc
```

Figure 2.12: Methods for the HalfEdge Class

With these methods, a function that returns the size of the face incident in half edge e_1 can be written conveniently as shown in figure 2.13.

```
int getSize(Face f1){
    HalfEdge e1 = f1.getIncFace();
    HalfEdge ex = e1.getNext();
    HalfEdge eSave = e1.getNext();
    int size = 1;
    while (e1.getNext()!=eSave){
        size++;
        e1=e1.getNext();
       }
    return size;
}
```

Figure 2.13: Size of the Face Incident in a HalfEdge

It is straightforward to see that the size of a face can be determined in time proportional to the size of the face.

The degree of a vertex v can be determined in time proportional to the degree value of the vertex as shown in figure 2.14.

```
int v1getDegree(){
    int degree = 1;
    HalfEdge e1 = v1.getIncHalfEdge();
    HalfEdge eSave = e1;
    While(e1.getTwin().getNext() != eSave){
        degree++;
        e1 = e1.getTwin().getNext();
      }
    return degree;
}
```

Figure 2.14: Degree of a Vertex v

2.5 Collision-free Paths and Triangulation

A commonly used method for constructing collision-free path is based on the triangulation of free space [Kal05]. The method is based on triangulating free space (space outside of obstacles) and selecting nodes inside each triangle. The centroid of the triangle is commonly used to place nodes. Let us explain this method with an example as shown in figure 2.15, which shows triangulation of free space. In each triangle a node is placed in its centroid. Two centroid nodes are connected by an edge if the corresponding triangles share an edge. When all connection of centroids is done, a structure called the **d**ual of the triangulation is formed as shown in figure 2.16. It is observed that the dual lies outside obstacles and hence can be used for planning collision-free paths. Furthermore, it can be observed that the dual does have good clearance from obstacles.

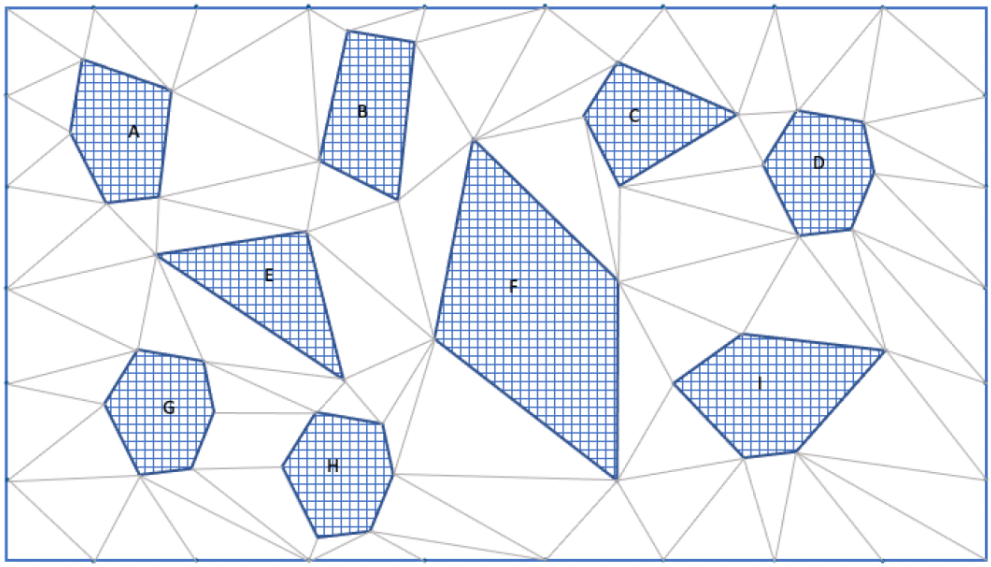


Figure 2.15: A Scene of Obstacles Inside a Box

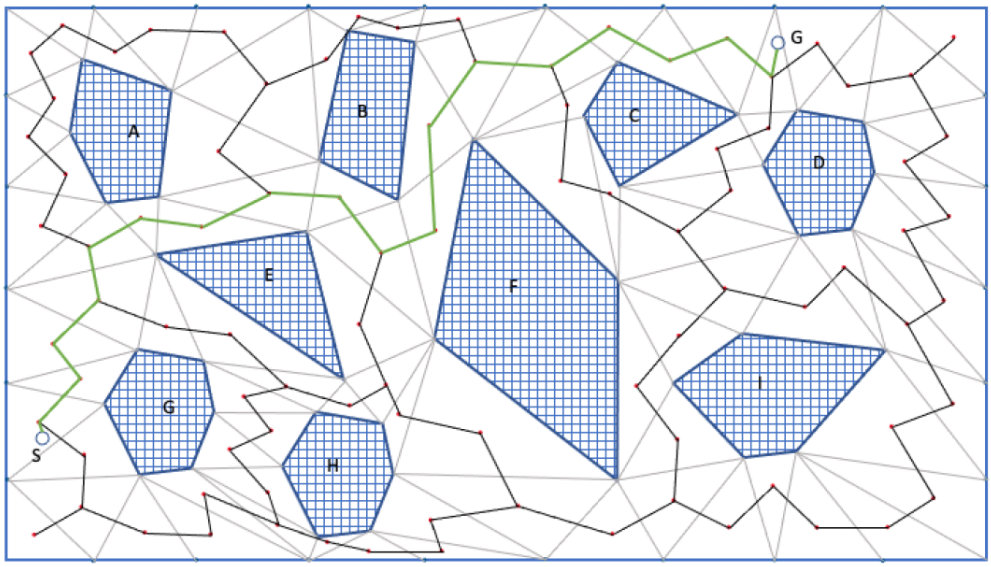


Figure 2.16: Illustrating the Triangulation of Free Space and the Dual

Chapter 3

Modified Triangulations

In this chapter we present modifications to the standard triangulation method for extracting collision free paths. We discuss the cases where the dual of a triangulation becomes an invalid structure: the dual could actually intersect with obstacles. We then present two methods for fixing this intersection problem. The first algorithm is based on modifying the given triangulation by the process of 'flipping'. The second algorithm we present is based on embedding obstacles with envelopes and generating a triangulation in the free-space outside the envelopes.

3.1 Difficulties with Standard Dual Method

As discussed in the previous chapter in the dual method, the free-space is triangulated by taking obstacle vertices as the triangulation vertices. However, for some obstacle distribution, the dual could actually intersect with obstacles which may result in generating an invalid path. This is illustrated in figure 3.1. An example of obstacles enclosed in a rectangular box is shown in figure 3.1a. A triangulation of the free space by locating nodes on the centroids of triangles and the corresponding dual are shown in figure 3.1b-c. As observed in figure 3.1d, the dual crosses an obstacle, making this structure invalid: the path extracted from this dual could intersect with obstacle D. This is stated in the following lemma.

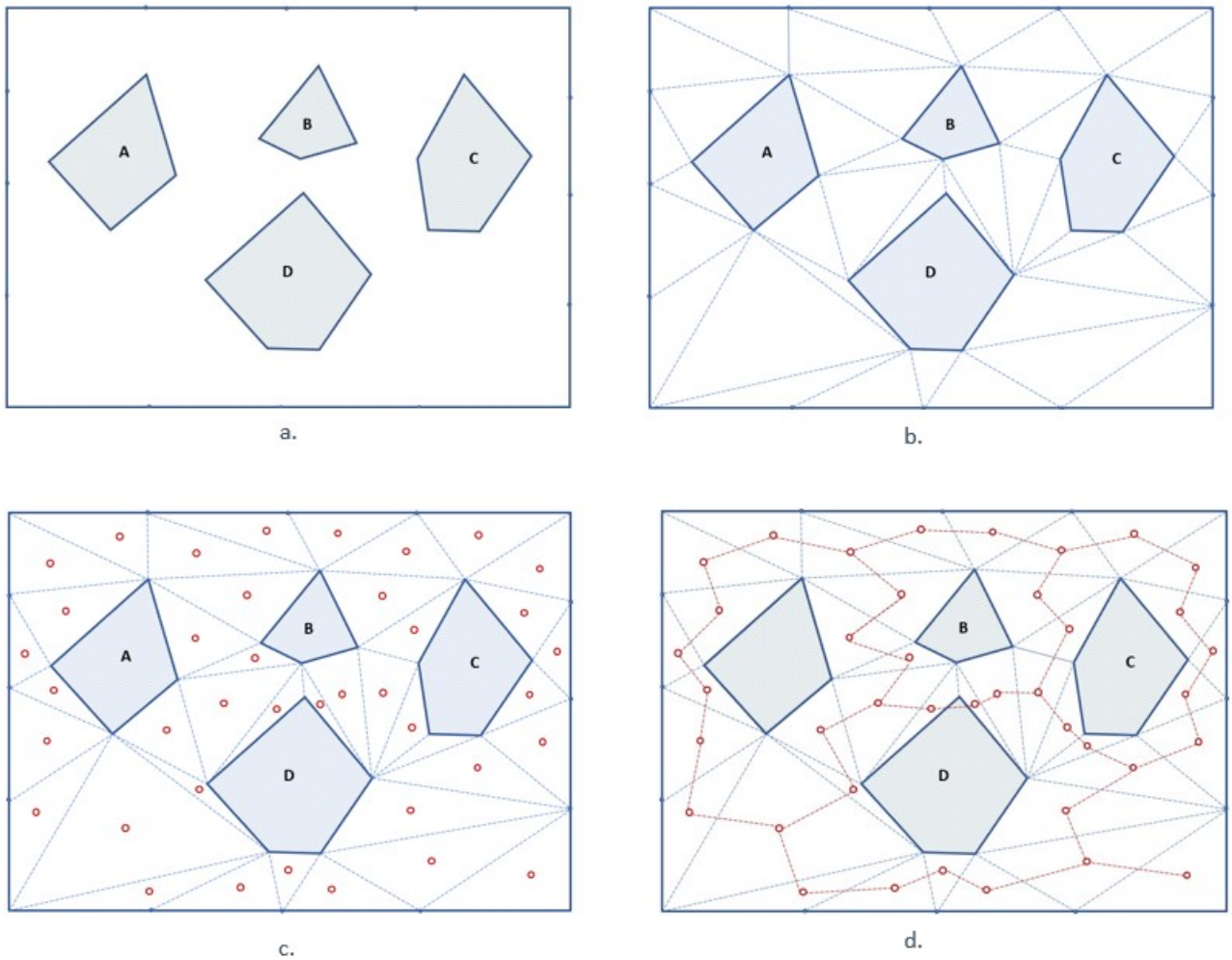


Figure 3.1: Invalid Triangulation Dual

Lemma 3.1 *The triangulation dual could intersect with obstacles.*

3.2 Method of Flipping

Lemma 3.1 motivates us to seek for a triangulation so that its dual is free of intersection with obstacles. We consider the process of 'triangle flipping' for a pair of triangles that are adjacent to each other (i.e. they share an edge). Let us examine a pair of adjacent triangles bgf and baf, that share a common edge bf as shown in figure 3.2.

In the quadrilateral gbaf, diagonal ga is inserted and diagonal bf is removed. This process transforms triangle pair (bgf and baf) into a new pair (afg and agb). Similar flipping is performed in the quadrilateral afed.

Our approach for fixing the invalid dual is to flip the triangles in the neighborhood of the

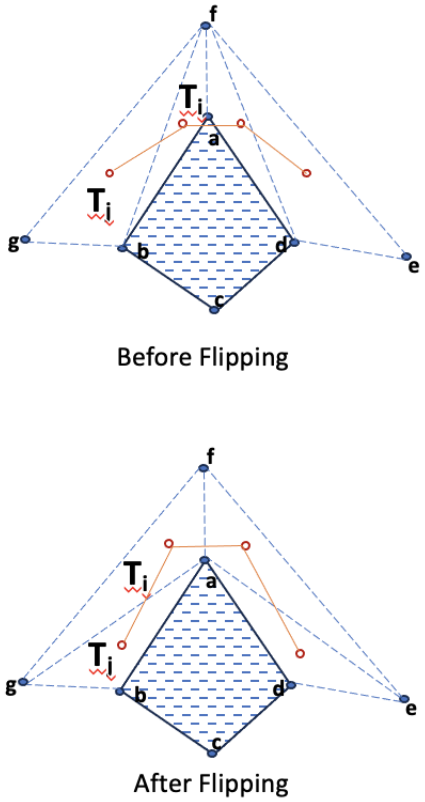


Figure 3.2: Illustrating Triangle Flipping

obstacle vertex when the dual crosses or is very close to the obstacle.

Figure 3.3 illustrates the flipping process near the top vertex of obstacle D. It is observed that after flipping, the dual does not intersect with obstacle and becomes valid.

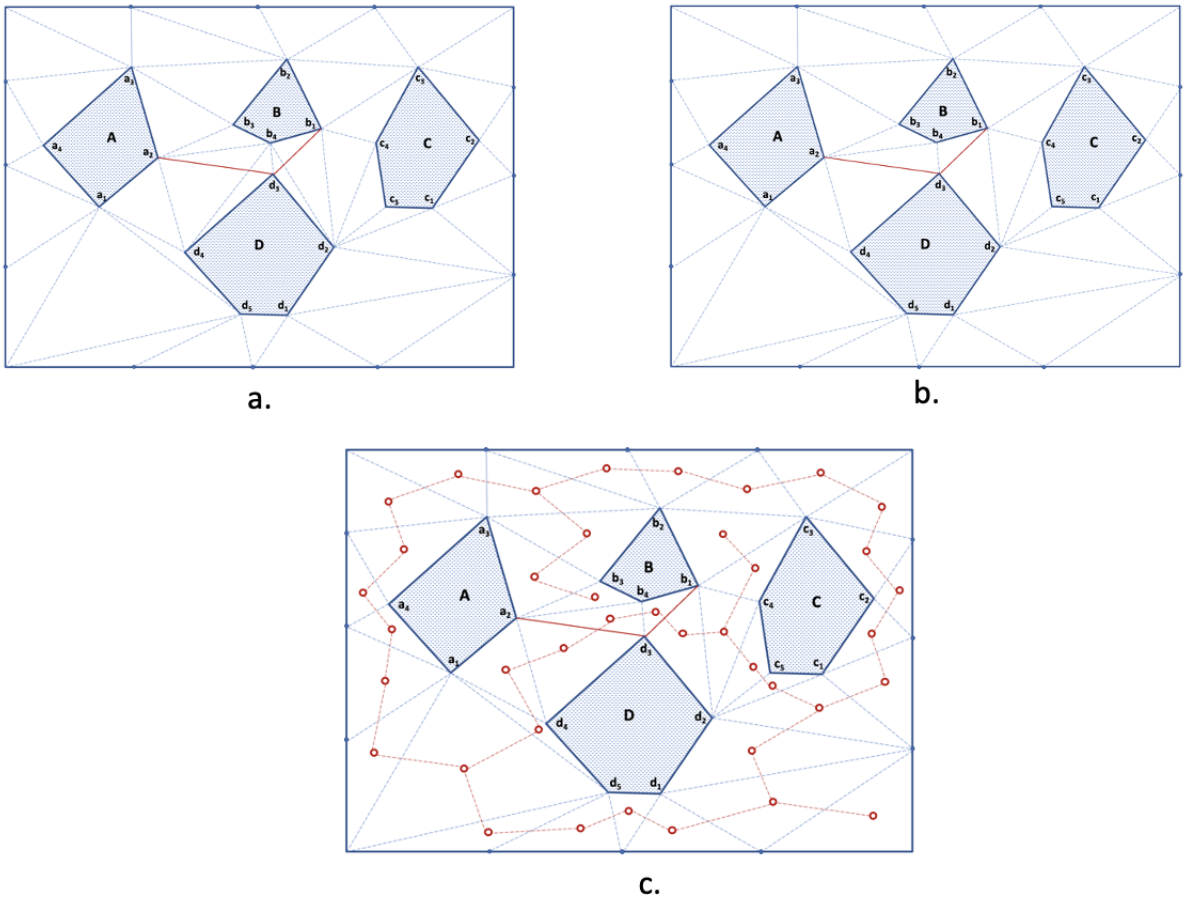


Figure 3.3: Modifying Dual by Flipping

An algorithm for flipping a skinny pair of adjacent triangles can be described as follows. The free space in the presence of polygon obstacles can be triangulated by using the standard triangulation algorithms of polygons with holes [dBvKOS00, O'R98]. The triangulated space is represented in a Doubly Connected Edge List (DCEL) data structure for convenient inspection of triangles. By traversing the data structure, we can identify skinny pairs of adjacent triangles. For these identified skinny pairs, we perform a flip operation as explained in figure 3.2. The resulting triangulation (after all needed flipping) will have a dual that does not intersect with obstacles and has better clearance. A formal sketch of the algorithm is listed as Algorithm 1.

Algorithm 1: Triangle Flipping

Input: Polygonal obstacles P_1, P_2, ... P_q enclosed in a rectangular box R
Result: Triangulation of free space with non-skinny triangles T_1, T_2, ... T_m

Step 1:
Triangulate the free region inside box R. Let the list of the triangles be T_1, T_2, ... T_m

Step 2:
Check the triangles T_1, T_2, ... T_m and mark all adjacent pair (T_i, T_j) that are skinny

Step 3:
for *all adjacent skewed pair* T_i = <*a,f,b* >*and* T_j = <*b,f,g* >*marked in Step 2* **do**
  a) replace T_i = <a,f,b >by <a,f,g >
  b) replace T_j = <b,f,g >by <b,a,g >
end

Step 4:
Output T_1, T_2, ... T_m

Time Complexity Analysis: The time complexity of Algorithm 1 can be done in straightforward manner. Triangulating a polygon with holes can be done in $O(n^2)$ time. Hence, Step 1 takes $O(n^2)$ time. This triangulation can be represented in doubly connected edge list data structure with the same time. Once the triangulation is available in DCEL form, it can be navigated smoothly and skinny triangles can be identified efficiently in linear time. Hence, Step 2 can be done in $O(n)$ provided the triangulation is available in DCEL form. Flipping a pair of adjacent triangles can be done in constant time. There can be at most $O(n)$ skinny pair of triangles. Hence, Step 3 takes $O(n)$ time. Since the time of Step 1 is the dominant time, the total time complexity is $O(n^2)$.

3.3 Method of Expansion

One way to avoid the intersection of the dual with obstacles is to slightly move triangulation vertices away from obstacles. When the vertices of triangles are not coinciding with obstacle vertices and away from obstacles the chance of intersection between the dual and and obstacles is reduced significantly. The idea is to put a small strip around the obstacles and have the vertices of triangles on the strip away from obstacles. This can be better modeled by considering the expansion of obstacles, similar to obstacle growing described in [O'R98].

An image is constructed for boundary edge such that the image edge is parallel to the boundary edge and shifted distance δ away. This is illustrated in figure 3.4. Consider boundary edge (a_1, a_2) of obstacle A. Edge ($a_{1,2}$, $a_{2,1}$) is the image edge for obstacle boundary edge (a_1,a_2). Similarly, the image edge for boundary edge (a_2, a_3) is ($a_{2,2}$, $a_{3,1}$). It is noted that by following this scheme, each

obstacle vertex a_i has its image vertex pair $(a_{i,1}, a_{i,2})$. When all image vertices are connected by straight edges by following the boundary an envelop of width δ is formed which looks like expanded obstacle.

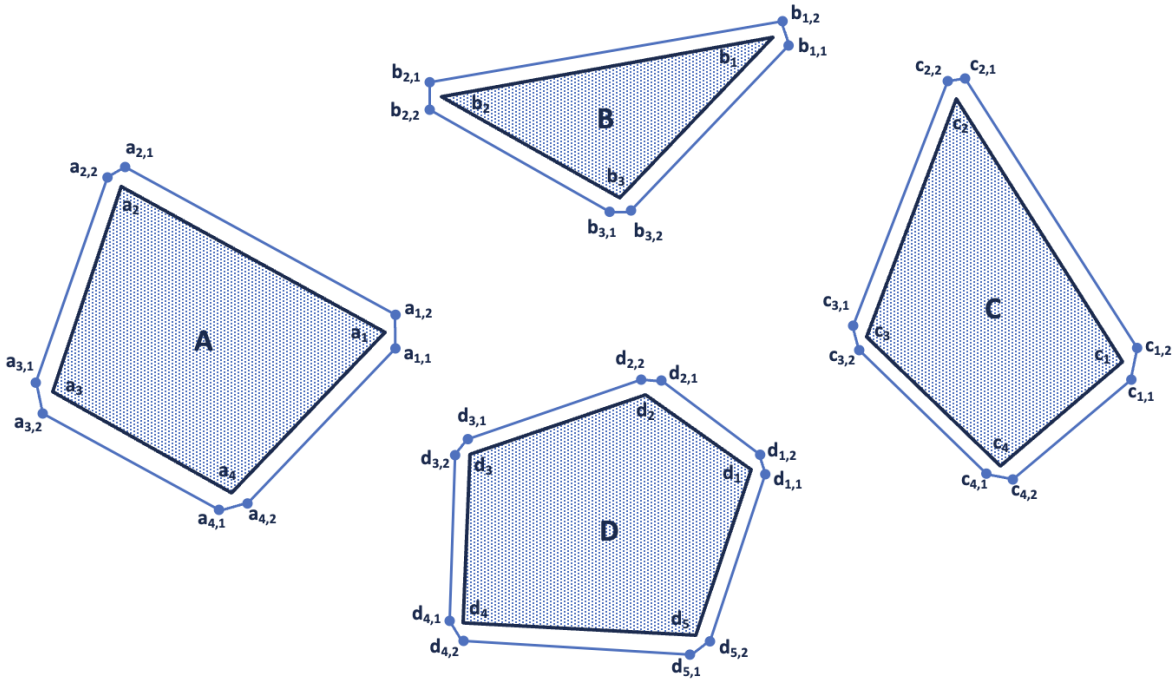


Figure 3.4: Obstacle Expansion

Observe that if an obstacle has m vertices then its envelop will has $2m$ vertices. It is a simple matter to construct image vertices for all obstacle vertices in constant time. Hence the envelop of each of an obstacle with vertices m can be constructed in $O(m)$ time.

Figure 3.5 shows the construction of dual by triangulating the free space by using vertices of the expanded obstacle.

Algorithmic description of the method of expansion can be summarized as follows. A line segment l can be shifted parallel to itself by a distance δ (away from the obstacle) by examining co-ordinates of endpoints, and the slope of l. The line segment $l^{'}$ parallel to l can be created in constant time. The shifted lines can be jointed together to create expanded obstacle. Algorithm 2 is a detailed description of the algorithm used for the expansion method.

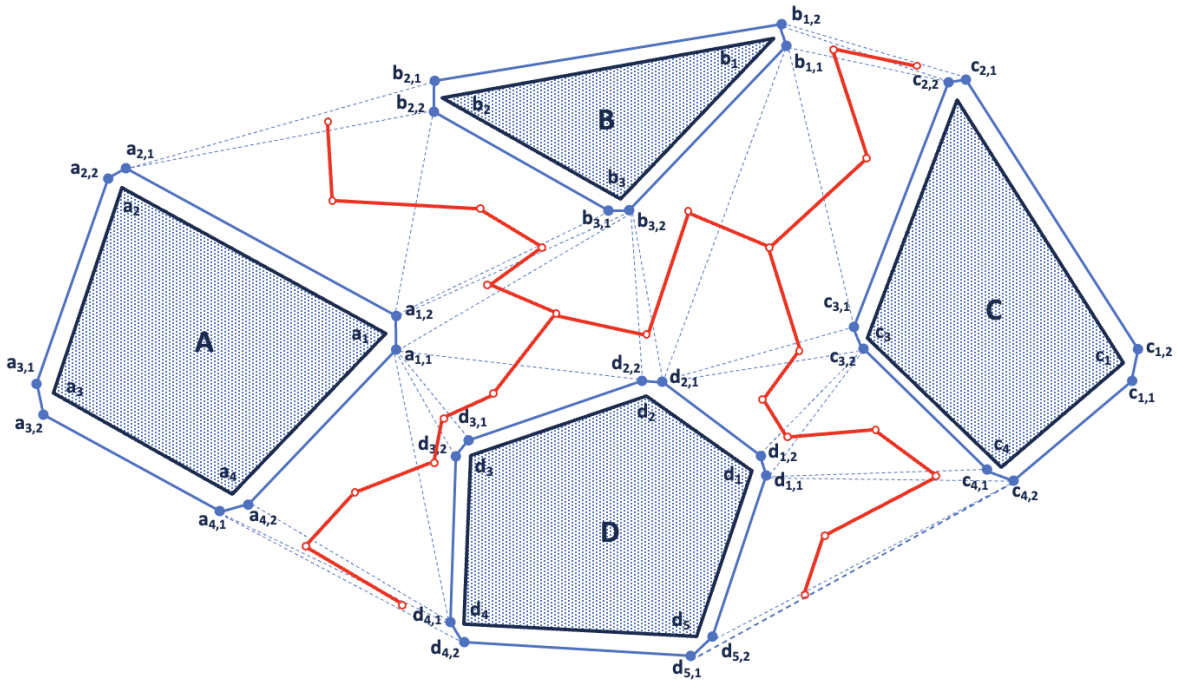


Figure 3.5: Dual After Obstacle Expansion

Algorithm 2: Method of Expansion

Step 1:
for *obstacle* O_1 **do**
| Construct edges parallel to the edges of O_1 and shifted by δ
end

Step 2:
Repeat Step 1 for all other obstacles $O_2, O_3, ..., O_m$

Step 3:
Connect all shifted edges to construct expanded obstacles.

Step 4:
Triangulate free-space outside all expanded obstacles.

Step 5:
Construct dual of the triangulation obtained in Step 4.

Step 6:
Apply Dijkstra's shortest path algorithm in the dual to obtain collision free path.

Time Complexity Analysis: The image edge of each obstacle edge can be done in $O(1)$ time. This implies that Step 1 and Step 2 can be done in $O(n)$, where n is the number of vertices in all obstacles. Connecting shifted edges to construct envelop can be done in $O(n)$ time, which is the time for Step 3. Triangulation of free space can be done in $O(n^2)$ time by using standard

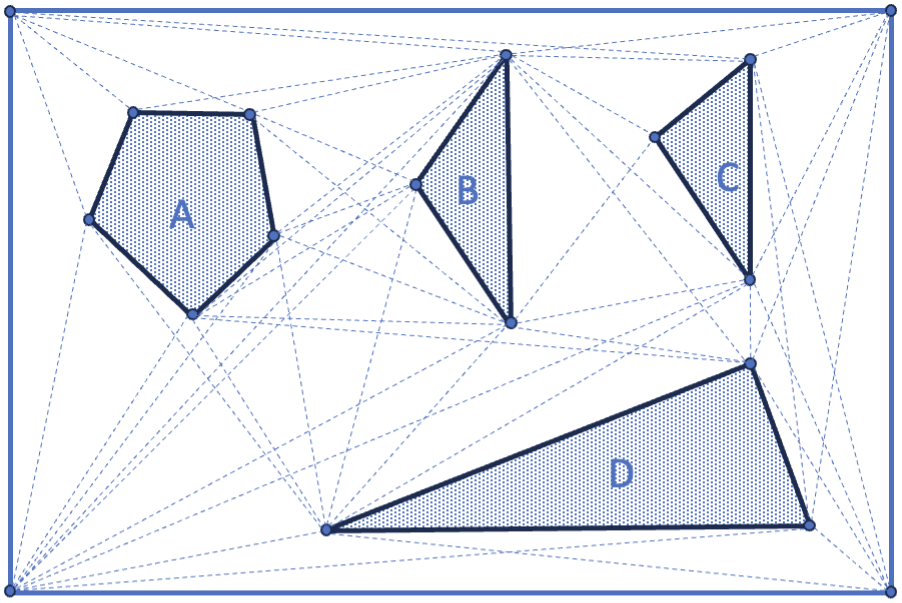
visibility graph structure. Once triangulation is available dual can be constructed in $O(n)$ time if the triangulation is available in DCEL form. Dijkstra's algorithm for Step 6 can be done in $O(n)$ time as the dual is a planar graph. Therefore, the total time complexity is $O(n^2)$.

Picking the value of δ:

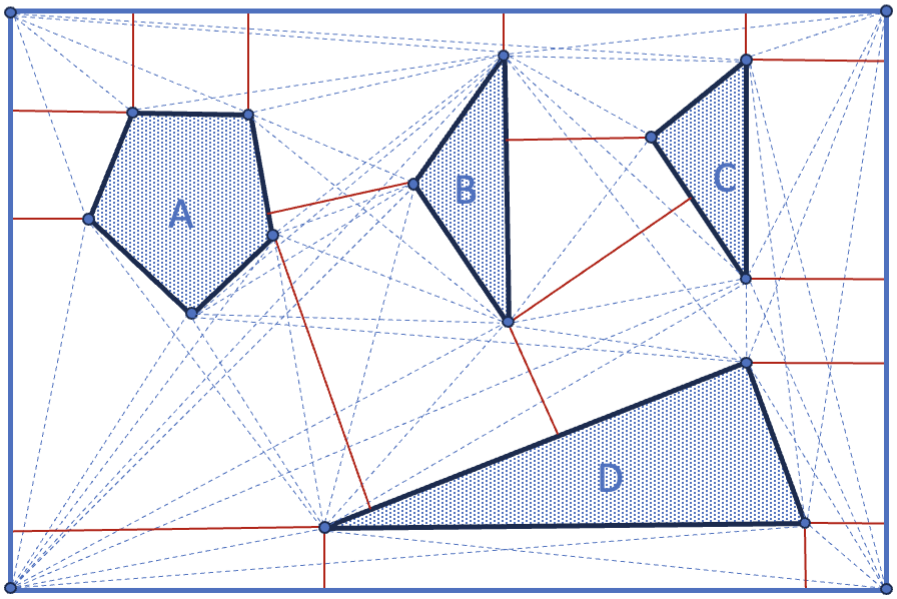
How to pick the value of strip with δ is a critical question. The value of δ should be big enough so that the dual does not close with the obstacles. Furthermore, the value of δ should not be very large to keep envelops of neighboring obstacles not to intersect. One way to address this issue is to examine the closest obstacle to each input obstacle. If min is the closest distance among all closest pairs then we could pick the value of δ as one fourth of the value of min.

One way to find the closest obstacle pair is to construct an extended Visibility Graph [O'R98]. The visibility graph of a collection of obstacles is obtained by connecting all visible pairs of vertices. Figure 3.6 shows an example of visibility graph.

The visibility edges can be extended by adding projection edges. **Projection edges** (Red line in the figure 3.6 the bottom one) are essentially the line segments representing the perpendicular distance between a vertex and an obstacle edge. Note that not all projection edges are shown in this figure.



a.



b.

Figure 3.6: Illustrating Extended Visibility Graph

Chapter 4

Discussion

We reviewed algorithms for constructing collision free paths in the presence of polygonal obstacles. We focused on the method of triangulation pairing for constructing collision-free path with high clearance. We demonstrated that the standard method of the dual of triangulation can sometime construct invalid paths: the paths could have very low clearance from obstacles or in some situation intersect with obstacles. We next presented two algorithms for modifying triangulation of free space so that the dual becomes valid. The first method is based on the technique of triangle flipping and the second method is based on the expansion of obstacle boundaries.

This study gives us insight to explore further methods for modifying triangulation that would increase clearance from obstacle. One such method could be the insertion of steiner vertices near obstacle vertices having sharp turn. This is illustrated in figure 4.1, where an invalid branch of the dual graph is lifted to avoid intersection. Let (c_1, c_2) be the edge of the dual that intersects with obstacle D as illustrated in the figure.

Two steiner vertices (yellow nodes) are inserted near the top vertex of obstacle D. This results in splitting two triangle into four. The centroids of newly created triangles are denoted as g_1 and g_2. Now, if we replace edge (c_1, c_2) by path (c_1, g_1, g_2, c_2), shown by yellow edges, the modified portion of the dual lifts up and the clearance is increased. It would be interesting to explore this lifting mechanism in detail.

It would also be interesting to examine the performance of the proposed algorithms by actual implementation in high level language like Java and/or C++.

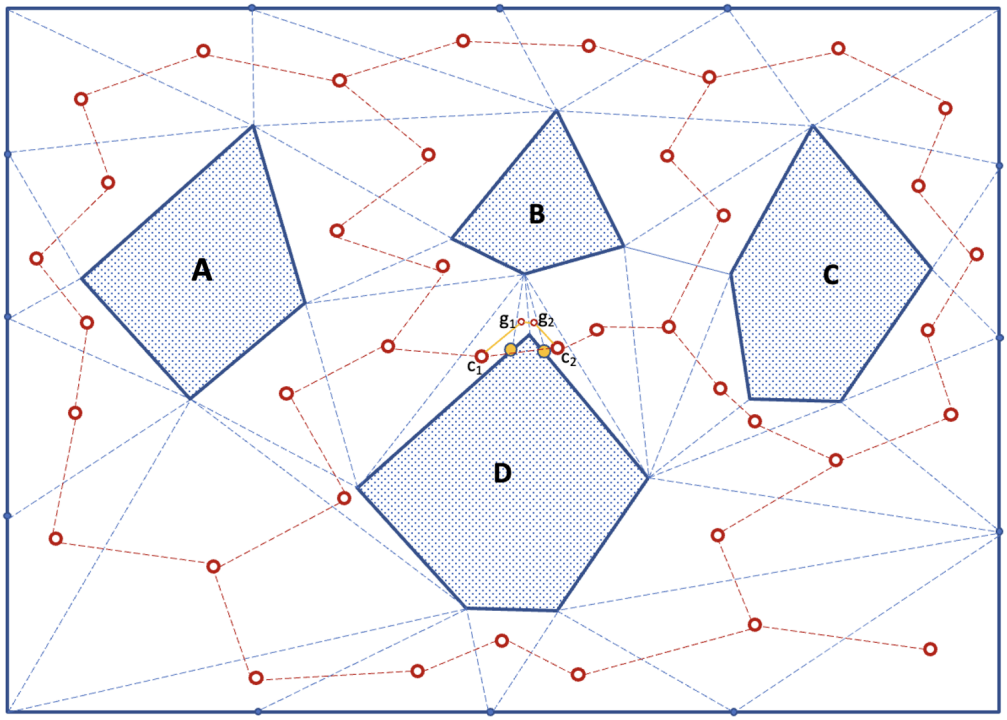


Figure 4.1: Lifting of Intersecting Edge

Bibliography

[Cha91] Bernard Chazelle. *Discrete Computational Geometry*, 6:485–524, 1991.

[dBvKOS00] M. de Berg, M. van Kreveld, M. H. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2 edition, 2000.

[DFS90] D.P. Dobkin, S.J. Friedman, and K.J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5:399–407, 1990.

[GM87] Subir Kumar Ghosh and David M. Mount. An output sensitive algorithm for computing visibility graph. *Proceedings of the 28th Annual Symposium on Foundation on Computer Science*, pages 11–19, 1987.

[Kal05] M. Kallmann. Path planning triangulations. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 49–54, Edinburgh, Scotland, July 2005.

[Lee78] DT Lee. Proximity and Reachablility in the Plane. *Communications of the ACM, PhD Thesis, UIUC*, 1978.

[O'R98] Joseph O'Rouke. *Computational Geometry in C, 2nd ed. New York, NY*. Cambridge University Press, 2 edition, 1998.

[PAZ03] A. Pantaledo, C. Andrade, and E. Zaparoli. A Review of Delaunay Mesh Generation for Heat Transfer Finite Element Analysis. *17th Proceedings of International Congress of Mechanical Engineering*, 2003.

[Tha23] Barun Thapa. MS Thesis. *High Clearance Collision Free Paths*, 2023.

Curriculum Vitae

Graduate College
University of Nevada, Las Vegas

Sandeep Maharjan
sandeepmaharjan55@gmail.com

Degrees:

Bachelor of Engineering in Computer Engineering, 2018
Tribhuvan University

Thesis Title: Triangulation Guided High Clearance Collision-Free Paths

Thesis Examination Committee:

Chairperson, Dr. Laxmi Gewali, Ph.D.
Committee Member, Dr. John Minor, Ph.D.
Committee Member, Dr. Mingon Kang, Ph.D.
Graduate Faculty Representative, Dr. Henry Selvaraj, Ph.D.