2003

Computing the minimum perimeter triangle enclosing a convex polygon: Theory and implementation.

Anna Valentinovna. Medvedeva
University of Windsor

COMPUTING THE MINIMUM PERIMETER TRIANGLE ENCLOSING A CONVEX POLYGON: THEORY AND IMPLEMENTATION

by

Anna V. Medvedeva

A THESIS

Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor
Windsor, Ontario, Canada

Your file Votre référence
ISBN: 0-612-82876-X
Our file Notre référence
ISBN: 0-612-82876-X

ABSTRACT

Geometric optimization, an important field of computational geometry, finds the best possible solution to a computational problem that involves geometric objects. It allows the development of faster and simpler algorithms by virtue of exploiting the geometric nature of the problem. An attractive fundamental problem in this area is one of approximating a convex n-gon with a simpler convex k-gon, where $k < n$, and the area or the perimeter of the approximate object is minimized. This problem arises in a wide range of applications, such as geographic information systems (GIS), spatial databases, pattern recognition, and computer graphics, to name but a few. The approximation of convex polygons with their respective enclosing triangles is a particularly interesting problem; however, finding an optimal linear time solution for computing the minimum perimeter triangle enclosing a convex polygon was a long-standing open problem, which turned out to be more difficult than determining an enclosing triangle of minimal area.

In this thesis, we suggest some theoretical and practical justifications for the linear time complexity of a recently proposed optimal solution and provide an efficient and robust implementation of that solution. We derive, analyze, and solve for various instances an algebraic equation that expresses the perimeter of the circumscribing triangle in terms of the geometric configuration of the enclosed object for the main subsidiary problem variant. In this problem, such an enclosed object is defined by a pair of points above a given line. The existence of this algebraic expression and its solution further suggest the existence of a linear solution to the original problem. Our object-oriented implementation of the algorithm relies on several essential geometric preliminaries, which were mathematically derived to provide for higher efficiency. The proposed implementation is complete and robust in the sense that it will work for all input instances. We have performed several software testing techniques to guarantee this claim. Our result is asymptotically more efficient than previous solutions for the same task.

DEDICATION

To my husband and friend, Yuriy Y. Pastyrskyy, for his love, support, understanding, and endless help.

ACKNOWLEDGEMENTS

There are many people I would like to thank for their help during my studies at the University of Windsor. First, I wish to express gratitude and appreciation to my graduate supervisor, Dr. Asish Mukhopadhay, for his careful and kind guidance, dedicated instruction, and thought-provoking discussions during our collaboration and my studies at the University of Windsor. His valuable advice and fruitful suggestions have not only contributed to the quality of my Master's thesis work, but have motivated me to become a more careful, creative, and inquisitive researcher, and provided an inspiration for my future endeavors. Dr. Mukhopadhyay, "a computational geometer who does absolutely first class work" [WWW3], has greatly impressed me with the quality of his research. Indeed, without his supervision and guidance, this thesis work would have not been possible.

Next, I would like to thank the Internal and External Readers from my Thesis Committee, Dr. Liwu Li and Dr. Myron Hlynka, for having the patience to read the manuscript and for their attentive and helpful comments and suggestions. Their understanding, knowledge, and feedback have helped me a great deal while working on the final version of this document.

My appreciation goes to the Thesis Committee's Chair, Dr. Peter Tsin, who equipped me with the knowledge in the subject of Algorithms through his comprehensive 60-454 course, Design and Analysis of Algorithms, and who has provided various support during my second year of studies at the University of Windsor.

My most heartfelt thanks go to my beloved husband, Yuriy Y. Pastyrskyy, for taking interest in what I do, for his love and understanding, for his support and sacrifices, and for the enormous help that he had provided to me during our 2.5-year stay in Canada. I am also grateful for all our discussions related to the subject of this thesis, being it theoretical work or implementation details. I thank Yuriy for all his brainstorming ideas and thoughtful suggestions that have inspired many creative solutions.

Special thanks to my father-in-law, Prof. Yaroslav A. Pastyrskyy, for his prompt and valuable responses to my mathematical questions, for his dedicated and patient support, and for his frequent e-mails from home.

Finally, I want to thank both families, my husband's in Lviv, Ukraine and mine in Seattle, the United States, for all their prayers, support, and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

List of Appendices

Chapter 1 INTRODUCTION

1.1 COMPUTATIONAL GEOMETRY AND OTHER COMPUTING SCIENCES

Computational geometry is having an ever-increasing influence on various branches of computing and information sciences. Database systems, pattern recognition, geographic information systems, and computer graphics are just a few examples of the fields where the procedures can be made more rigorous and the algorithms more efficient by the application of principles discovered in computational geometry.

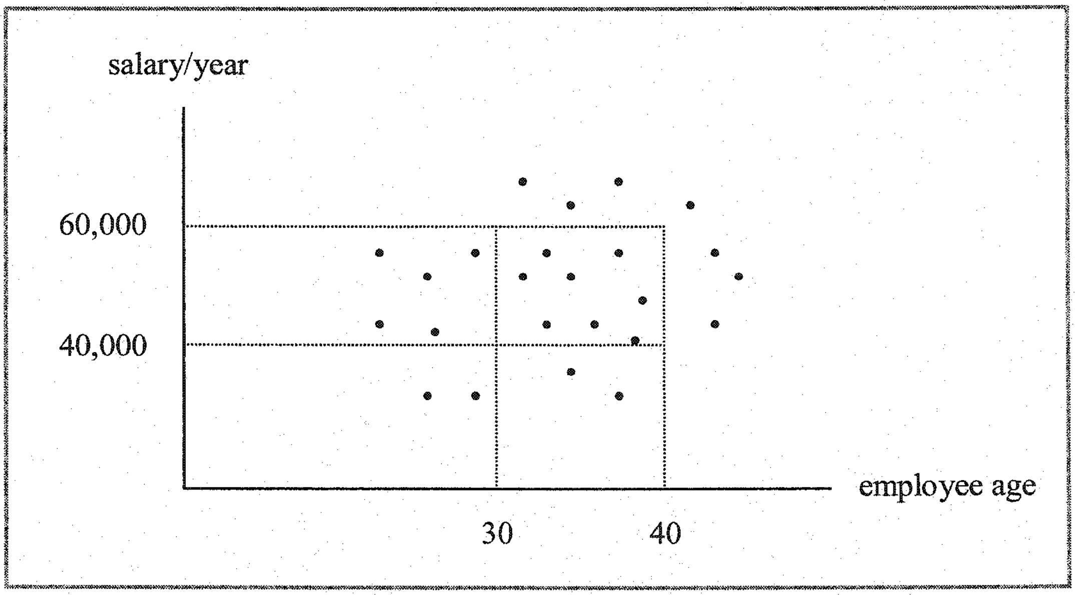


Figure 1.1 Geometric interpretation of a database query

Consider problems in the area of database systems that, at the first glance, have not much in common with computational geometry. Nevertheless, many types of database queries can be formulated geometrically. An example of a typical range query in a relational database is to retrieve all employees with yearly wages between 40,000 and 60,000 and who are between 30 and 40 years of age (Figure 1.1). A record, or a tuple, in a database that has *d* attributes can be interpreted as a point in *d*-space (multidimensional space). Then, queries that ask to report all records that lie in a certain query interval transform into reporting all the points lying in the corresponding *d*-dimensional block. The query region in this case is rectangular, and such rectangular range queries belong to orthogonal range searching [BKOS97].

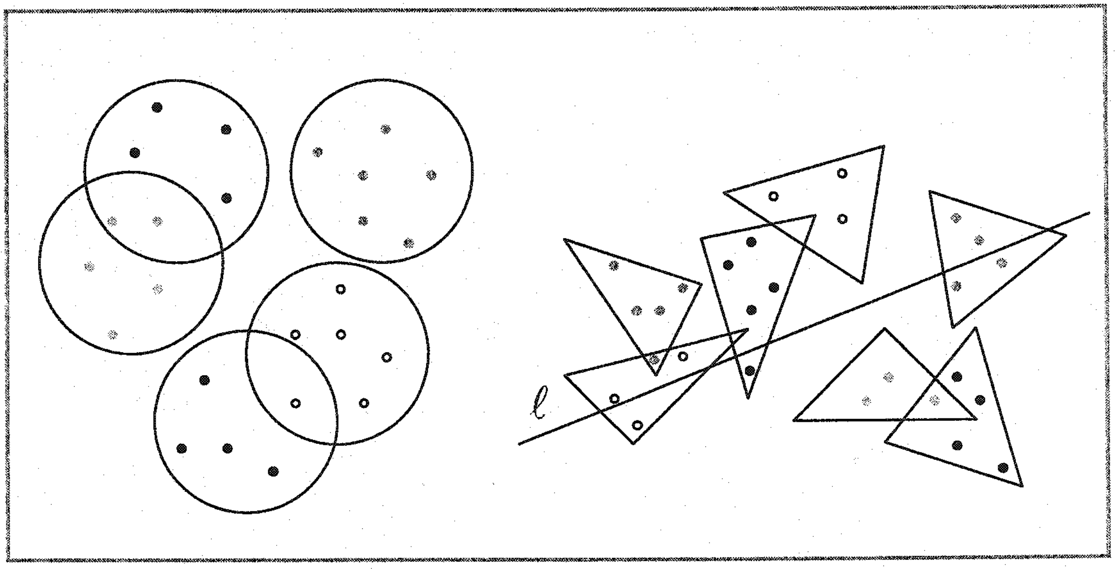


Figure 1.2 Examples of simplicial partitions (with disks, triangles) and a simplex range query (ℓ is a query line)

While simple rectangular range queries can be answered without involving the computational geometry techniques, answering range queries over non-rectangular regions, called simplices (see examples in Figure 1.2), can be difficult. The area of simplex range searching deals with the queries of more general structure, when the query range is of non-orthogonal form. Let us consider the following example [BKOS97]. The city authorities are planning to build a new airport. The population density can be represented on a map by plotting a point for every, say, 5000 people. Then, the query of how many people would be affected by the new airport construction (say, in terms of the noise level from the planes that exceeds a certain level), would transform into reporting the number of points in the query region around the new construction. The query region in this case is non-orthogonal and can generally be of any shape (the term "simplex" is typically described as "simple polygon"; however, the shape of the region can be more flexible). Simplex range searching deals with preprocessing of points into a data structure such that the points lying in a query simplex can be counted, or reported, efficiently.

Pattern recognition is another area that is broadly influenced by the advances in computational geometry. For instance, convex hulls and the algorithms for their selection, sorting, and computing have been extensively used in many pattern recognition problems [Tou85]. Voronoi diagrams have also been used to efficiently solve some geometric problems that arise in pattern recognition and other fields. Another geometric problem that occurs in pattern recognition and that was solved in computational geometry is a problem of decomposition of simple polygons into convex components (for example, the problem of decomposition of a non-convex polygon into a minimum number of convex polygons [KS85]).

One of the most effective ways of obtaining efficient algorithms (in many computing fields) is application of so-called bucketing techniques, also discovered in computational geometry (for details, see a survey [AEIIM85]). "Buckets" are the subregions into which the entire region under consideration is partitioned. The bucketing techniques are applicable, for example, to the problem of minimum-weight perfect matching in the plane, which asks to match the *n* points in pairs so as to minimize the sum of the distances between the matched points.

These and many other problems that find their solutions in computational geometry are used in pattern recognition and other computing fields.

1.2 Concepts of Geometric Optimization in Computational Geometry

In general, there are two kinds of computing problems:

1. An *optimization problem* asks, what is the best solution? An example is the well-known traveling salesman problem, which minimizes the cost of overall travel.
2. A *decision problem* asks, is there a solution with a certain characteristic? An example is a Hamiltonian cycle with a cost less than some fixed cost *k*.

Geometric optimization problem is an optimization problem induced by a collection of geometric objects (e.g. [FS75], [DB83]). The process of finding the best possible solution to such a computational problem is called geometric optimization. The objective of geometric optimization is to find an optimal solution with the minimum (or maximum) value in a region that contains all possible problem solutions.

Geometric optimization allows the development of faster and simpler algorithms by virtue of exploiting the geometric nature of the problem (see [AS98] as well as the examples in section 1.1). This is because often a non-geometric problem can be reduced to a geometric one with a simpler and more efficient solution (e.g. orthogonal range searching for relational database querying).

1.3 The Problem of Approximation of Convex Polygons

The problem of finding a simpler convex polygon (or polytope in higher dimensions) that approximates a given convex polygon (polytope) [Gru83] has been studied extensively in the past two decades. Being fundamental in nature, this problem, nevertheless, finds various practical applications, discussed in section 1.5. In its general case, this problem can be described as circumscribing and inscribing convex n-gons with, respectively, minimum and maximum area (or perimeter) k-gons. Typically, $k < n$, and the resulting approximate polygon is, in its geometric sense, simpler than the original polygon.

There are, naturally, two "flavors" of this general problem, where one deals with finding the circumscribing polygon and the other attempts to find the inscribing polygon. Both the enclosure and inclusion problems can be viewed as polygon approximation problems. The existing approaches for solving both types of problems are briefly discussed in Chapter 2 (section 2.1).

1.4 The Problem of Finding the Minimum Perimeter Triangle Enclosing a Convex Polygon

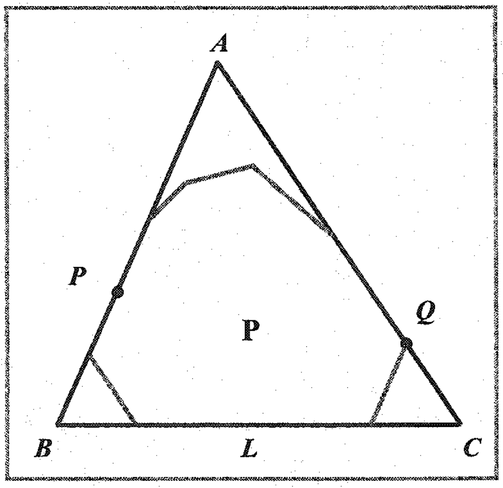


Figure 1.3 A triangle that circumscribes a convex polygon **P**

A special case of the polygon approximation problem is a problem of finding a triangle of minimal perimeter that circumscribes a given convex polygon (Figure 1.3). This problem constitutes the major subject of this thesis and will be investigated in detail in subsequent chapters. The geometric significance of points P and Q in Figure 1.3 will also be discusses in detail.

An optimal linear time solution to the minimum perimeter triangle problem was recently suggested by Bhattacharya and Mukhopadhyay [BM01], [BM02].

1.5 Problem Applications

Solutions to the polygon approximation problems find their applications in various practical settings. The following list provides some of these important applications.

1. Stock-cutting problems in manufacturing (cut a sheet of material into smaller subparts under various constraints, such as all subparts are congruent to a given shape),
2. Packing and optimal layout problems,
3. Collision avoidance in robotics,
4. GIS (geographic information systems) that maintain maps and spatial data for the environment, military, and city planning,
5. Spatial databases,
6. Pattern recognition,
7. Scientific computing and visualization,
8. Computer graphics,
9. Data compression.

1.6 Computational Model and Other Assumptions

Our computational model for the problem of finding the minimum perimeter triangle enclosing a convex polygon assumes a random-access computer with infinite precision real arithmetic. While this assumption is permissible for the theoretical solution, it will not be appropriate for the implementation of the theoretical result. We will address this issue later when we discuss robustness of our implementation in Chapter 6 (section 6.6).

We further assume that the solution of simple algebraic and trigonometric equations takes constant time, $O(1)$. In addition, we presume that all geometric primitives that are computed in the course of the algorithm (such as cross points, antipodal points, and excircles), which, in general, may involve solving several algebraic equations of various degrees, can also be obtained in constant time.

Finally, our computational model for the algebraic solution to the main subsidiary problem, discussed in Chapter 4, assumes that the algebraic equations involving polynomials of degree higher than four can also be solved in constant time. While this assumption is not accurate in the general sense, it nonetheless may be applied here, since we can simply view such high-degree polynomials as a "black box", and apply the power of, say, numerical methods, to provide the solution(s) in an efficient manner.

1.7 MOTIVATION FOR THE THESIS

Our motivation for this thesis is two-fold.

First, it has been evident for some time that in several practical situations, the best algorithm from the theoretical point of view is prone to be outperformed by more naïve methods. That is why, claiming an efficient algorithm with optimal or near-optimal time complexity is not always sufficient for the purposes of solving a practical problem. If we wish the results of our research to be of some practical use, we should not forget to investigate the efficiency of our solution in a practical setting. We considered implementing a novel linear time algorithm for computing the minimum perimeter triangle enclosing a convex polygon due to Bhattacharya and Mukhopadhyay [BM01], [BM02] in order to offer an efficient practical solution to this problem, in addition to achieving an optimal theoretical result.

Second, it has been unknown to us whether the number of iterations necessary to compute a minimum perimeter triangle corresponding to each polygon edge is finite and whether it depends (and if so, how) on the number of polygon edges n. In its degenerate case, the problem of finding the minimum perimeter triangle enclosing a convex polygon can be construed as a problem of finding the minimum triangle configuration for an object defined by a pair of points above a given line (Chapter 4). We considered solving this main subsidiary problem variant algebraically in order to verify that a solution to the minimum perimeter triangle problem can be expressed and solved mathematically. We also used our implementation of the algorithm to verify in many problem instances that the number of iterations mentioned earlier is not only finite, but also is small enough to consider this solution to be of practical importance.

1.8 OBJECTIVES OF THE THESIS

First, this thesis aims at providing an efficient and robust implementation of a novel linear time algorithm for computing the minimum perimeter triangle enclosing a convex n-gon. The implementation should be *efficient* in the sense that it should comply with the algorithm's linear time complexity while achieving a small constant factor. To attain this, all the preliminary geometric operations used in the algorithm should be computed efficiently, that is in constant time. The implementation should be *robust* in the sense that it should work for all input instances. The efficiency and robustness are necessary for providing a solution of practical use.

Second, this thesis aims at solving the minimum perimeter triangle problem for the main subsidiary case algebraically. Expressing the perimeter of the enclosing triangle via the geometric configuration of the enclosed object that is defined by a pair of points above a line, analyzing this expression, and providing the solution for various instances is the second goal of the thesis. The characterization of the perimeter function in the mathematical sense is important for better understanding of the problem and for analyzing the approaches to finding its best solution.

1.9 STRUCTURE OF THE THESIS

In Chapter 1 we provide an introduction to the area of computational geometry and its field of geometric optimization. We also introduce the problem of approximation of convex *n*-gons with simpler convex *k*-gons. A particular problem of finding the minimum perimeter triangle enclosing a convex polygon is also introduced in Chapter 1.

Chapter 2 gives a review of the literature for the problems of computing the minimum area and minimum perimeter triangles that enclose convex polygons (sections 2.2 and 2.3, respectively). A brief literature overview regarding more general polygon enclosure and inclusion problems (where a triangle is extended to a general *k*-gon), is provided in section 2.1.

In Chapter 3, we present our theoretical framework for computing the minimum perimeter triangle enclosing a convex polygon. This includes an overview of a recently-proposed linear time algorithm, proofs for the minimum perimeter triangle configuration conditions, and a mathematical derivation of many geometric operations used in this algorithm.

Chapter 4 concludes the theoretical portion of the thesis by describing an algebraic approach to solving the main subsidiary case of the minimum perimeter triangle problem. It provides an equation that expresses the perimeter of the circumscribing triangle via the geometric configuration of the enclosed object, defined by two points above a line. Minimizing this expression for the perimeter gives a solution to the main subsidiary problem.

Chapter 5 discusses an object-oriented methodology for our implementation of the linear time algorithm for the problem at hand. This includes OOA and OOD (object-oriented analysis and design), OOP (object-oriented programming) in C++, as well as a discussion of several software testing techniques applied to the resulting software system.

Chapter 6 addressed the implementation in more detail. In addition, here, we concentrate on such important issues of our software system as efficiency and robustness.

Chapter 7 evaluates the framework proposed in the thesis, including theoretical and implementation results.

Finally, Chapter 8 provides a conclusion and projects the directions for future work.

Chapter 2 An Overview of Existing Approaches

2.1 The Problem of Approximation of Convex Polygons: A Brief Overview of the Literature

Recall from Chapter 1 (section 1.3) that there are two general types of polygon approximation problems: the inclusion and enclosure problems.

Let us first consider the inclusion problem.

The *potato-peeling problem* asks to find the largest convex polygon contained inside a given simple polygon. A *potato* of polygon $\mathcal{P}$ is the maximum area (or perimeter) convex polygon contained in $\mathcal{P}$. The potato-peeling problem was solved by Chang and Yap [CY84] for a general case and for the case when the desired polygon was maximized with respect to perimeter. The proposed algorithms have $O(n^7)$ and $O(n^6)$ time complexities, respectively.

Dobkin and Snyder [DS79] considered the potato-peeling (inclusion) problem, when the polygon included inside a convex polygon was a triangle of maximal area. They proposed a linear time algorithm for this case. Later, their result was extended from triangles to convex k-gons by Boyce *et al.* [BDDG85]. Boyce *et al.* obtained algorithms for both, maximal area and perimeter cases. The running time of these solutions is $O(k \cdot n \log^2 n)$. The main idea behind these algorithms is that the vertices of any maximal k-gon must be a subset of the vertices of the input polygon. In both, [DS79] and [BDDG85], the input polygon is restricted to be convex. Thus, the techniques proposed in these papers do not work for a more general case of simple polygons. The general potato-peeling problem was solved in [CY84], as it was mentioned earlier.

The enclosure problem has been considered for the following cases.

The problem of enclosing a convex polygon with a triangle of minimal area was first considered by Klee and Laskowski [KL85]. They derived an $O(n\log^2 n)$ solution. Later, O'Rourke *et al.* [OAMB86] improved it to linear time, which is optimal. We will give a more detailed discussion of these approaches in section 2.2. Also, we will provide a comprehensive coverage of the problem investigated in this thesis; that is, finding the minimum perimeter enclosing triangle for a given convex polygon (beginning in section 2.3 and throughout this thesis).

De Pano [DeP84] extended the method described in [KL85] from triangles to convex *k*-gons. He obtained a solution for all *k*. His solution's time complexity is exponential in *k*: $O(n^{k-2}\log^2 n)$. Later, Chang and Yap [CY84] improved De Pano's result to $O(n^3\log k)$. Finally, Aggarwal *et al.* [ACY85] refined the latter solution to $O(n^2\log n\log k)$.

In addition to triangles and general *k*-gons, the enclosure problem was also approached for enclosing rectangles and squares. For the problem of finding the smallest rectangle containing a given polygon, Toussaint [Tou79] proposed a linear time solution. In generalization to three dimensions, O'Rourke [ORo84] described an $O(n^3)$ algorithm for the smallest rectangular box enclosing a given polyhedron. The problems related to finding the smallest square containing a given polygon were addressed in the paper by De Pano and Aggarwal [DA84].

2.2 Finding the Minimum Area Triangle Enclosing a Convex Polygon

2.2.1 Problem Statement

Given a convex polygon $\mathcal{P}$, find a triangle $\mathcal{T}$ that encloses $\mathcal{P}$ and has the minimum area among all triangles enclosing $\mathcal{P}$.

2.2.2 The First Solution

Klee and Laskowski [KL85] provided the first solution to the problem of finding a triangle of minimal area that encloses a convex polygon. They derived an $O(n\log^2 n)$ algorithm that finds all such triangles. In general, there exists more than one triangle (a family of triangles) with the minimum area property.

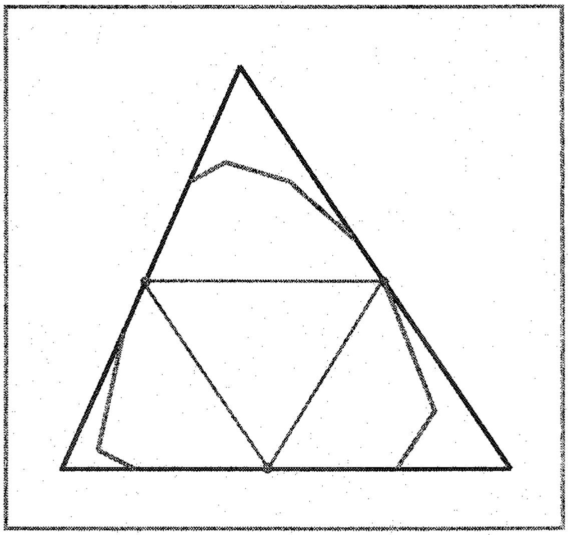


Figure 2.1 Finding the minimal area enclosing triangles

The first and subsequent approaches for finding the minimum area enclosing triangles are based of the following two heuristics discovered in [KL85] (see Figure 2.1).

1. First, the midpoint of each side of the enclosing triangle $\mathcal{T}$ touches polygon $\mathcal{P}$.
2. Second, at least one side of triangle $\mathcal{T}$ is flush with an edge of polygon $\mathcal{P}$.

The strength of the first solution to the minimum area problem lies in proving these important properties, as well as in establishing an elegant geometric characterization of the minimal area enclosing triangles, which permits the avoidance of brute-force optimization. The authors show that although there may be infinitely many local triangle minima, they fall into at most *n* equivalence classes, each of which is a "segment" of triangles having the same area [OAMB86]. In other words, there are at most $O(n)$ regions that contain the local minima. Finding these local minima takes $O(n\log^2 n)$ time; then, selecting the global minima from the local ones is accomplished in additional $O(n)$ time, making the overall time complexity of this solution $O(n\log^2 n)$.

2.2.3 AN OPTIMAL SOLUTION

O'Rourke *et al.* [OAMB86] improved the solution of Klee and Laskowski [KL85] to linear time, and showed that $O(n)$ is the optimal time complexity for this problem. The algorithm suggested by O'Rourke *et al.* finds all minima and just one minimum in $O(n)$ time.

The linear time solution is built on the same heuristics (midpoint and flush edge) as the first solution; however, the researchers avoid computing each local minimum afresh, without using any information gained from finding the previous local minima [OAMB86]. They move from one minimum to the next in an orderly fashion, which allows them to achieve the linear time complexity.

2.3 Computing the Minimum Perimeter Triangle Enclosing a Convex Polygon

2.3.1 Problem Statement

Given a convex polygon $\mathcal{P}$, find a triangle $\mathcal{T}$ that encloses $\mathcal{P}$ and has the minimum perimeter among all triangles enclosing $\mathcal{P}$.

2.3.2 The First and Subsequent Solutions

The first solution to the minimum perimeter problem was proposed by De Pano [DeP87]. The time complexity of this algorithm is $O(n^3)$. An important contribution of the paper is establishing the following property (similar to the minimal area enclosing triangles):

Every convex *n*-gon, where *n* is the number of polygon edges, has a minimum perimeter circumscribing triangle with at least one side flush with a polygon's edge.

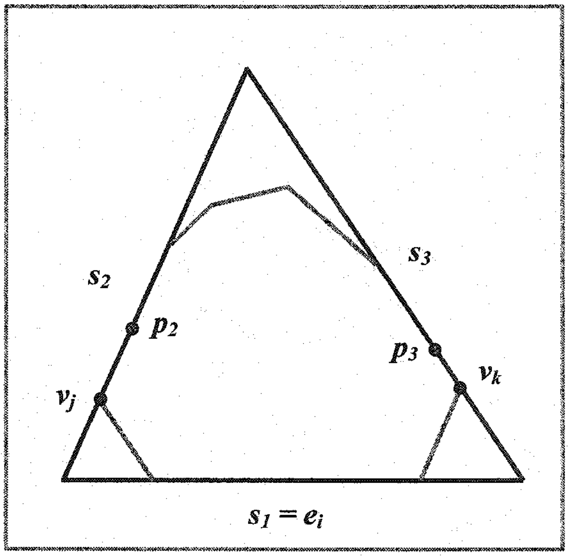


Figure 2.2 Finding the minimal perimeter enclosing triangle

De Pano suggested the following framework for finding the minimum perimeter triangle enclosing a convex polygon. For any triple (i, j, k), let $T_{i,j,k}$ be the minimum perimeter circumscribing triangle with its first side s_1 flush with the polygon's edge e_i, its second side s_2 containing vertex v_j of the polygon, and its third side s_3 containing vertex v_k of the polygon (Figure 2.2). Then, there exists a point p_2 on the triangle's side s_2 and the polygon, and the point p_3 on the triangle's side s_3 and the polygon, such that the length of s_2 between point p_2 and the end-point it shares with s_1 equals the length of s_3 between point p_3 and the end-point it shares with s_1 [AP88]. This approach allows computing such $T_{i,j,k}$ triangles in constant time. There are n^3 possible triples (i, j, k) to be considered and, therefore, De Pano's algorithm runs in $O(n^3)$ time.

De Pano's complexity was later improved to $O(n^2)$ by Chang and Yap [CY84]. The latter approach is based on the principle discovered for computing the minimum area enclosing triangles, which is also applicable to the minimum perimeter problem: there are at most $O(n)$ regions that contain local minima. In a given region, it takes $O(1)$ time to determine the local minima. Each region is "two-dimensional", since there are $O(n)$ regions corresponding to each polygon edge and a total of n edges. A region is thus denoted as *Region(i, j)*. Therefore, the overall time complexity of this solution is $O(n^2)$.

Aggarwal and Park [AP88] used the powerful *matrix searching* technique in higher dimensions to reduce the complexity of the minimum perimeter triangle problem to $O(n\log n)$. The matrix searching technique is based on searching in sorted matrices. The main idea of this approach is that a set of candidate critical values can be represented by an $n \times n$ matrix (in two dimensions), where each row and column is sorted. Then, the approach recursively eliminates sub-matrices with certain properties, finally leaving n sub-matrices. The binary search is then performed on the remaining n candidates in order to find the final minimum or maximum answer. The running time of the matrix searching technique is $O(\log n)$ times the cost of the decision problem. The minimum (or the maximum) element of every row of a *totally monotone* matrix can be found in linear time. In a totally monotone matrix (see a fragment in Figure 2.3), $a_{i_1 j_1} < a_{i_1 j_2} \Rightarrow a_{i_2 j_1} < a_{i_2 j_2}$, where a is an element of the matrix and i and j are the row and column matrix indices respectively.

$$\begin{array}{c|c|c|}
 & j_1 & j_2 \\
\hline
i_1 & a_{i_1 j_1} & a_{i_1 j_2} \\
\hline
i_2 & a_{i_2 j_1} & a_{i_2 j_2} \\
\hline
\end{array}$$

Figure 2.3 A fragment of a totally monotone matrix

As it was mentioned above, Aggarwal and Park used the matrix searching technique to derive their $O(n \log n)$ solution to the minimum perimeter triangle problem. Their algorithm creates an $n \times (2n-2) \times (2n-1)$ matrix $A = \{a_{ijk}\}$ that is totally monotone and is defined as follows. For $1 \leq i \leq n$ and $i < j < k < i+n$, let a_{ijk} be the perimeter of triangle $T_{i, j \bmod n, k \bmod n}$, provided this triangle exists [AP88]. If this triangle does not exist, a_{ijk} is set to infinity. According to De Pano's cubic solution [DeP87], each entry in matrix A can be computed in constant time. And since the perimeter of the minimum perimeter triangle for the polygon is simply the minimum entry in A, it is possible to find the solution to the problem in $O(n \log n)$ time, based on the fact that matrix A is totally monotone.

It is important to note that even a powerful matrix searching technique that often provides linear solutions, did not yield a linear time answer to the minimum perimeter triangle problem. Finding a linear time solution to this problem turned out to be more difficult than finding an optimal linear time solution to the minimum area problem.

The existence of a linear time algorithm had been a matter of conjecture for over 10 years.

2.3.3 AN OPTIMAL SOLUTION

Finally, Bhattacharya and Mukhopadhyay [BM01], [BM02] proposed a linear time algorithm for solving the minimum perimeter triangle problem. Their solution will be discussed in great detail in the next chapter (Chapter 3).

Chapter 3 OUR FRAMEWORK FOR COMPUTING THE MINIMUM PERIMETER TRIANGLE ENCLOSING A CONVEX POLYGON

3.1 A LINEAR TIME ALGORITHM OVERVIEW

We begin with providing a detailed overview of a linear time algorithm for computing the minimum perimeter triangle enclosing a convex polygon of *n* edges/vertices due to [BM01] and [BM02]. Our algorithm description will be based on the paper [MM03]. In the discussion that follows (all remaining thesis chapters), a *polygon* refers to a *convex polygon*.

It was established in [DeP87] that a minimum perimeter triangle that circumscribes a convex polygon is flush with at least one of its edges (Figure 3.3).

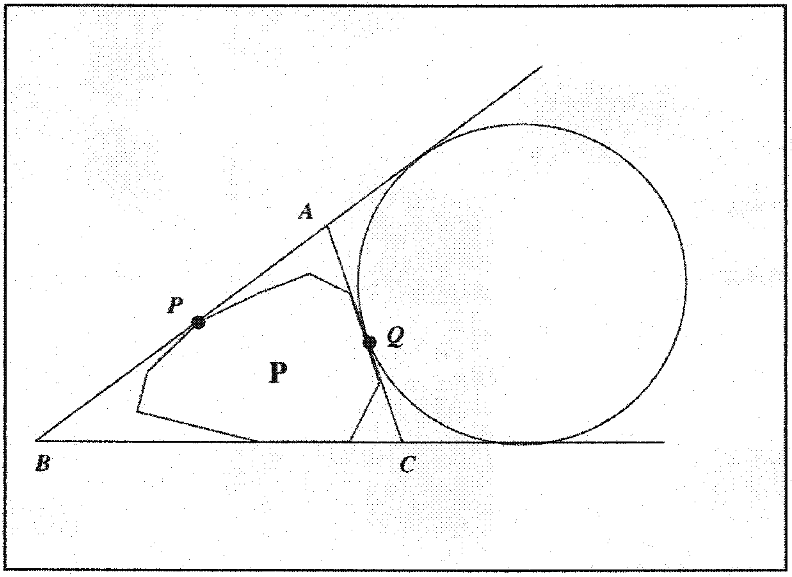


Figure 3.1 Fitting a circle into a wedge from the right

The main idea of the algorithm is to consider each polygon edge in turn and compute a minimum perimeter triangle that is flush with this edge. This computation is built on a novel scheme of *circle-fitting* and *wedge-flipping*. The former consists of fitting the smallest circle into a wedge that contains the given polygon as shown in Figure 3.1. Once such a circle is determined, we also have a new wedge that contains the polygon as shown in Figure 3.2, and we repeat the previous step with this new wedge. This is *wedge-flipping*.

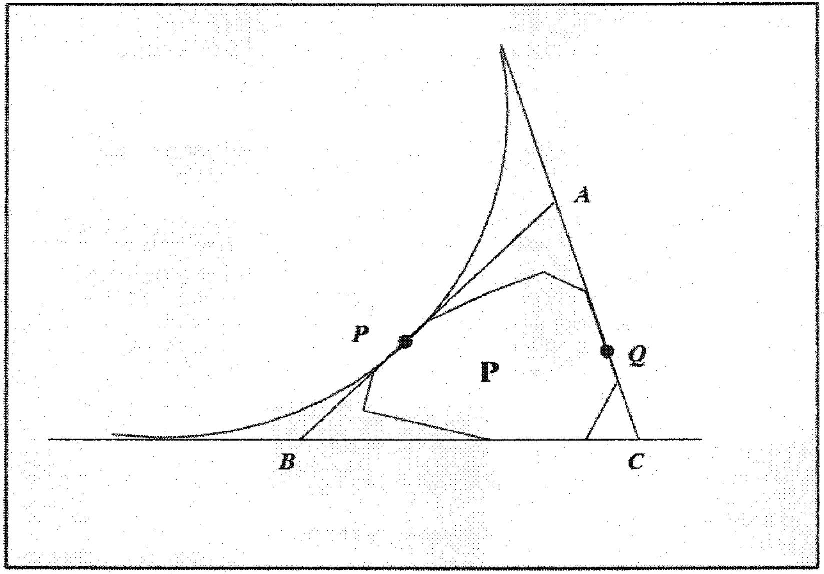


Figure 3.2 Fitting a circle into a wedge from the left

It was proved in [BM01] and [BM02] that in each step, or *iteration*, we reduce the perimeter of the enclosing triangle. We stop when we obtain an enclosing triangle ΔABC such that $BP = CQ$, where P is the point where the circle fitted into the wedge *W(CA, CB)* touches AB, and Q is the point where the circle fitted into the *wedge W(BA, BC)* touches AC (see Figure 3.3).

The circle-fitting procedure makes use of a solution to the following basic problem.

Problem 1 Given a wedge *W(BA, BC)* and a point *Q* contained in it, find a triangle of minimum perimeter that has two of its sides along the arms of the wedge and the third side incident on the given point *Q*.

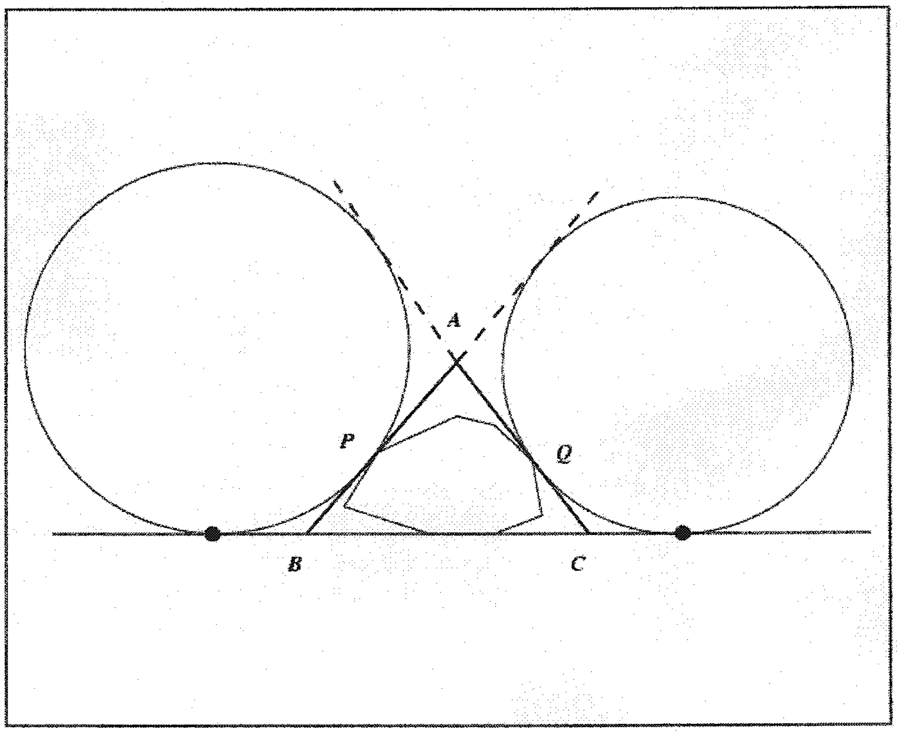


Figure 3.3 A minimum perimeter configuration for a given edge

When the circle-fitting and wedge-flipping procedure terminates, we have a triangle of minimum perimeter. The proof of the former makes fundamental use of the fact that the solution to the following problem (Problem 2) is a unique one and can also be found by circle-fitting and wedge-flipping.

Problem 2 Given two points *P* and *Q* at heights h_P and h_Q respectively, $h_P \geq h_Q$, above a line *L*, *P* to the left of *Q*, find a Δ *ABC* of minimum perimeter such that the side *BC* is incident on *L*, while the points *P* and *Q* are interior to the sides *AB* and *AC* respectively.

Finally, the linearity of the algorithm crucially depends on the fact that the following *left-interspersing lemma* holds [BM02]. Its chief implication is that we do not have to backtrack as we move from one edge to the next in an anticlockwise order.

Lemma The search for a minimum perimeter circumscribing triangle never backtracks as we traverse the polygon $\mathcal{P}$ in an anticlockwise order.

Proof The proof is provided in [BM02]. ■

The polygon needs to be traversed in a clockwise order too, as shown in the papers [BM01] and [BM02], and a similar *right-interspersing lemma* underlies the linearity of this search.

Let us take a closer look at how this linear time algorithm works step-by-step. Figures 3.6 through 3.10 illustrate some details of the algorithm that will be discussed shortly. An input polygon is defined by a set of its edges and a set of its vertices (Figures 3.4 and 3.5 respectively) that are named in a counter-clockwise order.

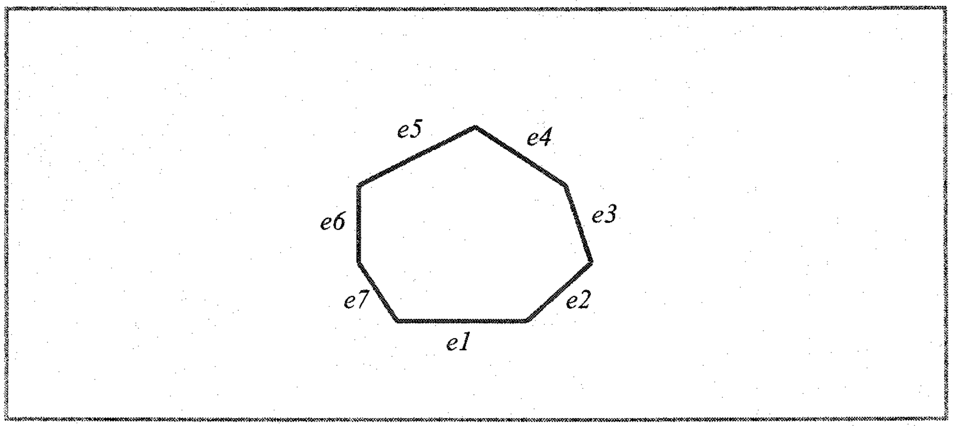


Figure 3.4 Edge notation for a convex polygon

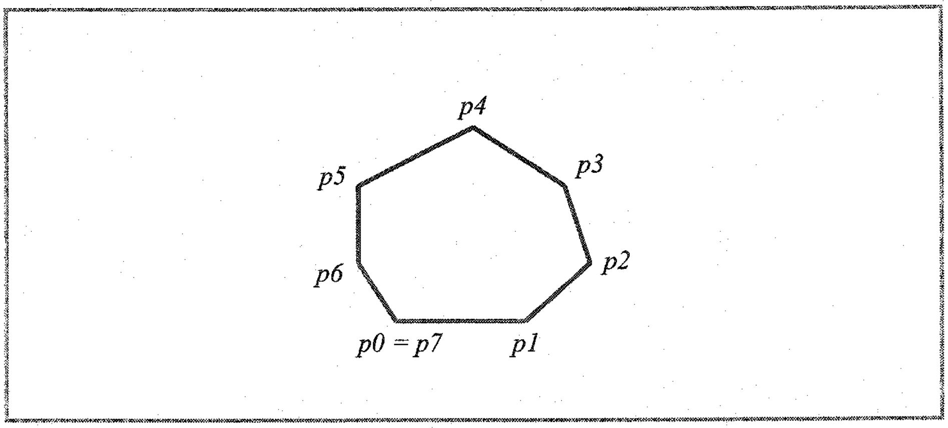


Figure 3.5 Vertex notation for a convex polygon

A list of procedures discussed below is performed for each polygon edge in turn. We refer to the overall procedure as an *anticlockwise search* of polygon edges due to the direction in which the edges are considered. There is a corresponding *clockwise search* with the only difference being the order in which the polygon edges are traversed.

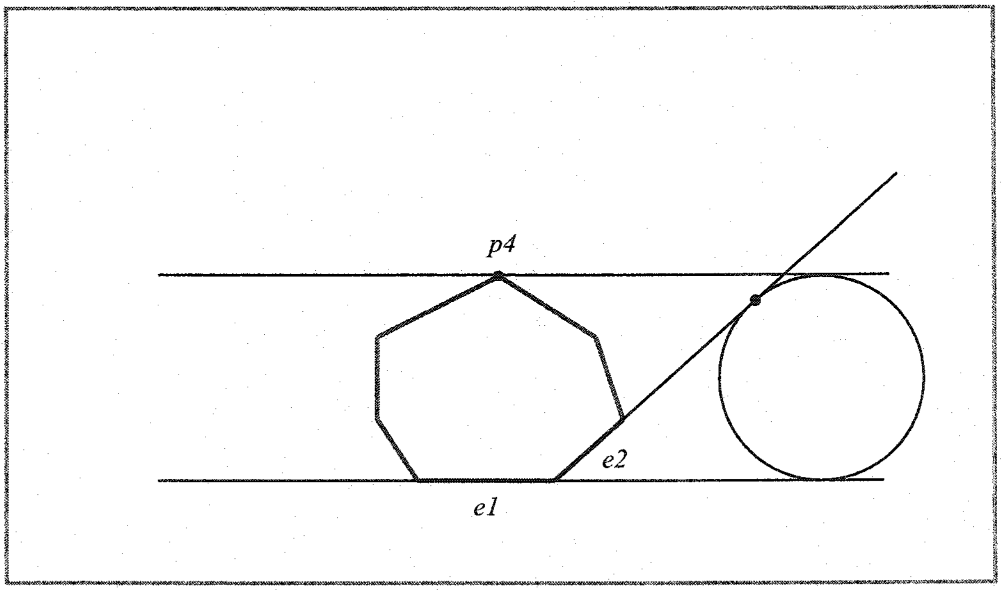


Figure 3.6 Linear time algorithm: Step 1

First, the algorithm finds a circle inscribed in a degenerate wedge, formed by an edge under consideration – e_1 to begin with – and a line antipodal to e_1, and that also touches the next anticlockwise edge, e_2 (Figure 3.6). If the point of tangency belongs to e_2 (not the case in our example), we stop. This point of tangency becomes point *Q* (see Figure 3.10), and the tangent line that contains point *Q*, e_2 here, becomes a new triangle side. Otherwise, we find a circle through polygon vertex p_2 inscribed in the same degenerate wedge as before (Figure 3.7). If the tangent to the circle at this point is also tangent to the polygon (meaning it only touches the polygon in one point), we stop and assign point *Q* and a new triangle side correspondingly. Else, we move to the next edge e_3 and repeat the procedure that was just performed for edge e_2. In this manner, we find point *Q* by a brute-force search.

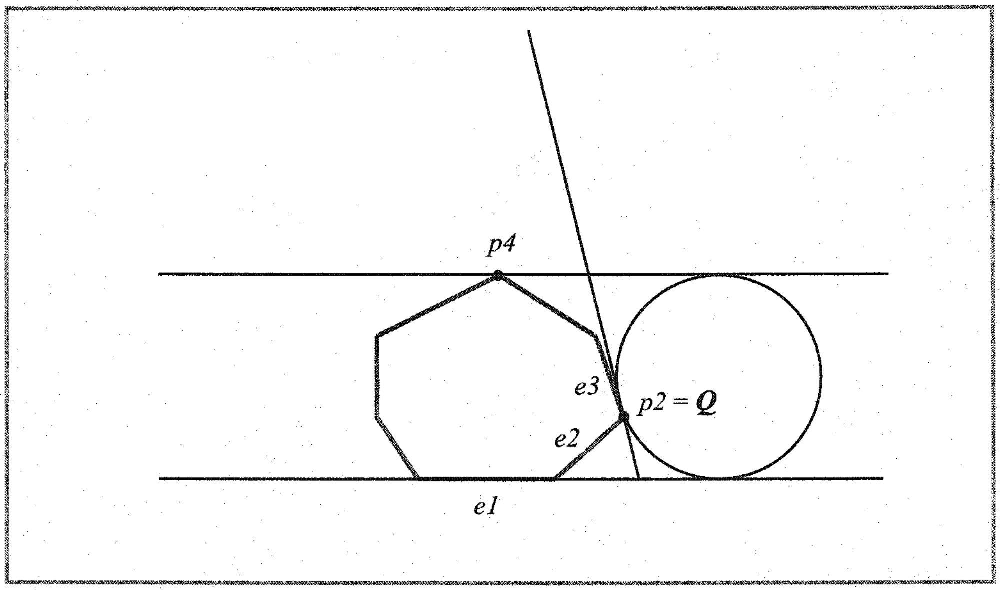


Figure 3.7 Linear time algorithm: Step 2

We define a concept (already mentioned above) of an *antipodal line* for edge e_i as a line parallel to e_i, tangent to the polygon, and that passes through a polygon vertex antipodal to the e_i or lies on a polygon edge that includes an antipodal to e_i vertex.

Another concept used in the algorithm is one of a *balanced wedge*. A wedge W is said to be balanced when we find a line through a given point Q contained in W, such that this line is tangent to the largest circle inscribed in W that passes through point Q. A wedge is balanced for a polygon when such a tangent line is also tangent to the polygon. In this case, point Q is on the boundary of the polygon, inside W. And we can see how the concept of a balanced wedge can be used to find such a point Q.

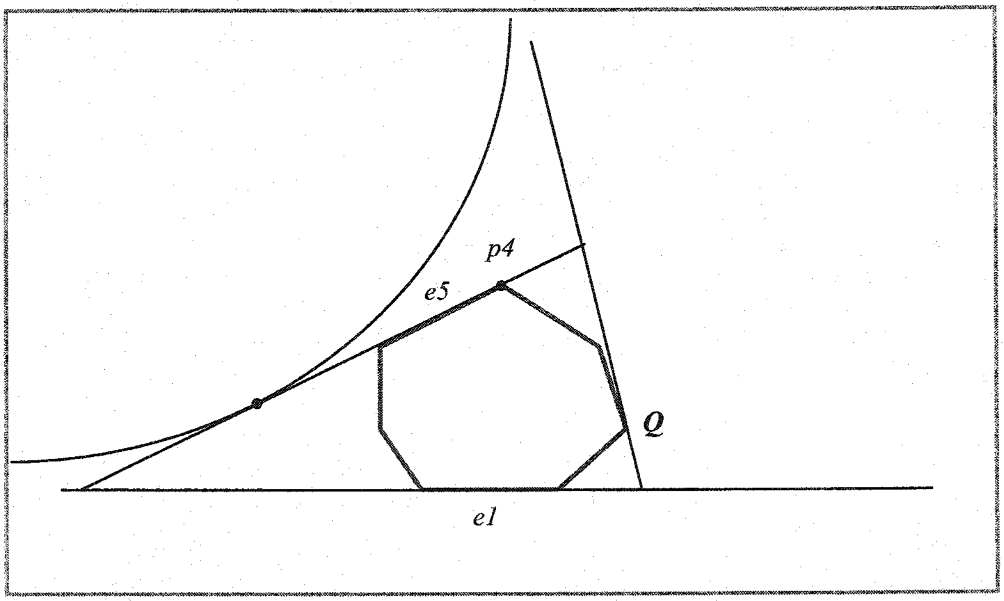


Figure 3.8 Linear time algorithm: Step 3

Once we have found a balanced wedge for the first degenerate configuration (steps 1 and 2 in Figures 3.6 and 3.7 respectively), we switch to a new wedge whose arms are along the newly found tangent line and our current edge e_1 (Figure 3.8). We find point P (see Figure 3.10) and a line tangent to P in the same brute-force manner as we found point Q (Figures 3.8 and 3.9). The only difference is that now we start from an edge that immediately follows the antipodal point or the antipodal line (e_5 here).

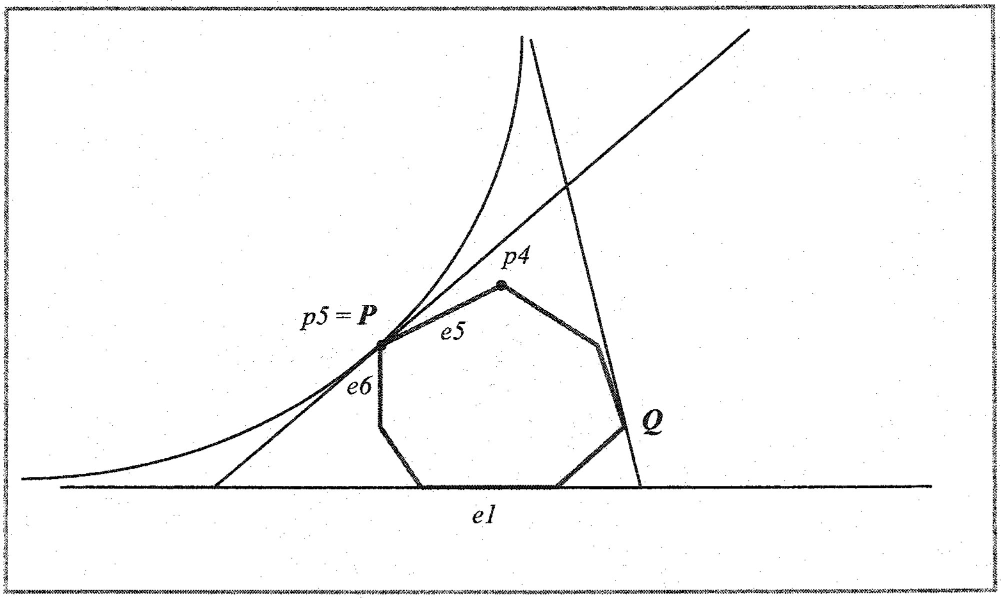


Figure 3.9 Linear time algorithm: Step 4

Once we have found points P and Q and their corresponding tangent lines that become two new sides of the enclosing triangle (Figure 3.10), we have completed the first *iteration* of the algorithm for edge e_1. We now check the following condition, necessary (but not sufficient) to attain a minimum perimeter triangle configuration for a given edge: $|BP| = |CQ|$. If this condition is satisfied, we have the minimum perimeter triangle for the current edge, and we stop. No more iterations are needed for this polygon edge. If, on the contrary, the condition is not satisfied, we repeat the procedures outlined for the first iteration again. This will be the second iteration for a given edge. The only difference here and in any subsequent iteration is that we do not have to start with a degenerate wedge, and instead use the tangent line found for point P in the preceding iteration. This evidently allows us to progress to a new enclosing triangle configuration, which, as it was shown in [BM01], has a reduced triangle perimeter. Finally, we remark that if the height of point P becomes less than the height of point Q at any time, we stop and move on to the next polygon edge, as such a configuration would not generate a minimum perimeter triangle according to [BM01]. It is then necessary to traverse the edges in a clockwise order to check if such an edge has a minimum perimeter triangle corresponding to it.

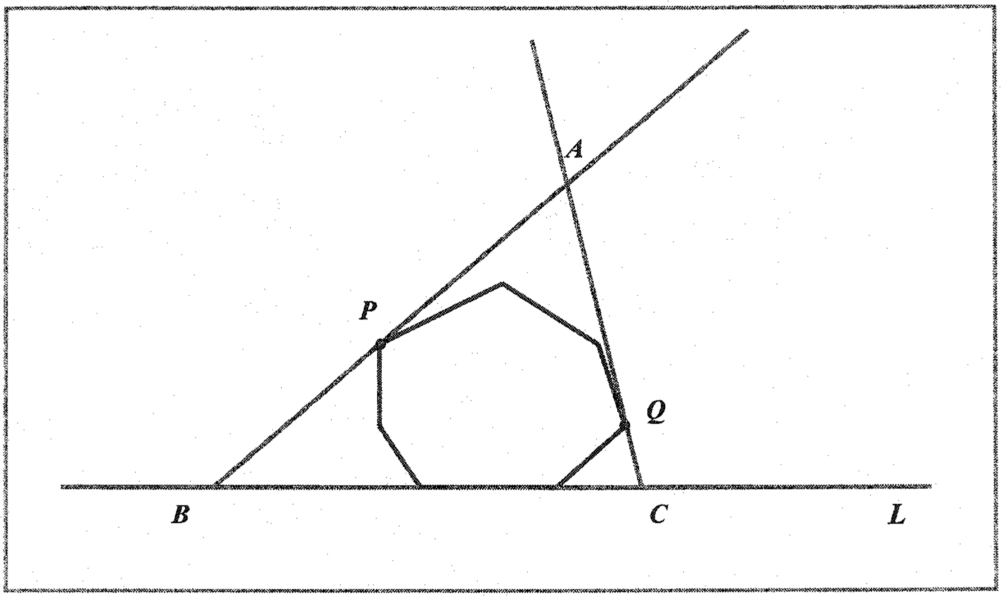


Figure 3.10 Linear time algorithm: The result of the first iteration for edge e_1

Note that determining whether a line is tangent to a polygon can be done by considering the n products of the form $(y_i - m \cdot x_i - b) \cdot (y' - m \cdot x' - b)$, where (x_i, y_i) is a polygon vertex ($0 \le i < n$), (x', y') is the center of the circle that defines the tangent line, and $y = m \cdot x + b$ is the equation of the tangent line. If all such products are negative or all are positive (some may be zero as those for the vertices on the tangent line), then, the line is tangent to the polygon.

Below is the algorithm in pseudocode that computes the minimum perimeter triangle circumscribing a convex polygon in linear time [BM01], [BM02]. Note that procedure MinPerimeter has to be executed for every polygon edge. The minimum perimeter triangle for the polygon is the minimum among the triangles computed for all polygon edges (for some edges, such triangles may not exist – see Algorithm).

```
Algorithm   MinPerimeter ( $e_i$, $\mathcal{P}$ )
Input       A convex polygon $\mathcal{P}$ and an edge $e_i$ of this polygon
Output      A triangle $\Delta\, ABC$, with $e_i$ lying on side $BC$
Choose a line antipodal to $L$ (to $BC$)  /* This gives a degenerate left wedge that contains $\mathcal{P}$ */
Set $u \leftarrow \infty$, $h_P \leftarrow \infty$, $h_Q \leftarrow 0$
1     ( $AC$, $Q$ ) ← FindBalanced ( $W(L, AB)$, $\mathcal{P}$, $h_P$ )
      $v \leftarrow length(CQ)$,    $h_Q \leftarrow height(Q)$
      if ( $h_Q > h_P$ ) STOP /* There is no $\Delta\, ABC$ for edge $e_i$ */
      if ( $u = v$ ) go to 2

      ( $AB$, $P$ ) ← FindBalanced ( $W(L, AC)$, $\mathcal{P}$, $h_Q$ )
      $u \leftarrow length(BP)$,    $h_P \leftarrow height(P)$
      if ( $h_Q > h_P$ ) STOP /* There is no $\Delta\, ABC$ for edge $e_i$ */
      if ( $u \neq v$ ) go to 1
2     return triangle $\Delta\, ABC$
```

3.2 Providing Proofs for the Minimum Perimeter Triangle Configuration Conditions

Theorem The minimum perimeter triangle configuration is achieved when the following two conditions are satisfied (see Figure 3.11):

1. $BP = CQ$,
2. $AP + AQ = BC$.

These conditions are not independent; that is, one can be inferred from the other. They were originally mentioned in [BM01] and then presented in [BM02]; however, there were no detailed proofs provided to justify these claims. Due to their importance for the algorithm, we provide the corresponding proofs here.

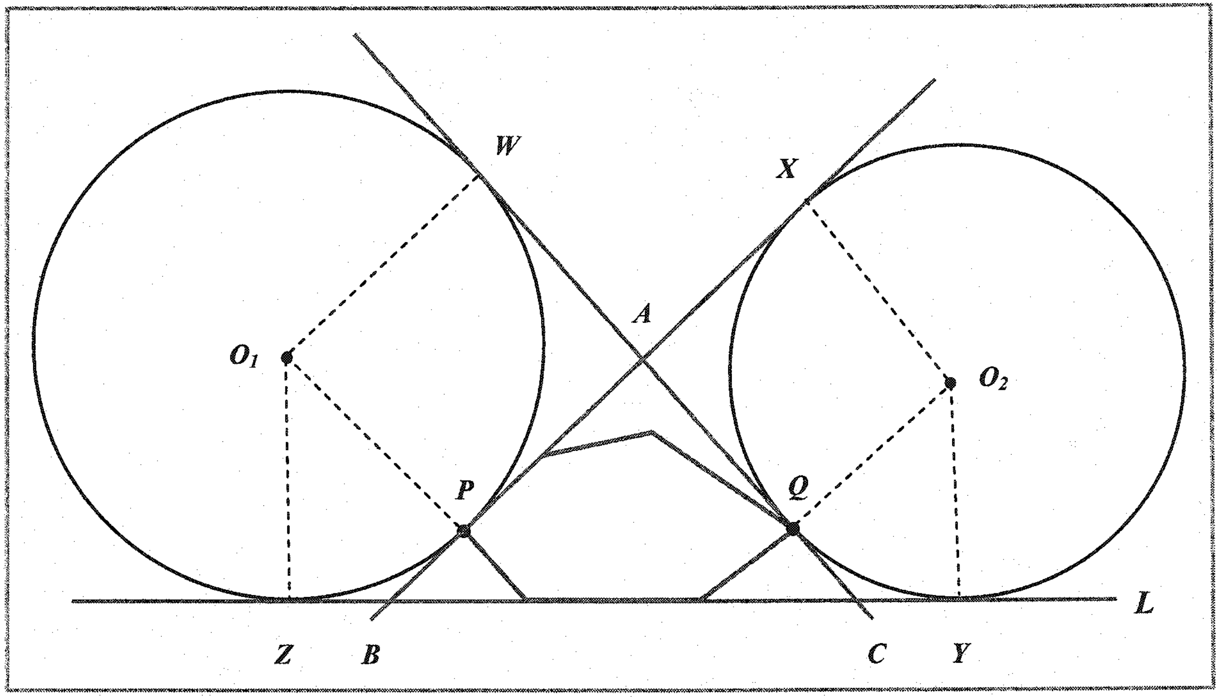


Figure 3.11 The minimum perimeter configuration conditions: Geometry

Proof Consider the minimum perimeter configuration of Figure 3.11. It was shown in [BM01] that points P and Q lie on the polygon. We now show that the algebraic conditions specified in the Theorem are correct.

1. The six equalities below follow from geometry of the minimum perimeter triangle configuration (see Figure 3.11):

   $BP = BZ,\qquad CQ = CY,$

   $AP = AW,\qquad AQ = AX,$

$CZ = CW, \qquad BY = BX.$

The perimeter of the enclosing triangle can be expressed as follows:

$$\begin{aligned} \textit{Perimeter} \quad &= BC + BP + AP + AC \\ &= \underbrace{BC + BZ}_{CZ} + AP + AC \\ &= CZ + \underbrace{AP + AC}_{CZ} = 2 \cdot CZ \end{aligned}$$

$AP + AC = CZ$ in the last expression since $CZ = CW = AC + AW = AC + AP$.

On the other hand, the perimeter can also be expressed as follows:

$$\begin{aligned} \textit{Perimeter} \quad &= BC + CQ + AQ + AB \\ &= \underbrace{BC + CY}_{BY} + AQ + AB \\ &= BY + \underbrace{AQ + AB}_{BY} = 2 \cdot BY \end{aligned}$$

$AQ + AB = BY$ in the last expression since $BY = BX = AB + AX = AB + AQ$.

Thus we have:

$$2\underbrace{CZ}_{BC+BZ} = 2\underbrace{BY}_{BC+CY} \Rightarrow BZ = CY \Rightarrow BP = CQ\,. \;\square$$

2. $BY = BX \Rightarrow$

$BC + CY = AB + AX \Rightarrow$

$BC + CQ = AB + AQ$

Subtracting BP from both sides of the last expression we obtain:

$BC + CQ - BP = AB - BP + AQ$

And since $CQ - BP = 0$ (due to property 1 of the Theorem) and $AB - BP = AP$,

we finally have: $BC = AP + AQ$. ■

3.3 GEOMETRIC PRELIMINARIES FOR THE LINEAR TIME ALGORITHM

The algorithm uses several geometric operations as it computes the enclosing triangles for the polygon. In particular, it relies on the operations such as finding a point and a line antipodal to a polygon edge, finding an excircle for a triangle, computing a point of tangency of a circle and a line, finding a circle inscribed in a wedge, computing a cross point of two lines, and so on. To achieve maximum efficiency for our implementation of the algorithm, we mathematically derived optimal solutions for all operations specified in the algorithm. All geometric primitives were determined through the use of Euclidian orthogonal coordinates.

The following discussion provides the expressions for some of these primitives. Most expressions below were derived using line coefficients (from line equations). In cases when lines are vertical, some trivial modifications to the derived expressions are needed. They are omitted here due to their simplicity and space limitations.

1. Finding a point antipodal to an edge.

Proposition The farthest pair of a set of N points in the plane constitutes an antipodal pair of points.

Proof Let S be a set of N points in the plane. Let also points A and B constitute the farthest pair of these N points. To prove that points A and B also constitute an antipodal pair of points, we need to show that a set of N points in the plane admits parallel lines of support (by definition of antipodal points [PS85]). On Figure 3.12, such lines of support are labeled L_1 and L_2. Since B is the farthest point from A, it follows that all points from set S must lie in the disk determined by the circle centered at point A with radius equal to *distance* (*A*, *B*). It then follows that we can construct a line of support L_1 which is tangent to the circle at point *B*. We can construct a line of support L_2 in a similar way. Since L_1 and L_2 are each orthogonal to line segment [*A*, *B*], they must be parallel, thus proving the Proposition. ■

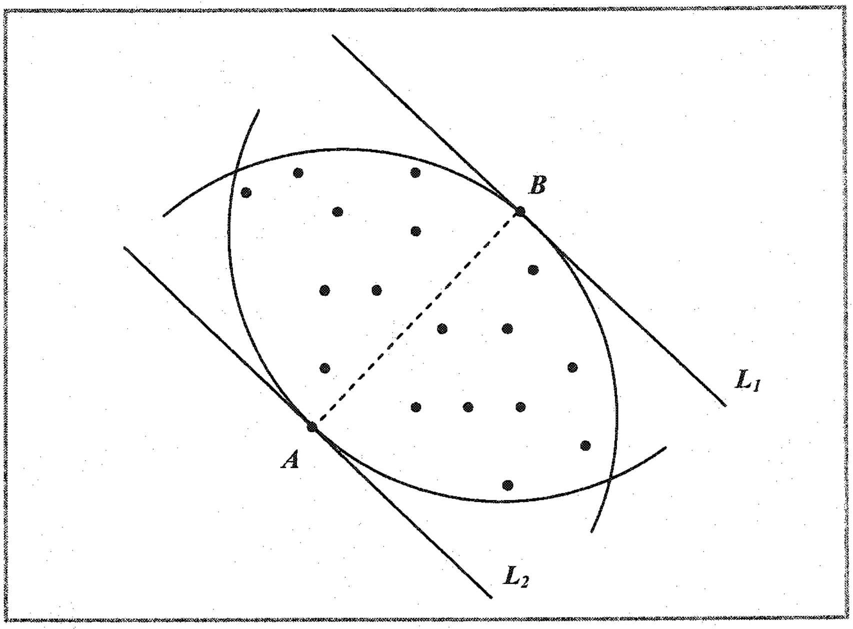


Figure 3.12 Finding an antipodal pair of points

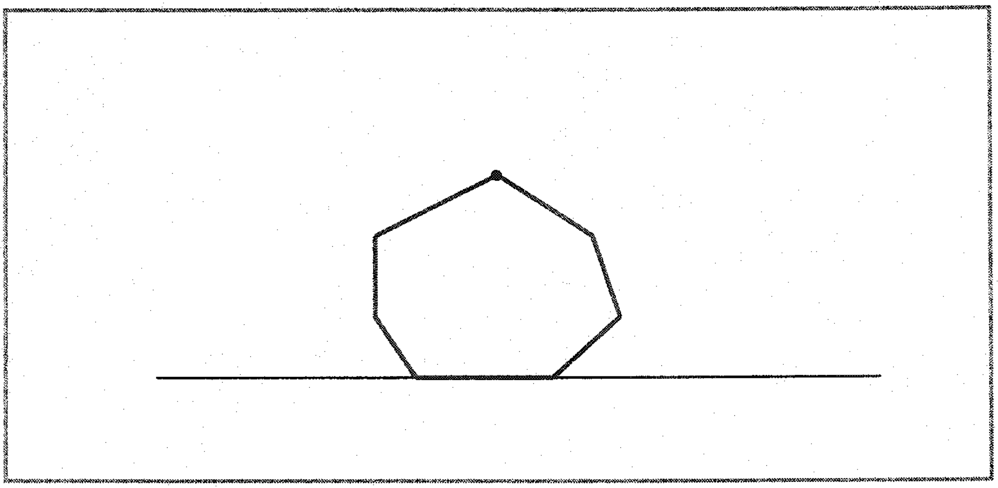


Figure 3.13 Finding a point antipodal to an edge

Based on the Proposition, a point antipodal to a given polygon edge is found as the farthest from that edge polygon vertex (Figure 3.13).

2. Finding a line antipodal to an edge.

A line that is antipodal to a given polygon edge is parallel to that edge and passes through an antipodal to that edge point (Figure 3.14). The antipodal line may be flush with another polygon edge.

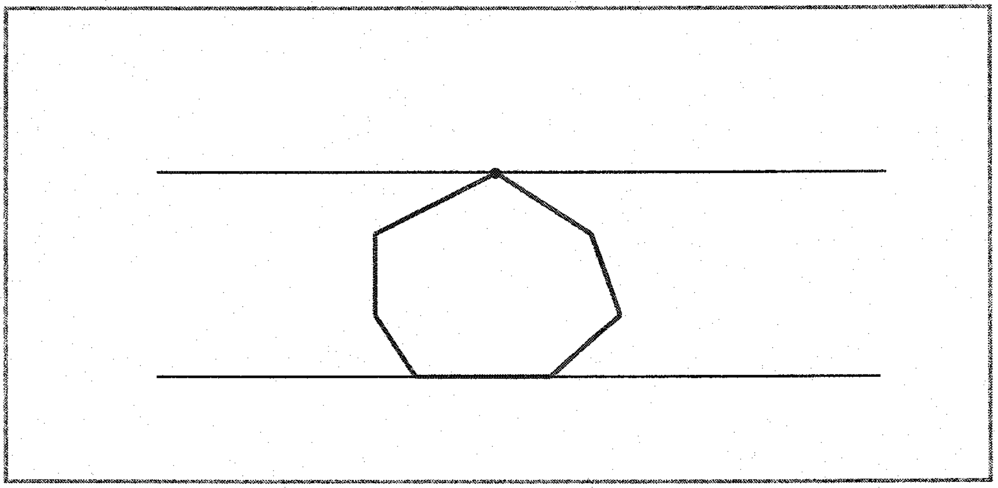


Figure 3.14 Finding a line antipodal to an edge

3. Finding a circle inscribed in a degenerate wedge and that touches a line.

There are always two circles that can be inscribed between two parallel lines and that touch a third line, non-parallel to the first two lines (Figure 3.15).

If the two parallel lines, L_1 and L_2, are given by their corresponding equations $y = m_1 x + b_1$ and $y = m_2 x + b_2$ (note that for parallel lines, $m_1 = m_2$), and the third line's equation is $y = m_3 x + b_3$ (line L_3), then, the coordinates of the centers of the two circles and the circles' radiuses (which are the same in this case) are found using the following logic. (Note that choosing one of the two circle solutions is trivial as the excircle of choice and the polygon lie on the opposite sides of line L_3.)

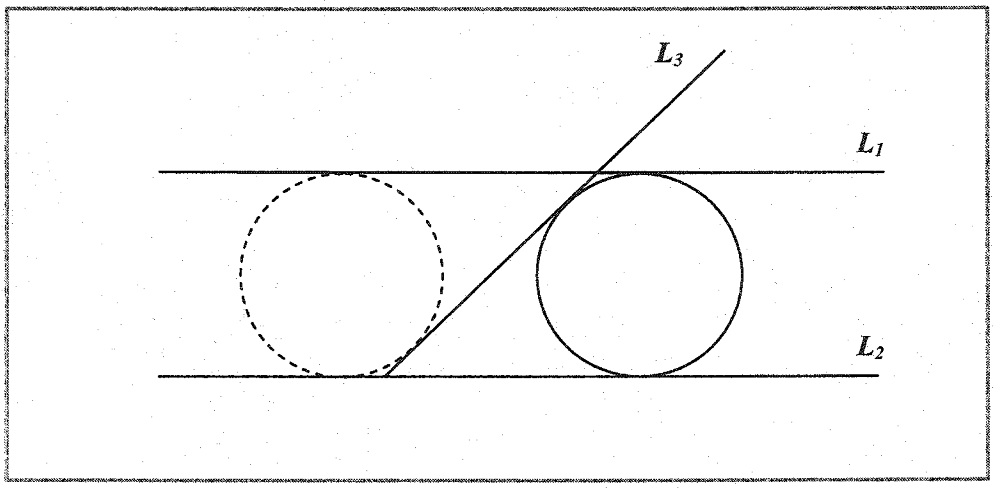


Figure 3.15 Finding a circle inscribed in a degenerate wedge and that touches a line

First, we notice that the radius of the resulting circle is half the distance between the two parallel lines. Second, finding the center of the circle can be done after analyzing the geometric configuration illustrated in Figure 3.16. We can see from Figure 3.16 that our resulting circle is inscribed in a rhomb formed by the two given parallel lines and the third line and a line parallel to the third line, line L_4 (from simple geometry, it is only possible to inscribe a circle in a parallelogram if that parallelogram is a rhomb).

The main idea of our approach is that the radius of the circle whose diameter is defined by a line segment of line L_3 confined between lines L_1 and L_2 (the "bigger" circle in Figure 3.16) is equal to the length of the median of the right triangle whose right angle is fixed in the center of the "smaller" circle and the base is flush with the diameter of the "bigger" circle. In other words, $\angle MO_2N = 90°$ and the radius of the circle centered in point O_1 is equal to the median of ΔMO_2N that corresponds to its rights angle. Finding the length of the median is trivial: it equals to $\frac{1}{2}$ of the length of line segment [M, N]. It then only remains to shift the center of the "bigger" circle by the length of the median to the right, so to arrive at the center of the "smaller" circle that was our objective (O_1 to O_2 shift). Note that in a general case, "smaller" and "bigger" terms are not relevant and were only used in our discussion to provide clarity in referral to Figure 3.16.

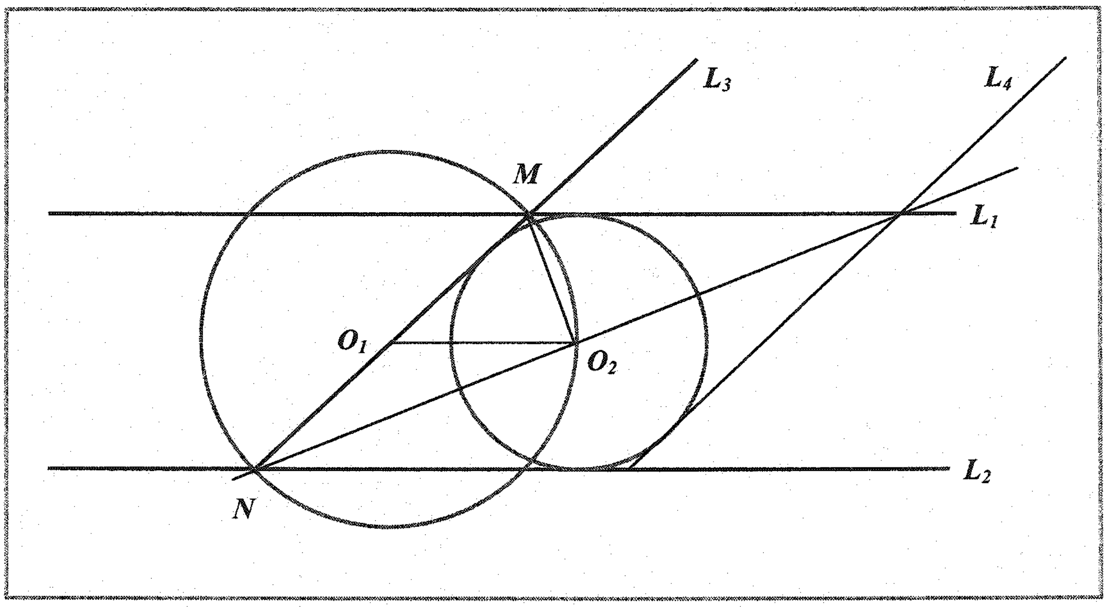


Figure 3.16 Geometric details for finding a circle in Figure 3.15

Due to space limitations and due to their simplicity, we omit the expressions for the final solution.

4. Finding a circle inscribed in a degenerate wedge and that passes through a point.

The following idea provides a solution to this geometric primitive. Let the coordinates of a point inside a degenerate wedge be (x', y') and let us assume that this point is contained in the wedge (Figure 3.17). There are, as in the previous case, always two solutions. For both of them, the radius is found trivially as $\frac{1}{2}$ of the distance between the two parallel lines. Choosing one of the two solutions is not difficult and depends of the geometric configuration of the polygon. We always choose a circle that is exterior to the polygon.

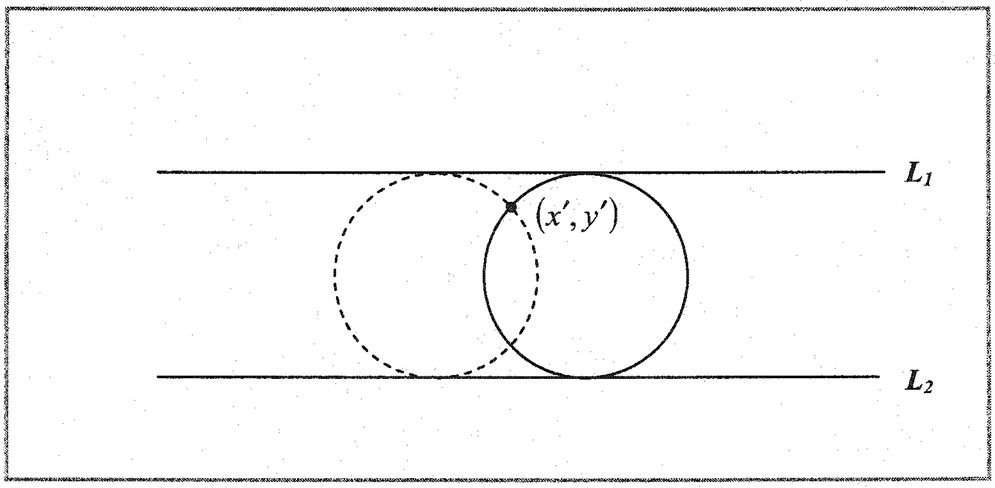


Figure 3.17 Finding a circle inscribed in a degenerate wedge and that passes through a point

If the two parallel lines, L_1 and L_2, are given by their corresponding equations $y = m_1 x + b_1$ and $y = m_2 x + b_2$ (for parallel lines, $m_1 = m_2$), and the coordinates of a given point are (x', y'), then, the coordinates of the centers of two circles (x_1, y_1) and (x_2, y_2) can be found by solving the following system of equations (we omit the solution here due to its simplicity):

$$\begin{cases} y = m_1 x + \dfrac{b_1 + b_2}{2} \\ (x' - x)^2 + (y' - y)^2 = (d/2)^2 \end{cases}$$, where d is the distance between the given parallel lines.

The first equation of the system implies that the center of the inscribed circle(s) lies on the line that is parallel to the given parallel lines and is "half-way" between them. The equation of such a line is the first equation in the system. The second equation of the system implies that the given point (x', y') belongs to the circle(s). Solving this system for x and y, we obtain the coordinates of the two circles inscribed between the two parallel lines and that pass through (x', y').

5. Finding a circle inscribed in a wedge and that touches a line.

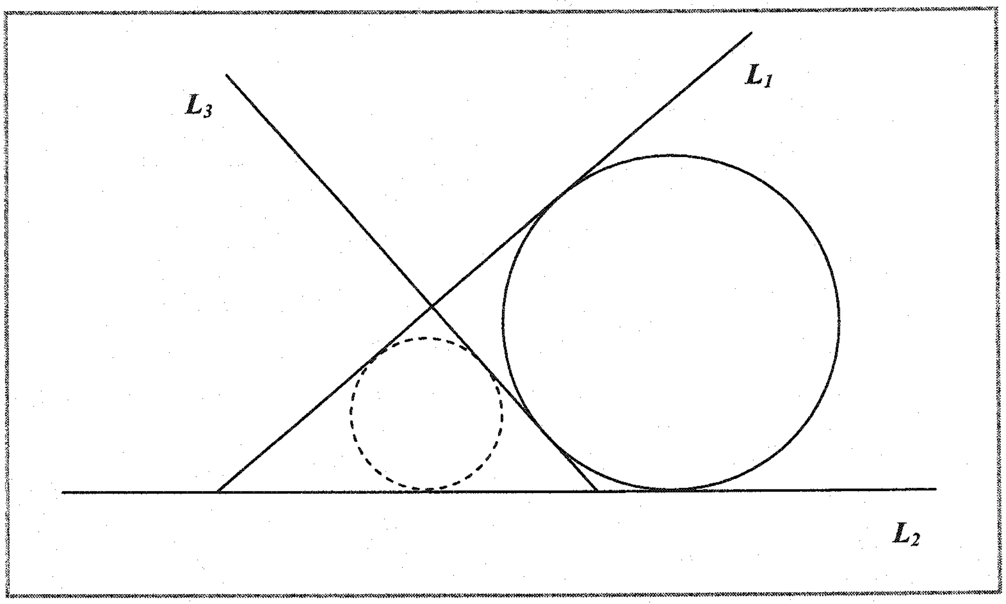


Figure 3.18 Finding a circle inscribed in a wedge and that touches a line

Here, we have three non-parallel lines $y = m_1x + b_1$, $y = m_2x + b_2$, and $y = m_3x + b_3$. We want to find a circle inscribed in a wedge formed by the first two lines, L_1 and L_2, and that touches the third line, L_3 (is an excircle to it). There are two mathematical solutions to this condition (Figure 3.18), which have different radiuses. The solution of choice (the excircle) simply has a bigger radius.

Generally speaking, there are a total of four possible solutions to this condition, two of which are not shown in Figure 3.18 and are inscribed in the other wedge formed by lines L_2 and L_1. This wedge is obtuse-angled in Figure 3.18. We thus find all four of these solutions algebraically and then choose the two that have their circle centers on the bisector of the given wedge.

$$M_1' = \sqrt{m_1^2+1} + \sqrt{m_2^2+1} \qquad M_1'' = \sqrt{m_1^2+1} + \sqrt{m_3^2+1}$$

$$M_2' = \sqrt{m_1^2+1} + \sqrt{m_2^2+1} \qquad M_2'' = -\sqrt{m_1^2+1} + \sqrt{m_3^2+1}$$

$$M_3' = -\sqrt{m_1^2+1} + \sqrt{m_2^2+1} \qquad M_3'' = \sqrt{m_1^2+1} + \sqrt{m_3^2+1}$$

$$M_4' = -\sqrt{m_1^2+1} + \sqrt{m_2^2+1} \qquad M_4'' = -\sqrt{m_1^2+1} + \sqrt{m_3^2+1}$$

$$B_1' = b_1\sqrt{m_2^2+1} + b_2\sqrt{m_1^2+1} \qquad B_1'' = b_1\sqrt{m_3^2+1} + b_3\sqrt{m_1^2+1}$$

$$B_2' = b_1\sqrt{m_2^2+1} + b_2\sqrt{m_1^2+1} \qquad B_2'' = b_1\sqrt{m_3^2+1} - b_3\sqrt{m_1^2+1}$$

$$B_3' = b_1\sqrt{m_2^2+1} - b_2\sqrt{m_1^2+1} \qquad B_3'' = b_1\sqrt{m_3^2+1} + b_3\sqrt{m_1^2+1}$$

$$B_4' = b_1\sqrt{m_2^2+1} - b_2\sqrt{m_1^2+1} \qquad B_4'' = b_1\sqrt{m_3^2+1} - b_3\sqrt{m_1^2+1}$$

$$K_1' = m_1\sqrt{m_2^2+1} + m_2\sqrt{m_1^2+1} \qquad K_1'' = m_1\sqrt{m_3^2+1} + m_3\sqrt{m_1^2+1}$$

$$K_2' = m_1\sqrt{m_2^2+1} + m_2\sqrt{m_1^2+1} \qquad K_2'' = m_1\sqrt{m_3^2+1} - m_3\sqrt{m_1^2+1}$$

$$K_3' = m_1\sqrt{m_2^2+1} - m_2\sqrt{m_1^2+1} \qquad K_3'' = m_1\sqrt{m_3^2+1} + m_3\sqrt{m_1^2+1}$$

$$K_4' = m_1\sqrt{m_2^2+1} - m_2\sqrt{m_1^2+1} \qquad K_4'' = m_1\sqrt{m_3^2+1} - m_3\sqrt{m_1^2+1}$$

The four possible circle centers are now given by the following expressions. Note that finding the circles' radiuses is now trivial as they are simply the distances from the circle centers to the corresponding arms of the given wedge.

$X = (B'' \cdot M' - B' \cdot M'')/(K' \cdot M'' - K'' \cdot M')$ (for all four cases),

$Y = (B' \cdot K'' - B'' \cdot K')/(K'' \cdot M' - K' \cdot M'')$ (for all four cases).

6. Finding a circle inscribed in a wedge and that passes through a point.

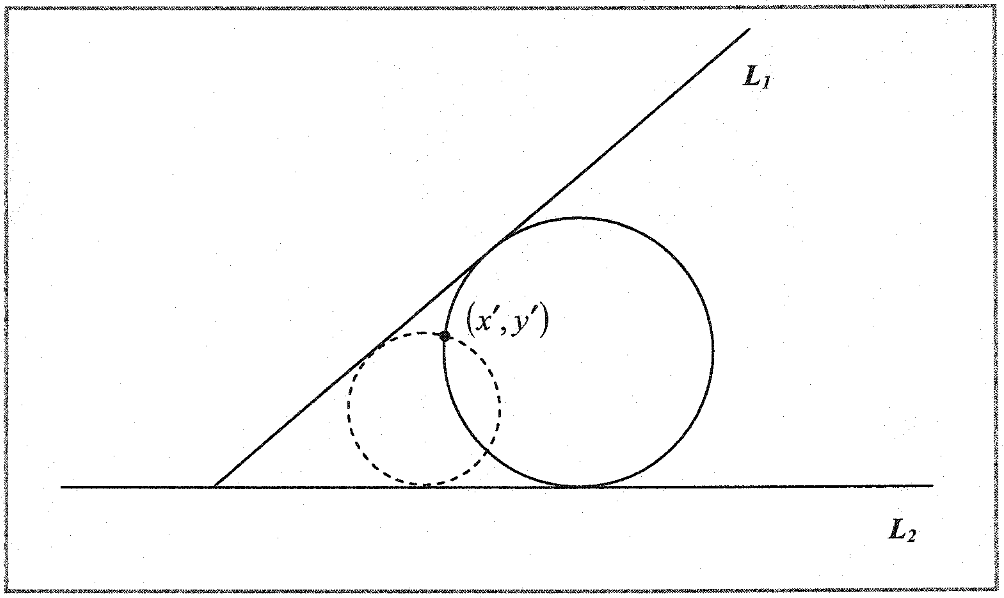


Figure 3.19 Finding a circle inscribed in a wedge and that passes through a point

Finally, we find a solution to the condition depicted in Figure 3.19, where the point inside the wedge has the coordinates (x', y'). Note that this solution will be of a fundamental importance to our algebraic expression derived for the main subsidiary problem in Chapter 4.

$$A = (m_1 \cdot m' + 1)^2$$

$$B = -2(m_1^2 + 1) \cdot (x' + y' \cdot m') + 2m' \cdot (b' \cdot m_1^2 + b_1) + 2m_1(b' - b_1)$$

$$C = (m_1^2 + 1) \cdot ((x')^2 + (y')^2 - 2y' \cdot b') + b' \cdot (2b_1 + m_1^2 \cdot b') - b_1^2$$

$$D = B^2 - 4 \cdot A \cdot C$$

In the equations above, *m'* and *b'* are the coefficients of a line equation that defines a bisector of the given wedge. There are always two distinct bisectors for a given non-degenerate wedge, which cross at a 90° angle. Choosing the correct bisector is done with respect to the location of point (x', y') that is contained in the wedge. Mathematically, the two bisectors are found as follows:

$$a_1 = m_2\sqrt{m_1^2+1} - m_1\sqrt{m_2^2+1} \qquad a_2 = -m_2\sqrt{m_1^2+1} - m_1\sqrt{m_2^2+1}$$

$$b_1 = \sqrt{m_2^2+1} - \sqrt{m_1^2+1} \qquad b_2 = \sqrt{m_2^2+1} + \sqrt{m_1^2+1}$$

$$c_1 = b_2\sqrt{m_1^2+1} - b_1\sqrt{m_2^2+1} \qquad c_2 = -b_2\sqrt{m_1^2+1} - b_1\sqrt{m_2^2+1}$$

$$m_1' = -\frac{a_1}{b_1} \qquad b_1' = -\frac{c_1}{b_1} \qquad m_2' = -\frac{a_2}{b_2} \qquad b_2' = -\frac{c_2}{b_2}$$

Finally, the coordinates of the centers of the inscribed circles and the circles' radiuses are determined as follows ("left" and "right" circles refer to their centers' relative *x* coordinates).

Left circle: Right circle:

$$x_1 = \left(-B - \sqrt{D}\right)/2A \qquad x_2 = \left(-B + \sqrt{D}\right)/2A$$

$$y_1 = m' \cdot x_1 + b' \qquad y_2 = m' \cdot x_2 + b'$$

$$R_1 = \sqrt{(x_1 - x')^2 + (y_1 - y')^2} \qquad R_2 = \sqrt{(x_2 - x')^2 + (y_2 - y')^2}$$

Choosing between the two circles is done based on the circles' radiuses. Clearly, we choose a circle with a bigger radius.

Chapter 4 SOLVING THE MINIMUM PERIMETER TRIANGLE PROBLEM ALGEBRAICALLY

In this chapter, we discuss our findings regarding the expression of the circumscribing triangle's perimeter via the geometric configuration of its enclosed object for the main subsidiary problem (Problem 2). In this problem, a degenerate object if formed by a pair of points located at some heights above a given line. Such an algebraic expression and its solution help establish an existence of a linear time answer to the minimum perimeter triangle problem for a convex polygon.

We begin this chapter with reiterating the statements of the two important problems from Chapter 3.

4.1 A SIMPLE SUBSIDIARY PROBLEM

Problem 1 Given a wedge *W(BA, BC)* and a point *Q* contained in it, find a triangle of minimum perimeter that has two of its sides along the arms of the wedge and the third side incident on the given point *Q* (Figure 4.1; Problem 1 is repeated from Chapter 3).

A solution to this simple problem is obtained by fitting a circle into the wedge *W* that passes through point *Q*. A line tangent to this circle at point *Q* forms the third side of the minimum perimeter triangle for Problem 1. The first two triangle sides are flush with the arms of the given wedge. We note that for a given wedge it is always possible to inscribe two circles that pass through a given point *Q* (see section 3.3). We call one of these circles an *excircle* (a circle with a bigger radius) and the other an *incircle* [BM02] (Figure 4.1). For the purposes of this problem, we always select the excircle.

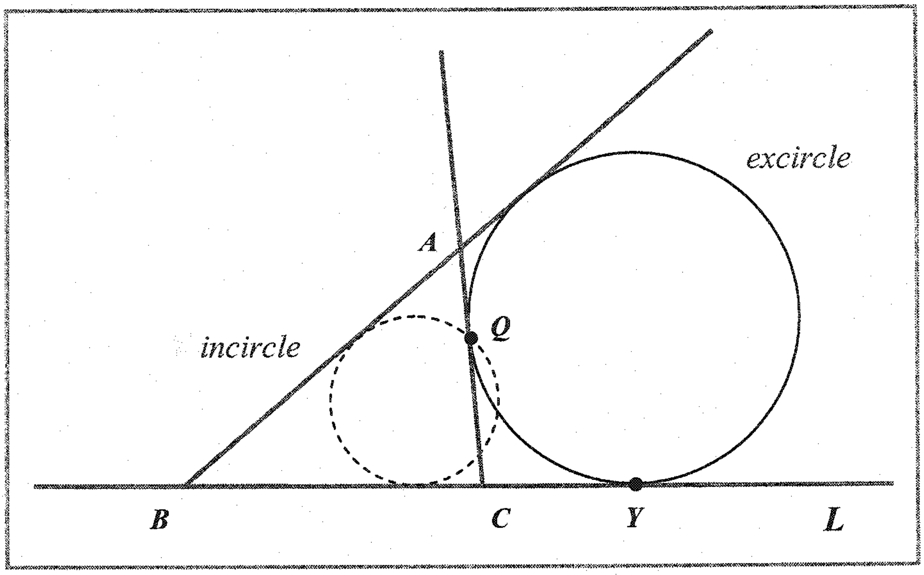


Figure 4.1 The first subsidiary problem

The reason why $\Delta\ ABC$ in the solution above is the minimum perimeter triangle for Problem 1 is due to the following [BM01]. The perimeter of the minimum perimeter triangle is equal to twice the length of a tangent from the apex of the wedge to the excircle of the triangle: $2 \cdot |BY|$ in Figure 4.1. (An excircle of a triangle is fitted into a wedge formed by the extensions of two of the triangle sides, while touching the third side from the outside of the triangle). The perimeter is clearly least when the excircle touches *AC* at *Q*.

4.2 THE MAIN SUBSIDIARY PROBLEM

Problem 2 Given two points *P* and *Q* at heights h_P and h_Q respectively, $h_P \geq h_Q$, above a line *L*, *P* to the left of *Q*, find a $\Delta\ ABC$ of minimum perimeter such that the side *BC* is incident on *L*, while the points *P* and *Q* are interior to the sides *AB* and *AC* respectively (Figure 4.2; Problem 2 is repeated from Chapter 3).

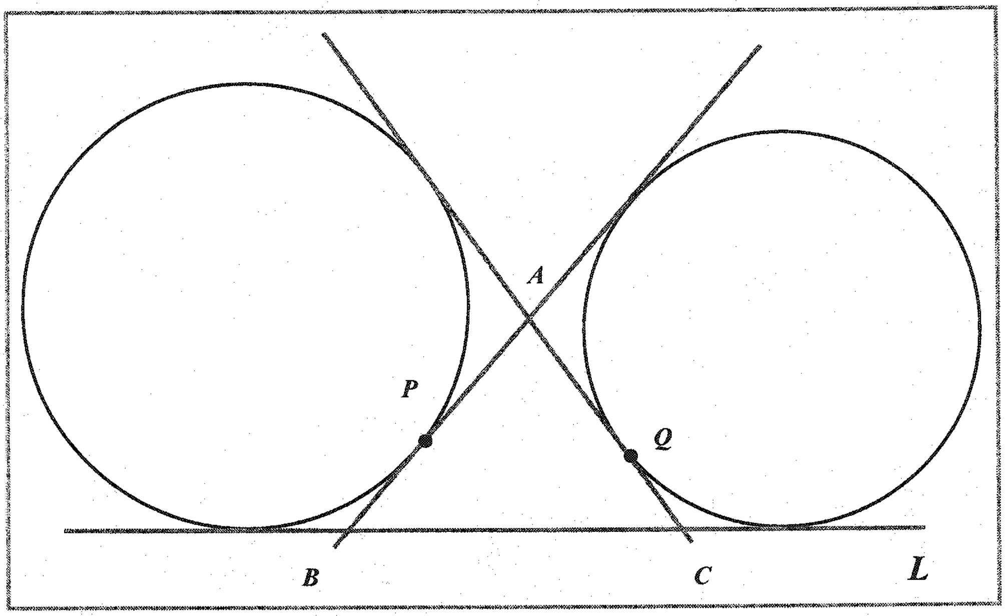


Figure 4.2 The second (main) subsidiary problem

```
Algorithm   MinPerimeter ( $P$, $Q$ )
Input       Two points $P$ and $Q$ lying above a line $L$ at heights $h_P \geq h_Q$
Output      A triangle $\Delta ABC$ of minimal perimeter with $P$ on side $AB$ and interior to
            it, and $Q$ on side $AC$ and interior to it

Choose a line through point $P$ parallel to $L$  /* This gives a degenerate left wedge that
contains point $Q$ */
Set $u \leftarrow \infty$
1    $AC \leftarrow$ FindBalanced ( $W(L, AB)$, $Q$ )
     $v \leftarrow$ length ( $CQ$ )
     if ( $u = v$ ) go to 2
     $AB \leftarrow$ FindBalanced ( $W(L, AC)$, $P$ )
     $u \leftarrow$ length ( $BP$ )
     if ( $u \neq v$ ) go to 1
2    return triangle $\Delta ABC$
```

The problem of finding the minimum perimeter triangle for a pair of points above a line (Problem 2) can be construed as a degenerate case of the problem where the minimum perimeter enclosing triangle is found for the polygon (and corresponds to one polygon edge, to be precise) [BM01]. In fact, the same scheme of circle-fitting and wedge-flipping used to derive the linear time algorithm for the polygon, can be used to solve Problem 2 (see pseudocode above, [BM01]).

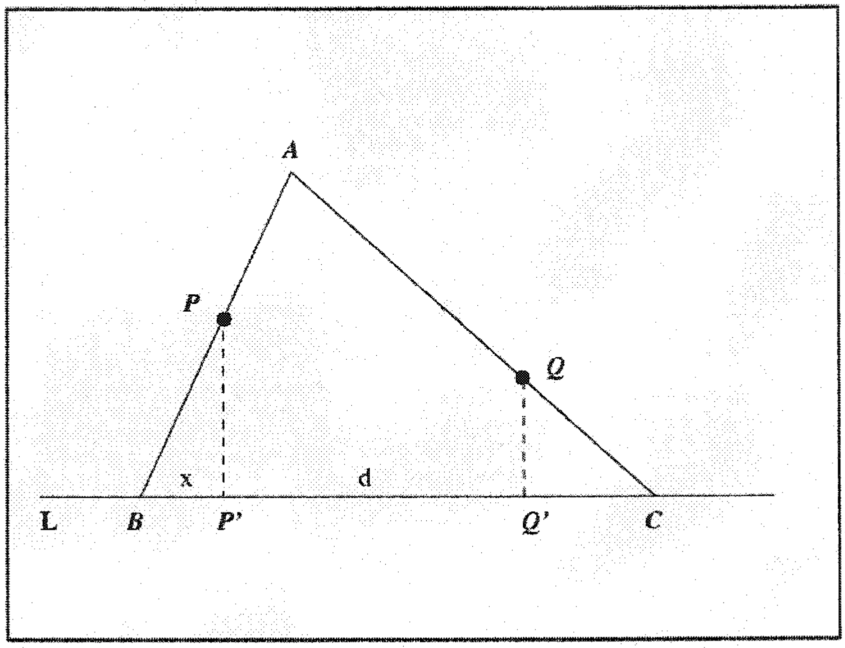


Figure 4.3 A triangle for points *P*, *Q*, and line *L*: Acute angle configuration

Let us consider solving Problem 2 algebraically. To do this, we aim at deriving an algebraic expression for the perimeter of the enclosing triangle and minimizing it with respect to the perimeter over one or more geometric parameters.

For the purposes of this problem, let us denote the height of point *P* over line *L* as *H* and the height of point *Q* over line *L* as *h*. Without loss of generality, we can reduce the number of parameters in the problem by setting *H* to 1, which is equivalent to scaling the remaining parameters by a factor 1/*H*. Thus, the scaled height of *P* becomes *H*/*H* = 1

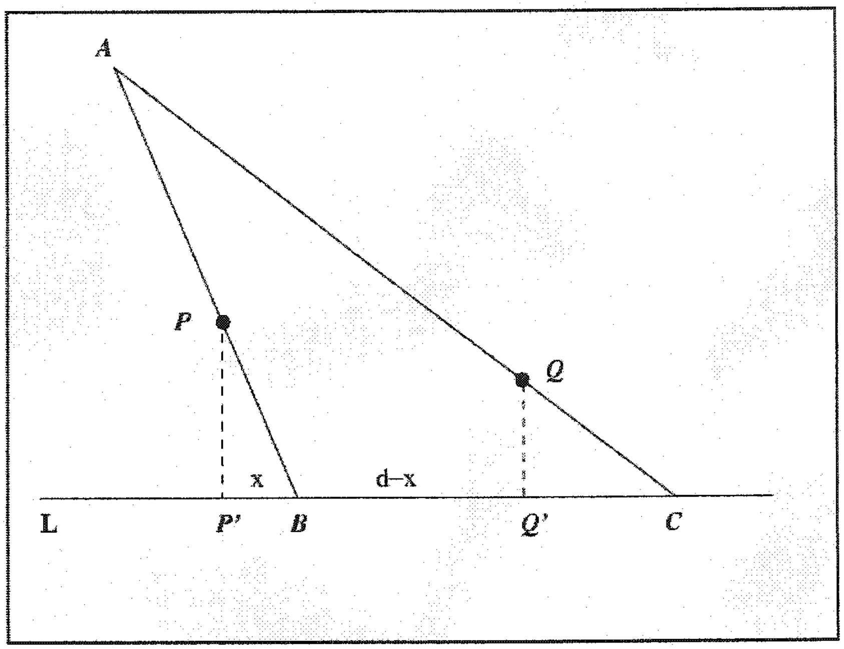


Figure 4.4 A triangle for points *P*, *Q*, and line *L*: Obtuse angle configuration 1

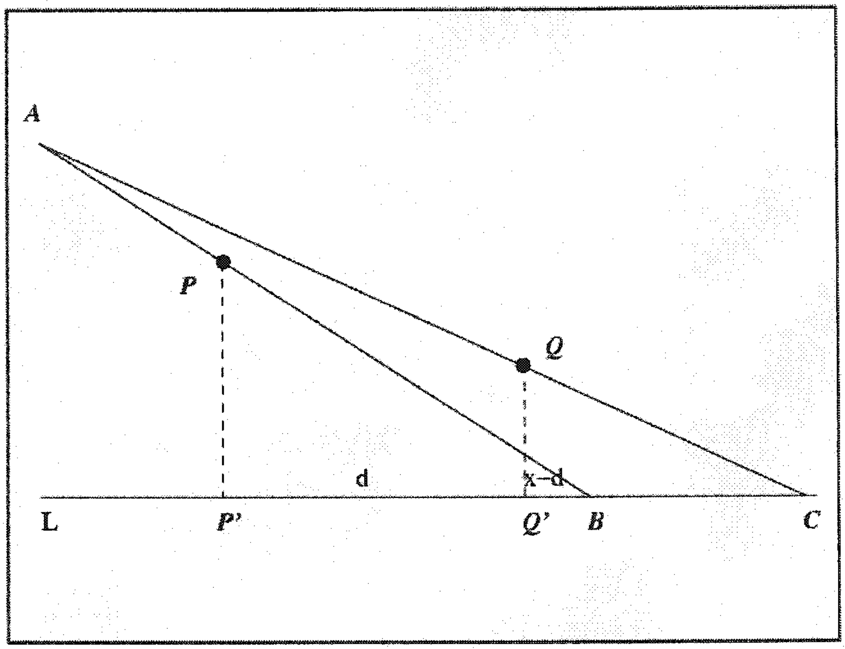


Figure 4.5 A triangle for points *P*, *Q*, and line *L*: Obtuse angle configuration 2

and the scaled height of *Q* becomes *h/H*. We also scale the horizontal distance between points P and Q, d/H. We further refer to the scaled measures of $|PP'|$, $|QQ'|$, and $|P'Q'|$ simply as 1, h, and d, respectively. (P' and Q' are the projections of P and Q on line L.)

Figures 4.3, 4.4, and 4.5 show three possible triangle configurations (one acute-angled and two obtuse-angled) that need to be considered for the purposes of Problem 2.

We denote the distance between points B and P' as x. We then observe the three possible cases of the enclosing triangles (Figures 4.3, 4.4, and 4.5) to find the minimum perimeter solution to Problem 2.

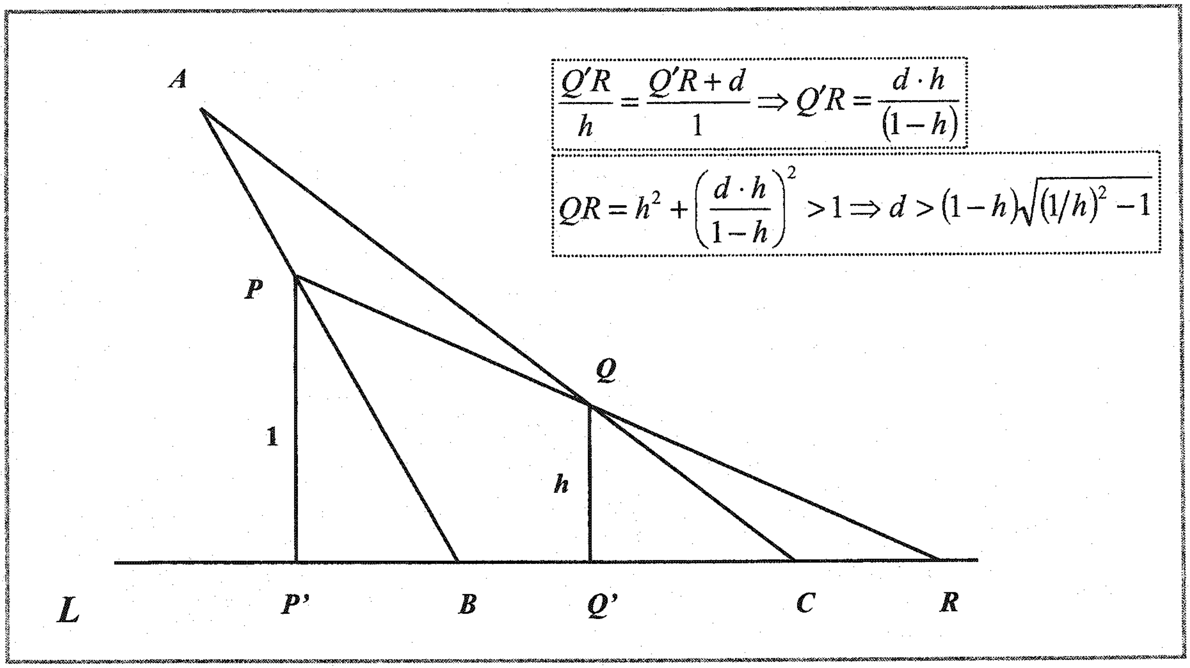

$\frac{Q'R}{h} = \frac{Q'R+d}{1} \Rightarrow Q'R = \frac{d \cdot h}{(1-h)}$
$QR = h^2 + \left(\frac{d \cdot h}{1-h}\right)^2 > 1 \Rightarrow d > (1-h)\sqrt{(1/h)^2 - 1}$
A
P
Q
1
h
L
P'
B
Q'
C
R

Figure 4.6 A degenerate configuration

Before we proceed with our derivation of algebraic expressions, it is important to note that the value of d, the horizontal distance between points P and Q, cannot be arbitrary, and is constrained by the following expression [BM01]:

$$d > (1-h)\sqrt{(1/h)^2 - 1}\,.$$

Clearly, there is no restriction on d when $h = 1$. The above restriction can be derived by examining a degenerate configuration shown in Figure 4.6 below. The idea is that in order to have a minimum perimeter triangle configuration, $|QR| > 1$.

(Expression $Q'R = \frac{d \cdot h}{(1-h)}$ follows from triangle similarity and helps derive our restriction – see Figure 4.6.)

Table 4.1 provides some practical values for the restriction on d.

h	$d >$ than...
0.01	98.995
0.1	8.955
0.2	3.919
0.3	2.226
0.4	1.375
0.5	0.866
0.6	0.533
0.7	0.306
0.8	0.150
0.9	0.048
1	0

Table 4.1 A restriction on the value of d: Examples for some h

4.3 SOLVING THE MAIN SUBSIDIARY PROBLEM FOR A SPECIAL CASE

The simplest case of the main subsidiary problem is when points P and Q are at equal heights over line L (Figure 4.7). In this case, we obtain a simple equation of degree four that gives us the minimum perimeter triangle. This equation is derived based on some triangle similarities observed in the configuration of Figure 4.7 and also on the fact that

$y = x + {}^{d}\!/_{2}$ (Figure 4.7). Here is this equation:

$$x^4 + d \cdot x^3 - {}^{d^2}\!/_{4} = 0 .$$

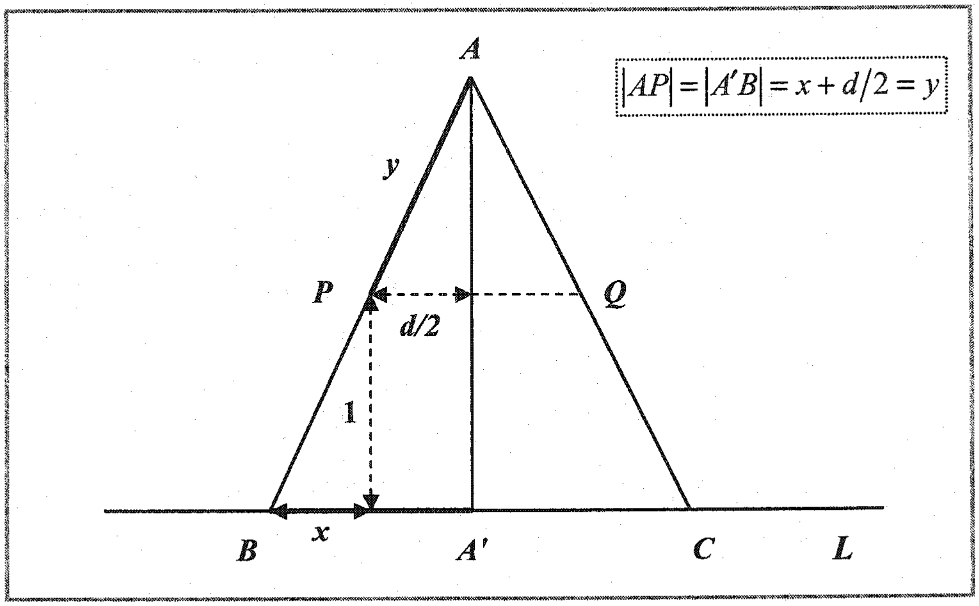


Figure 4.7 A special case: $H = h$

Solving this equation for x (which can be done exactly and in constant time), we obtain a configuration for the minimum perimeter triangle. A general solution to this equation (expressed via d) was derived using a software package *Mathematica 4 for Students* [WWW5]. A printout from Mathematica that contains four roots (two of which are real) is attached to the thesis (Appendix A). Note that one real root that is negative in value (once the corresponding values for variables h and d have been plugged into the solution expressions) does not have geometric meaning, as the only possible minimum perimeter configuration for the special case is one that is acute-angled.

Additionally, some numerical solutions to the equation above are provided in section 4.6.

4.4 DERIVING THE MAIN ALGEBRAIC EQUATION (GENERAL CASE)

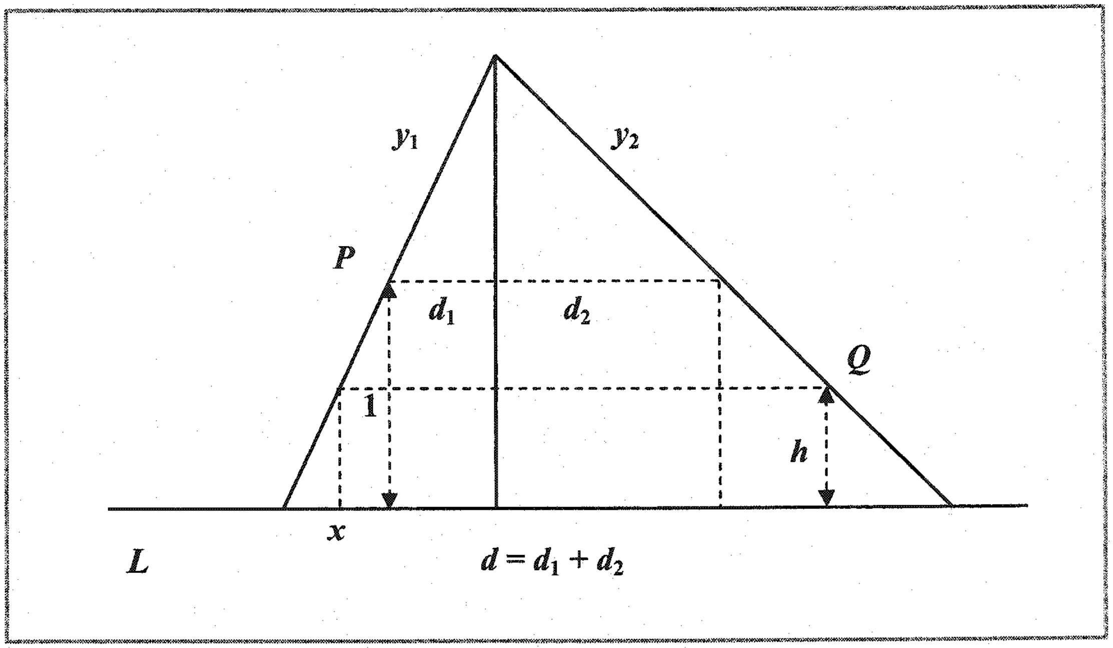


Figure 4.8 A general case: $H > h$ (acute-angled configuration)

1. The first approach

Solving the main subsidiary problem for a general case turned out to be more difficult than for a special case discussed in the previous section. Our first intuitive solution was to express the triangle's perimeter using the two properties of the minimum perimeter configuration proved in Chapter 3 (section 3.2).

The perimeter of the minimum perimeter triangle can be expressed as follows:

$$\mathbf{P} = 2 \cdot \left(x + d + \sqrt{x^2 + 1 - h^2} + \sqrt{x^2 + 1}\right).$$

To minimize the perimeter of the triangle, we took a derivative of this expression with respect to x and performed some simple calculations, obtaining the following equation that related several geometric attributes of the minimal perimeter triangle configuration:

$$3x^8 + 4\cdot\left(2-h^2\right)\cdot x^6 + 6\cdot\left(1-h^2\right)\cdot x^4 + 2h^2 - h^4 - 1 = 0.$$

The last expression corresponds to the acute angle configuration of Figure 4.3. There are two more variations of the expression for the perimeter corresponding to Figures 4.4 and 4.5. They include, instead of (x + d), the (d − x) and (x − d) terms, respectively. All these variations yielded the same solution. This, along with the fact that the resulting solution for x did not depend on d (depended only on h) raised suspicion concerning this first approach. In addition, the approach did not yield any rational roots for a special case when $H = h$.

2. The second approach

In our next approach, we analyzed the configuration depicted in Figure 4.8 (which is the same approach undertaken for a special case in the previous section). The figure possesses a variety of similar triangles, allowing the construction of a system of algebraic equations. Along with the similarity principle, it is also possible to utilize the following expression from simple geometry that connects the *half-perimeter* measure p with the lengths of the triangle's sides a, b, and c, and the height of the triangle's apex over its base a (h_a): $h_a = \frac{2}{a}\sqrt{p(p-a)(p-b)(p-c)}$. We arrived at the following expression for the acute angle configuration in Figure 4.8:

$$(d+x)\cdot\left(\sqrt{1+x^2} - \sqrt{1+x^2-h^2}\right) - \sqrt{\left(1+x^2\right)\cdot\left(1+x^2-h^2\right)} - \left(1+x^2-h^2\right) = 0.$$

This expression simplifies to the previously-derived equation for $h = 1$ (special case):

$$x^4 + d\cdot x^3 - {d^2}/{4} = 0.$$

Using this second approach, it is necessary to derive two more similar expressions for the obtuse angle configurations of Figures 4.4 and 4.5, as in general they are different.

3. Solving the main subsidiary problem for a general case (the third approach)

The weakness of the first and, to some extend, of the second approach is that they only consider the minimum perimeter triangle configuration without trying to characterize the perimeter function in general. In our final approach, we define the perimeter function, *Perimeter* $= f(x)$, and study its behavior as it is determined by the geometric parameters x, d, and h. However, unlike our first and second approaches, we allow this function to represent the perimeter of any circumscribing triangle, not just the minimum perimeter triangle. We define this general function based on the coordinates of points P and Q and of the triangle's vertex B.

We obtain our final expression as follows. We fix points P and Q in the plane and allow point B (the left triangle's apex) to move along the triangle's side BC (Figure 4.9). We transform the coordinate system in such a way that all points along the triangle's side BC have a zero y coordinate. We then begin moving point B from infinity on the left along the side BC in the right direction. As we move point B, we inscribe a excircle that passes through point Q into a wedge formed by the triangle sides BA and BC. The perimeter of the enclosing triangle is defined as twice the distance from the apex of this wedge (point B) to the point of tangency of the inscribed excircle and the extension of the side BC (point Y; see Figure 4.9). We denote the x coordinate of this point of tangency Y as F (the y coordinate of this point is zero as it was mentioned earlier). Furthermore, we notice that F is determined as a function of the x coordinate of point B.

The key idea of our approach is that finding the minimum perimeter triangle implies minimizing the following function:

$$Perimeter = 2(F(x) - x),$$

where x is the coordinate of point B as it moves along the side BC.

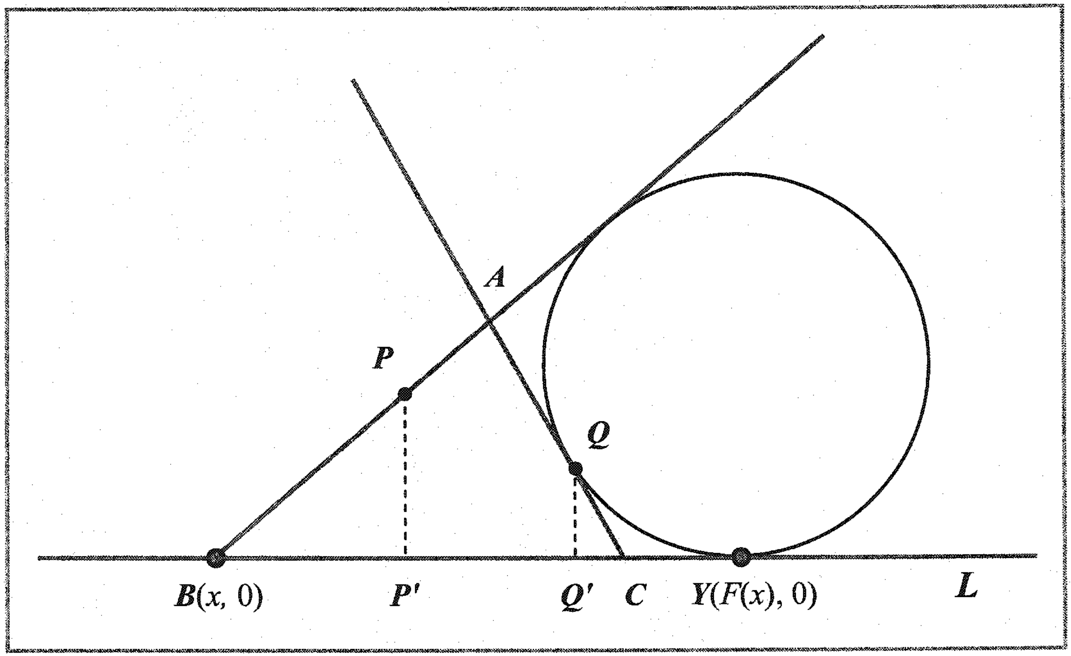


Figure 4.9 The perimeter function: Geometric insights

By inscribing a circle into the wedge *W(BA, BC)* we guarantee that one of the two wedges is always balanced. The other wedge *W(CA, CB)* becomes balanced when the minimum perimeter configuration is achieved. As we move point B from left-to-right, we balance the wedge *W(BA, BC)* by fitting an excircle into it through point Q. This allows us to find the corresponding coordinates of the remaining triangle vertices A and C.

Expressing the *Perimeter* function through the coordinates of points P, Q, and B

allows us to derive the main algebraic equation for Problem 2. It is then possible to transform this equation from the coordinate form into the form that uses already familiar distances x, h, and d. (Please note that the distance x between points B and P' as it was previously defined and the x coordinate of point B, x, are not the same. The "x" notation was chosen in both cases to signify the meaning of the corresponding variables.)

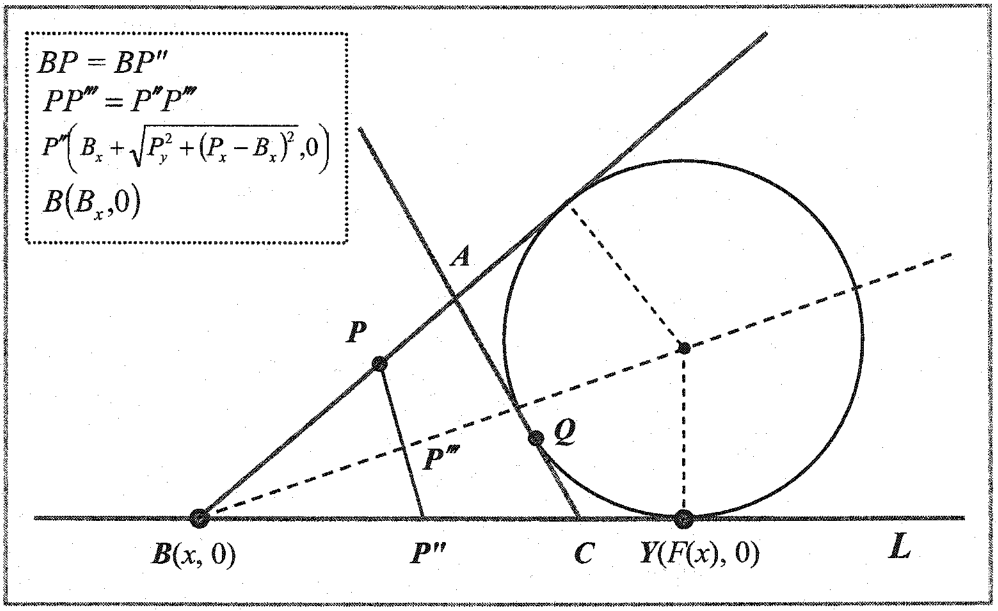


Figure 4.10 The perimeter function: Deriving the main equation

Let us denote the x and y coordinates of points P, Q, B, and Y as $P(P_x, P_y)$, $Q(Q_x, Q_y)$, $B(B_x, B_y)$, and $Y(Y_x, Y_y)$. Note that $B_y = Y_y = 0$. Using the geometry in Figure 4.10, we derive the following expressions involving the coordinates of these points.

$$M' = \frac{P_y}{P_x - B_x + \sqrt{P_y^2 + (P_x - B_x)^2}}, \quad B' = -B_x \frac{P_y}{P_x - B_x + \sqrt{P_y^2 + (P_x - B_x)^2}} = -B_x \cdot M',$$

$$M_1 = \frac{P_y}{P_x - B_x}, \; M_2 = 0, \qquad B_1 = -\frac{B_x P_y}{P_x - B_x} = -B_x \cdot M_1, \; B_2 = 0.$$

The coefficients calculated above are for the equations of the following lines:

$y = M_1 x + B_1$, the equation of a line defined by points $B(B_x, 0)$ and $P(P_x, P_y)$,

$y = M_2 x + B_2$, the equation of the *x* axis, which is incident on the triangle's side *BC*, and

$y = M'x + B'$, the equation of a bisector of the wedge *W(BA, BC)*.

We now have:

$$A = (M_1 \cdot M' + 1)^2$$

$$B = -2 \cdot (M_1^2 + 1) \cdot (Q_x + Q_y \cdot M') + 2M'(B' \cdot M_1^2 + B_1) + 2M_1(B' - B_1)$$

$$C = (M_1^2 + 1) \cdot (Q_x^2 + Q_y^2 - 2Q_y \cdot B') + B'(2B_1 + M_1^2 \cdot B') - B_1^2$$

$$D = B^2 - 4 \cdot A \cdot C$$

And we finally have an equation for the perimeter function expressed through the coordinates of points *P*, *Q*, and *B*:

$$Perimeter(x) = 2 \cdot (F(x) - x) = 2 \cdot (Y_x - B_x) = 2 \cdot \left(\frac{-B \pm \sqrt{D}}{2A} - B_x \right),$$

where the "+" sign needs to be chosen since we are interested in a circle with a bigger radius (an excircle inscribed in a wedge).

The final equation follows from the fact that the *x* coordinate of the center of the excircle is the same as the *x* coordinate of point *Y* (see Figure 4.10).

The equation above can be simplified to the following form:

$$\frac{Perimeter(x)}{2} = Q_x - B_x + \frac{Q_y P_y}{P_x - B_x + \sqrt{P_y^2 + (P_x - B_x)^2}} +$$

$$+ \sqrt{\left(Q_x + \frac{Q_y P_y}{P_x - B_x + \sqrt{P_y^2 + (P_x - B_x)^2}} \right)^2 - \left(Q_x^2 + Q_y^2 + 2 \cdot B_x \frac{Q_y P_y}{P_x - B_x + \sqrt{P_y^2 + (P_x - B_x)^2}} \right)}$$

We can now transform this expression from the coordinate form into the distance form using the following conversions:

$$\begin{gathered} Q_x - P_x = d, \\ P_x - B_x = x, \\ Q_x = d + x + B_x, \\ Q_y = h, \quad P_y = H = 1. \end{gathered}$$

We finally arrive at the following algebraic equation that relates the perimeter of the circumscribing triangle with the geometric parameters *x*, *d*, and *h*.

$$\boxed{\frac{Perimeter(x)}{2} = d + x + \frac{h}{x+\sqrt{x^2+1}} + \sqrt{\frac{h}{x+\sqrt{x^2+1}}\left(\frac{h}{x+\sqrt{x^2+1}} + 2d + 2x\right) - h^2}}$$

Minimizing this expression for the perimeter with respect to *x*, we can obtain a solution to Problem 2, that is the minimum perimeter triangle corresponding to a given combination of parameters *h* and *d*. We thus take a derivative of the last expression and then simplify its result, obtaining:

$$1 - \frac{h \cdot \left(1 + \frac{x}{\sqrt{1+x^2}}\right)}{\left(x+\sqrt{1+x^2}\right)^2} + \frac{\frac{h \cdot \left(2 - \frac{h \cdot \left(1 + \frac{x}{\sqrt{1+x^2}}\right)}{\left(x+\sqrt{1+x^2}\right)^2}\right)}{x+\sqrt{1+x^2}} - \frac{h \cdot \left(1 + \frac{x}{\sqrt{1+x^2}}\right)\left(2d + 2x + \frac{h}{x+\sqrt{1+x^2}}\right)}{\left(x+\sqrt{1+x^2}\right)^2}}{2\sqrt{-h^2 + \frac{h \cdot \left(2d + 2x + \frac{h}{x+\sqrt{1+x^2}}\right)}{x+\sqrt{1+x^2}}}} = 0$$

$$1 - \frac{h}{1+x^2+x\sqrt{1+x^2}} - \frac{h \cdot \left(-1 + h + d \cdot \left(x+\sqrt{1+x^2}\right)\right)}{\sqrt{2}\sqrt{1+x^2}\sqrt{\frac{h \cdot (d + x - h \cdot x)}{x+\sqrt{1+x^2}}}\left(x+\sqrt{1+x^2}\right)^2} = 0 \quad \text{(simplified).}$$

This Mathematica-generated equation can be solved for *x* to obtain the final solution.

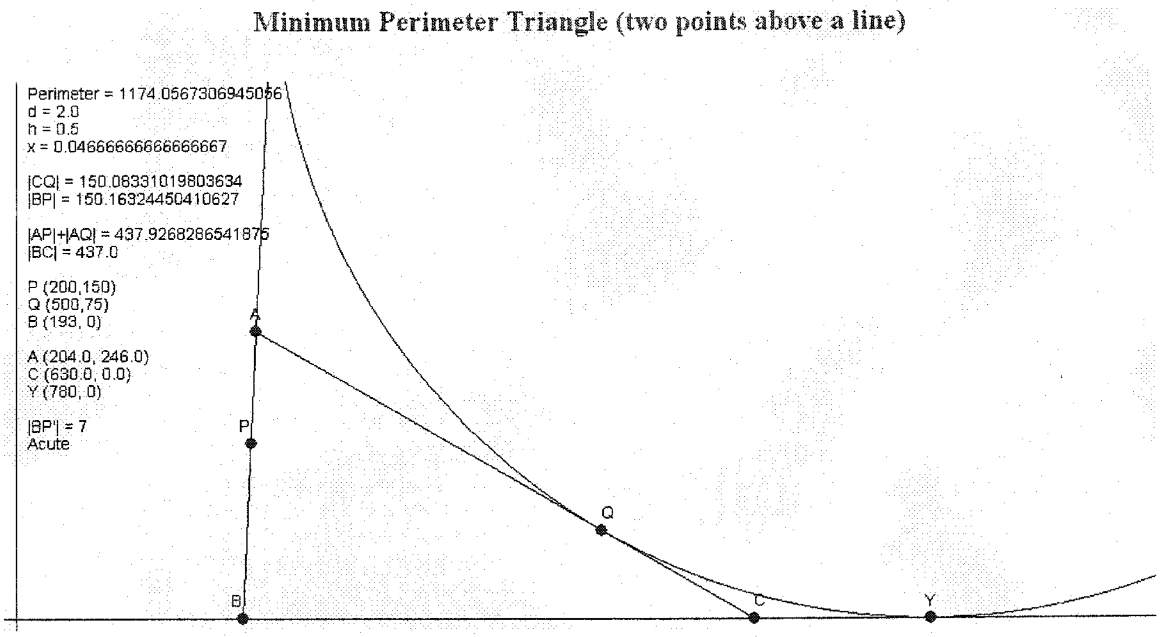
4.5 Analyzing the Main Equation

The perimeter function obtained in the previous section appears to be of high degree, and attempting to simplify its expression (by eliminating the square roots) yields a complex equation with high powers of x. Trying to solve this equation in its general form by using the software package Mathematica did not provide any results. However, when substituting concrete values for variables h and d, we were able to obtain the solutions for the above equation, and therefore characterize the behavior of this function.

It appears evident from our analysis that the perimeter function behaves as follows. It has, in cases when $H > h$ and $d > (1-h)\sqrt{(1/h)^2 - 1}$ (see section 4.2), two extremums, since our equation always has two real roots. Furthermore, after examining these roots, we conclude that one of them corresponds to the minimum perimeter configuration (Figure 4.11), while the other (the bigger root) corresponds to the "maximum" perimeter configuration (Figure 4.12).

These results are consistent with an observation that the two main properties of the minimum perimeter triangle configuration are satisfied twice as we move point B from infinity on the left along the triangle's side BC to the right.

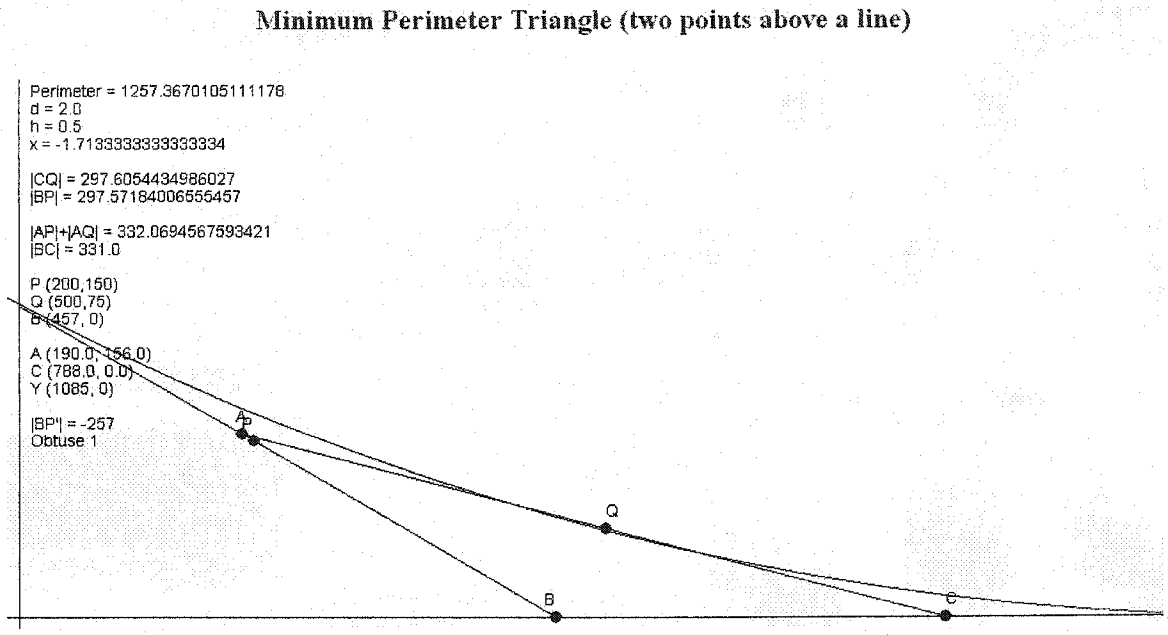
It is necessary to note that the algebraic function for the perimeter obtained in the previous section needs to be restricted in the geometric sense, so that not to allow any degenerate configurations which would violate the statement of Problem 2 (such as point P being interior to the side AB). This restriction can be easily established by noticing that point P remains interior to the triangle's side AB as long as the y coordinate of point A is greater than that of point P. The coordinates of point A can be calculated through the coordinates of points P, Q, and B based on finding the excircle for the wedge $W(BA, BC)$.



Minimum Perimeter Triangle (two points above a line)

```
Perimeter = 1174.0567306945056
d = 2.0
h = 0.5
x = 0.04666666666666667

|CQ| = 150.08331019803634
|BP| = 150.16324450410627

|AP|+|AQ| = 437.9268286541875
|BC| = 437.0

P (200,150)
Q (500,75)
B (193, 0)

A (204.0, 246.0)
C (630.0, 0.0)
Y (780, 0)

|BP'| = 7
Acute
```

Figure 4.11 The minimum perimeter configuration

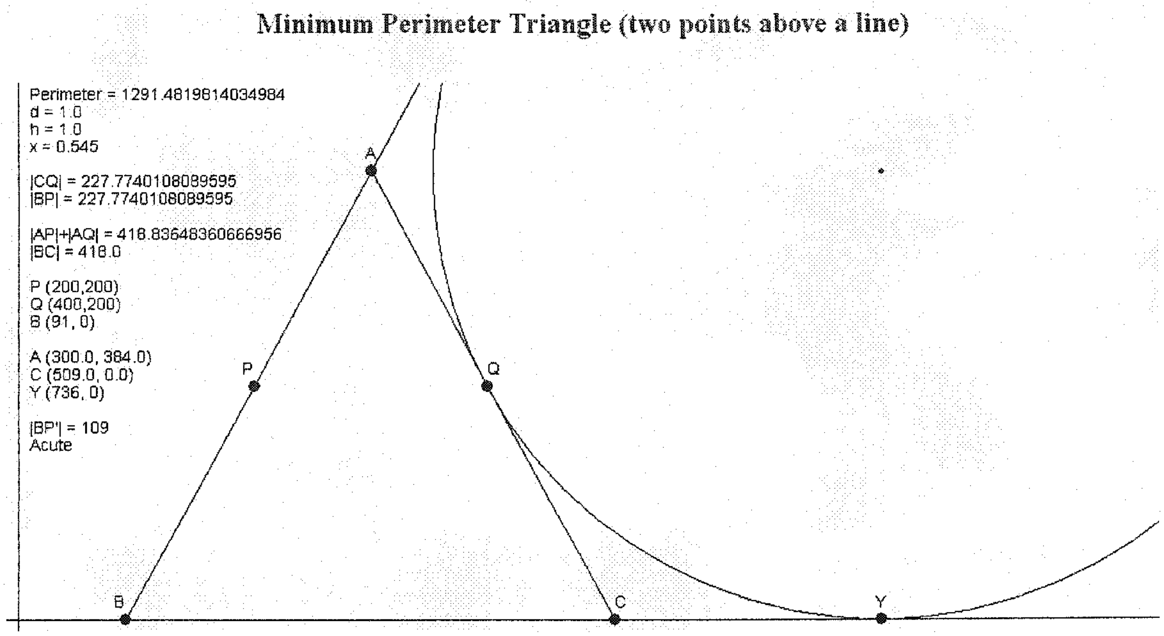


Minimum Perimeter Triangle (two points above a line)

```
Perimeter = 1257.3670105111178
d = 2.0
h = 0.5
x = -1.7133333333333334

|CQ| = 297.6054434986027
|BP| = 297.57184006555457

|AP|+|AQ| = 332.0694567593421
|BC| = 331.0

P (200,150)
Q (500,75)
B (457, 0)

A (190.0, 156.0)
C (788.0, 0.0)
Y (1085, 0)

|BP'| = -257
Obtuse 1
```

Figure 4.12 The maximum perimeter configuration

(Note that point B is fixed when the perimeter is minimal.) Finally, the minimum perimeter properties are satisfied only once for our special case when $h = H$ (Figure 4.13).



Minimum Perimeter Triangle (two points above a line)

```
Perimeter = 1291.4819814034984
d = 1.0
h = 1.0
x = 0.545

|CQ| = 227.7740108089595
|BP| = 227.7740108089595

|AP|+|AQ| = 418.83648360666956
|BC| = 418.0

P (200,200)
Q (400,200)
B (91, 0)

A (300.0, 384.0)
C (509.0, 0.0)
Y (736, 0)

|BP'| = 109
Acute
```

Figure 4.13 Only the minimum perimeter configuration when $h = 1$ ($H = h$)

4.6 Numerical Results

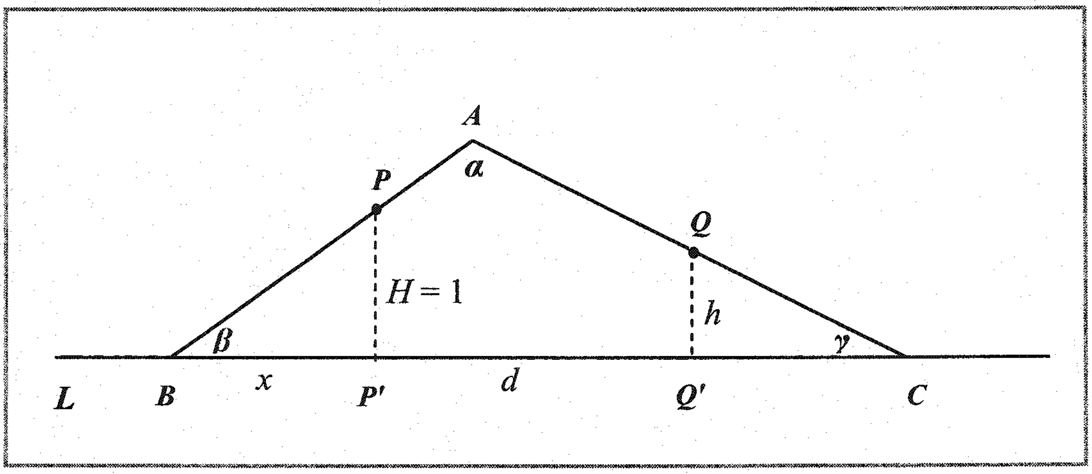


Figure 4.14 Solutions to the main subsidiary problem (triangle angles)

Tables 4.2 and 4.3 show some numerical solutions to the main subsidiary problem (the value of x uniquely identifies a family of triangles with the same angles). In addition, Appendix B contains a variety of numerical results in both, graph and tabular formats.

d	x	$\beta = \gamma$
1/100	0.068337	86.0907°
1/2	0.409586	67.7267°
1	0.544933	61.4126°
2	0.716673	54.3719°
5	1.01297	44.6308°
10	1.30292	37.5065°
15	1.50489	33.6041°
20	1.66500	30.9891°
25	1.79982	29.0570°
30	1.91742	27.5436°
100	2.89632	19.0480°

Table 4.2 Some numerical solutions to the minimum perimeter triangle problem (Special case: $H = h$)

h	d	$x_{\min}$	$\beta_{\min}$	$\gamma_{\min}$	$x_{\max}$	$\beta_{\max}$	$\gamma_{\max}$
0.2	10	0.178	79.9071°	11.3560°	-2.200	155.5560°	4.7473°
0.2	25	0.694	55.2394°	9.4570°	-6.140	170.7497°	1.8424°
0.4	10	0.662	56.4954°	19.4836°	-6.580	171.3586°	3.4456°
0.4	25	1.121	41.7349°	15.4427°	-16.632	176.5592°	1.3756°
0.6	10	0.929	47.1079°	26.0772°	-14.972	176.1788°	2.2916°
0.6	25	1.398	35.5764°	20.4306°	-37.489	178.4720°	0.9167°
0.8	10	1.132	41.4571°	31.9816°	-39.994	178.5677°	1.1458°
0.8	25	1.615	31.7656°	24.9076°	-99.998	179.4270°	0.4584°

Table 4.3 Some numerical solutions to the minimum perimeter triangle problem (General case: $H > h$)

Chapter 5 Implementation of a Linear Time Algorithm for Computing the Minimum Perimeter Triangle Enclosing a Convex Polygon

Our implementation follows the algorithm closely. We find a minimum perimeter triangle for each edge of an n-gon. The polygon edges are considered in clockwise and anticlockwise order, with both implementations available to the user. One of the sides of the minimum perimeter enclosing triangle is always flush with the corresponding polygon edge, a property proved by De Pano [DeP87] and utilized in the implementation. The remaining two sides are found by using the wedge-switching and circle-fitting technique, described in Chapter 3 (section 3.1).

One of the goals of our implementation is providing means for graphical demonstration of the execution of the algorithm. We thus create a graphical user interface (GUI) that displays all the geometric details of the algorithm, as well as handles the user input. Our software system is fairly complex as it interleaves the logic involved in the algorithm and the user interface involved in visualization and animation. The overall software system consists of approximately 8,000 lines of C++ code (including OpenGL and MFC code). Such complexity asks for proper software engineering procedures for the development of our software system, including OOA/OOD, OOP, and testing.

5.1 Object-Oriented Analysis and Design (OOA/OOD)

Figure 5.1 shows a class diagram for our software system that was engineered using Rational Rose 2000 [WWW2]. We will omit the stages of requirements engineering and of OOA/OOD that precede the creation of the class diagram due to space limitations.

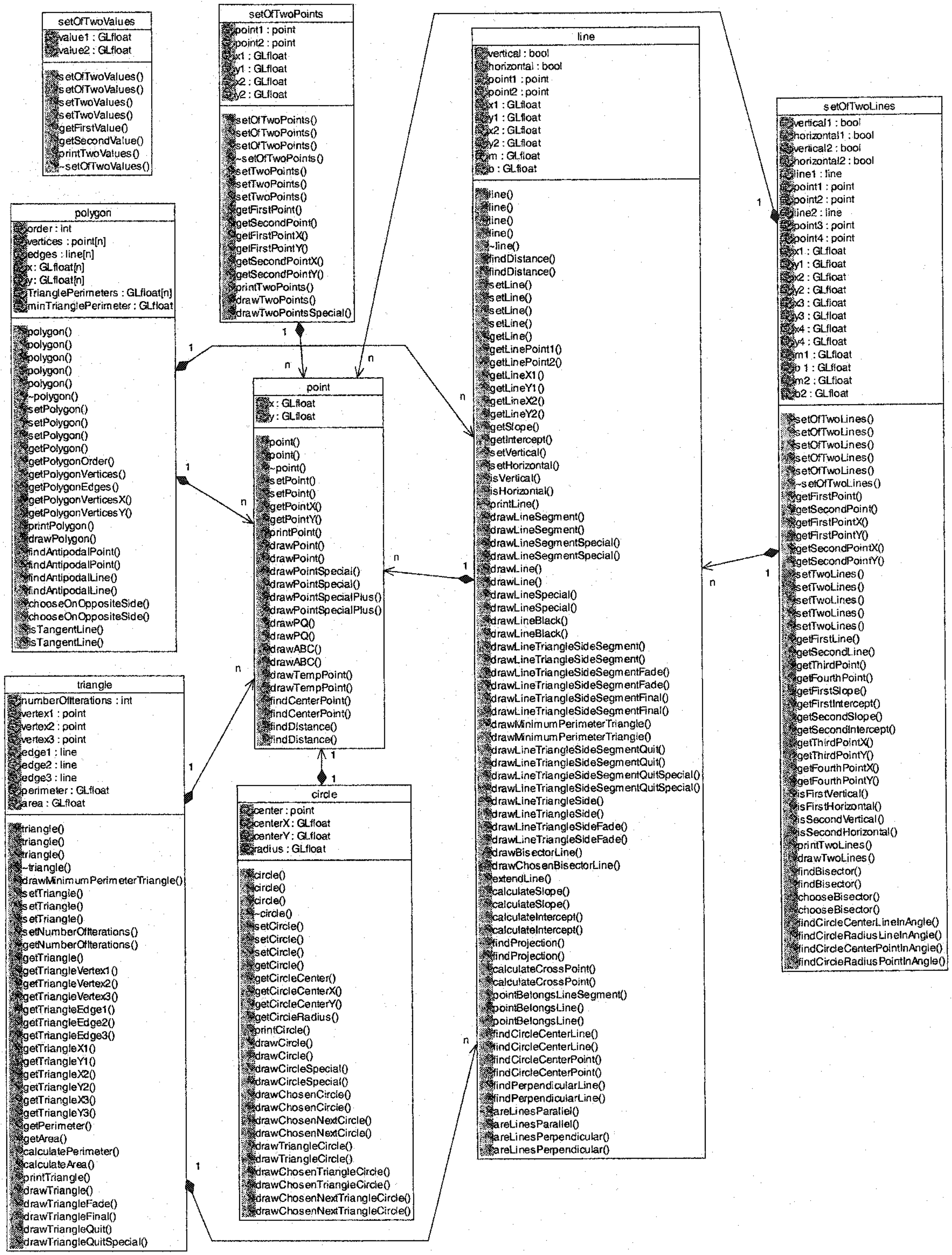


Figure 5.1 Software system class diagram engineered with Rational Rose 2000

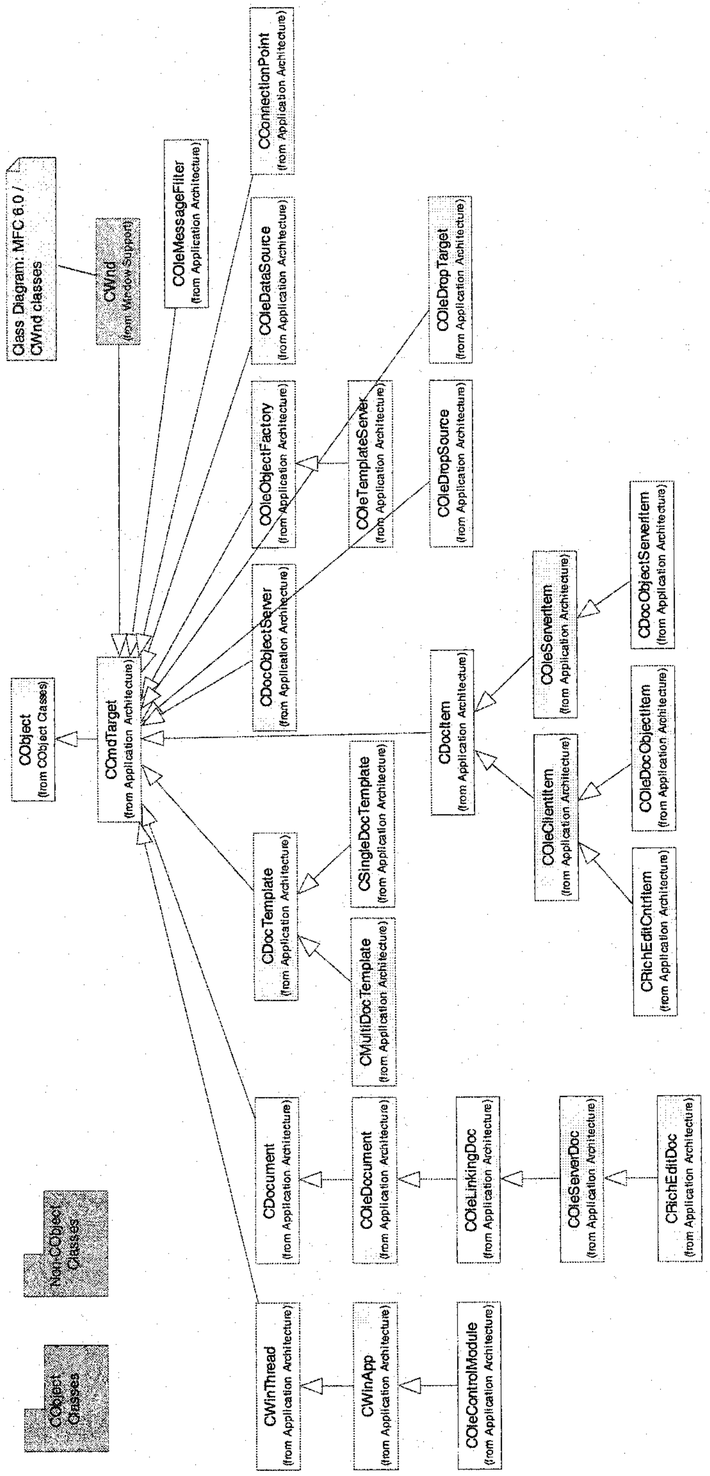

Class Diagram: MFC 6.0 / CWnd classes
CWnd (from Window Support)
COleMessageFilter (from Application Architecture)
CConnectionPoint (from Application Architecture)
COleDataSource (from Application Architecture)
COleDropTarget (from Application Architecture)
COleObjectFactory (from Application Architecture)
COleTemplateServer (from Application Architecture)
COleDropSource (from Application Architecture)
CDocObjectServer (from Application Architecture)
CObject (from CObject Classes)
CCmdTarget (from Application Architecture)
CDocItem (from Application Architecture)
COleServerItem (from Application Architecture)
CDocObjectServerItem (from Application Architecture)
COleClientItem (from Application Architecture)
COleDocObjectItem (from Application Architecture)
CRichEditCntrItem (from Application Architecture)
CSingleDocTemplate (from Application Architecture)
CDocTemplate (from Application Architecture)
CMultiDocTemplate (from Application Architecture)
CDocument (from Application Architecture)
COleDocument (from Application Architecture)
COleLinkingDoc (from Application Architecture)
COleServerDoc (from Application Architecture)
CRichEditDoc (from Application Architecture)
CWinThread (from Application Architecture)
CWinApp (from Application Architecture)
COleControlModule (from Application Architecture)
Non-CObject Classes
CObject Classes

Figure 5.2 Microsoft Foundation Classes (MFC) version 6.0:
A diagram generated with Rational Rose 2000

Figure 5.2 shows a general diagram for the MFC (Microsoft Foundation Classes) library [WWW1] that was used in our implementation to generate part of the GUI. The other part of the GUI was developed using OpenGL [WWW4].

5.2 OBJECT-ORIENTED PROGRAMMING (OOP)

We have implemented our software system in C++ on the Windows XP platform. We used Microsoft's Visual Studio 6.0 [WWW1] with its MFC (Microsoft Foundation Classes) library. In addition, we used OpenGL graphics library, version 1.2 [WWW4] for visualization, animation, and processing of interactive user input.

For the purposes of implementation convenience, we have introduced several C++ classes that correspond to some geometric primitives. The following class specification code shows examples of some classes used in the software system. Note that functions such as accessor functions, OpenGL and MFC drawing/interface functions, constructors and destructors, as well as some other functions are omitted below.

Once the code for all geometric operations is written, we follow the algorithm in invoking the necessary functions to accomplish the tasks specified by the algorithm. As we compute the geometric primitives during the progression of the algorithm (intersection points, antipodal lines, inscribed circles, etc.), we graphically display the intermediate results using OpenGL.

```
class point {
        private:
                GLfloat x;
                GLfloat y;
        public:
                point findCenterPoint(point point1, point point2);
                GLfloat findDistance(point point1, point point2);
                // some member functions omitted
};
```

```
class line {
        private:
                GLfloat m;
                GLfloat b;
        public:
                bool isVertical();
                bool isHorizontal();
                GLfloat findDistance(point anyPoint, line anyLine);
                point findProjection(point anyPoint, line anyLine);
                point calculateCrossPoint(line crossLine, line currentLine);
                bool pointBelongsLineSegment(point anyPoint);
                bool pointBelongsLine(line anyLine, point anyPoint);
                line findPerpendicularLine(point pointOnLine, line anyLine);
                bool areLinesParallel(line anyLine, line currentLine);
                bool areLinesPerpendicular(line anyLine, line currentLine);
                // some member functions omitted
};

class circle {
        private:
                point center;
                GLfloat radius;
        public:
                // member functions omitted
};

class polygon {
        private:
                point vertices[n];
                line edges[n];
        public:
                point findAntipodalPoint(line edge, polygon convexPolygon);
                line findAntipodalLine(line edge, polygon convexPolygon);
                bool isTangentLine(line lineThroughVertex, polygon convexPolygon);
                // some member functions omitted
};
```

5.3 Software System Testing and Testing Results

Reliability of a software system is a measure of success with which the observed behavior of the system conforms to system's specifications. Our expectation for this software system is that it should work for all input instances; that is, for any polygon specified via the Euclidian coordinates of its vertices in the plane.

We have performed the following testing techniques in order to improve the reliability of our software system. As a result of testing, we have been able to detect and eliminate several software system *faults* (or bugs). The following discussion of the testing techniques is based on [Bin00].

1. Black-box testing (testing the system while considering it as a "black box").

Black-box testing tests a software system product against the end user, against external specifications. It is done without any internal knowledge of the product. The test cases for black-box testing can be derived using approaches such as equivalence partitioning, boundary-value analysis, error guessing, and cause-effect graphing. All of these approaches were used to some extend in testing of our software system.

2. White-box testing (testing for possible algorithmic errors).

White-box testing requires knowledge of the internal program structure to derive test cases from the internal design specification or from the code. The test cases are designed based on the algorithmic logic. The goal here is the correct implementation of this logic.

3. OOT (object-oriented testing that discovers faults on the level higher than algorithmic, which is the object-oriented level for the system).

Object-oriented testing strategies include unit testing in the object-oriented context, where the smallest testable unit is a class or an object, and integration testing in the object-oriented context. Both unit and integration testing were performed on our software system. In particular, we used the following object-oriented *test patterns* from [Bin00]: Combinational Function, Invariant Boundaries, Round-trip Scenario, Integration Collaborations, and Extended Use Case.

Chapter 6 IMPLEMENTATION DETAILS

As it was stated in Chapter 5 (section 5.2), our implementation is in C++, and utilizes the OpenGL graphics library [WWW4] for visualization and animation and the MFC library [WWW1] for GUI generation. The proposed implementation computes the minimum perimeter enclosing triangle for any given convex polygon that is input by a user interactively by clicking with a mouse. Also, the user can specify the x and y polygon vertex coordinates by entering them in a form. Finally, several sample polygons are available to the user, so that no user input is required. The algorithm and our implementation run in $O(n)$ time, where n is the number of polygon vertices/edges.

The implementation takes the polygon vertices Euclidian coordinates as input and computes a minimum perimeter enclosing triangle corresponding to each polygon edge. For some edges, such a triangle does not exist [BM01], [BM02]. The implementation proceeds in both anticlockwise and clockwise order when considering polygon edges. It then reports a triangle with the smallest perimeter among the computed triangles.

6.1 CHOOSING AN INPUT POLYGON BY SPECIFYING THE EUCLIDIAN ORTHOGONAL COORDINATES OF ITS VERTICES: USE OF OPENGL GRAPHICS LIBRARY

We use OpenGL graphics library in order to code for the portion of our software system that processes user mouse clicks for entering the vertices of an input polygon. We record the user-entered coordinates in a file for future use and processing.

Figures 6.1 and 6.2 show the screenshots of our OpenGL-generated GUI, which enables entering the coordinates of the polygon vertices interactively.

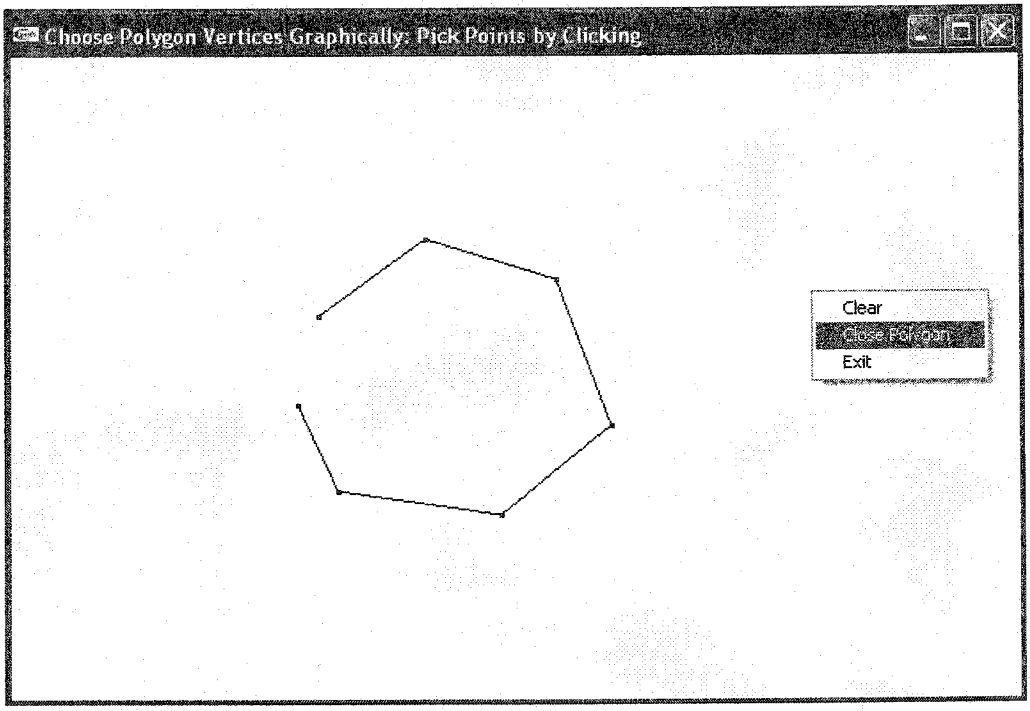

Choose Polygon Vertices Graphically: Pick Points by Clicking
Clear
Close Polygon
Exit

Figure 6.1 Choosing polygon vertices by clicking with a mouse

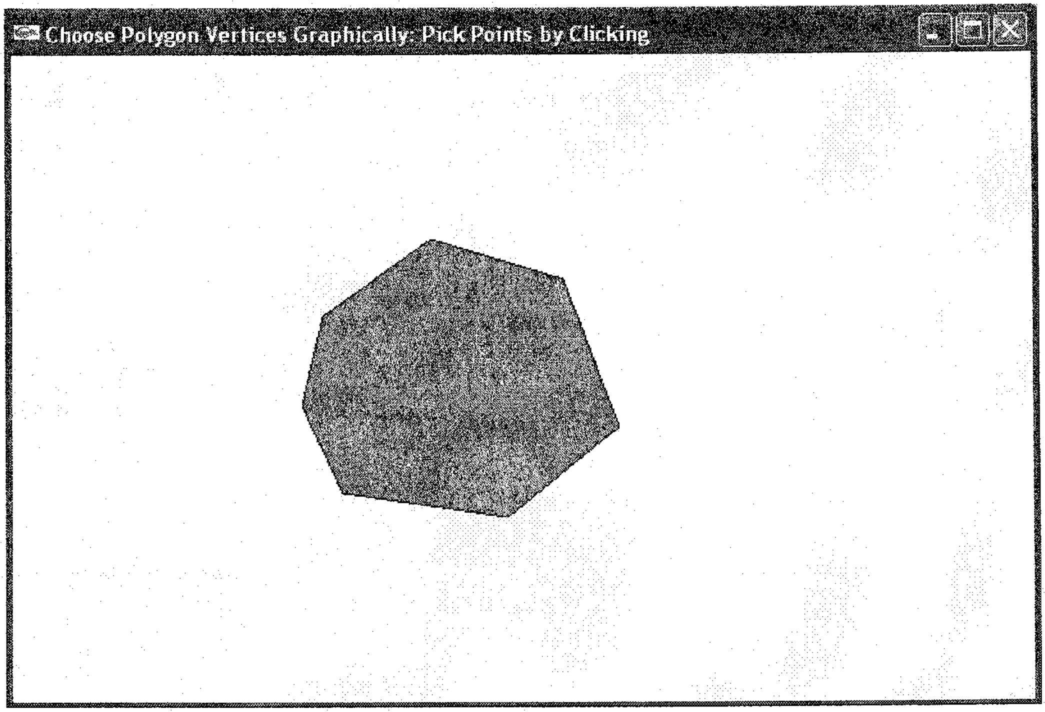

Choose Polygon Vertices Graphically: Pick Points by Clicking

Figure 6.2 A convex polygon is chosen or “closed”

6.2 VERIFYING THE INPUT POLYGON FOR CONVEXITY

There are several different ways to check if a polygon is convex. A trivial approach is to verify that all the inner angles in the polygon are less than 180°. We suggest utilizing the following technique for convexity verification, which is simpler than other evaluated techniques from the implementation point of view.

The magnitude of the cross product of three points A, B, and C on the plane is given by $(X_C - X_B)\cdot(Y_A - Y_B) - (X_A - X_B)\cdot(Y_C - Y_B)$. The idea is that if the magnitudes of every three consecutive vertices are all positive or all negative, the polygon is convex.

6.3 COMPUTING THE MINIMUM PERIMETER TRIANGLE

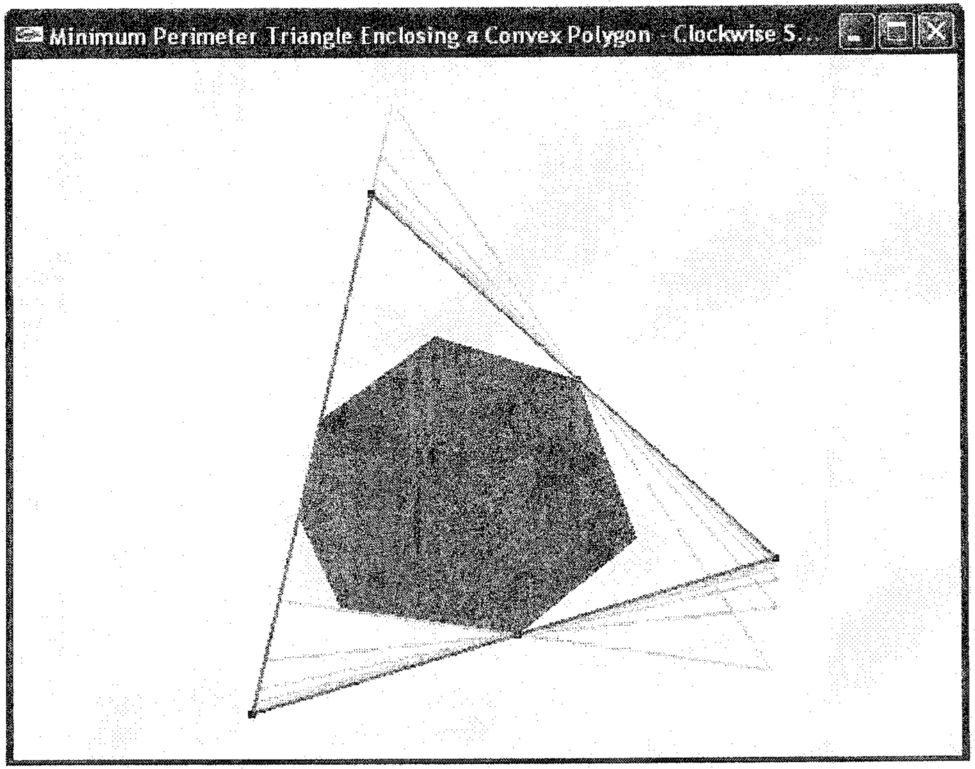

Minimum Perimeter Triangle Enclosing a Convex Polygon - Clockwise S...

Figure 6.3 A minimum perimeter triangle flush with one of the polygon edges (All iterations are shown)

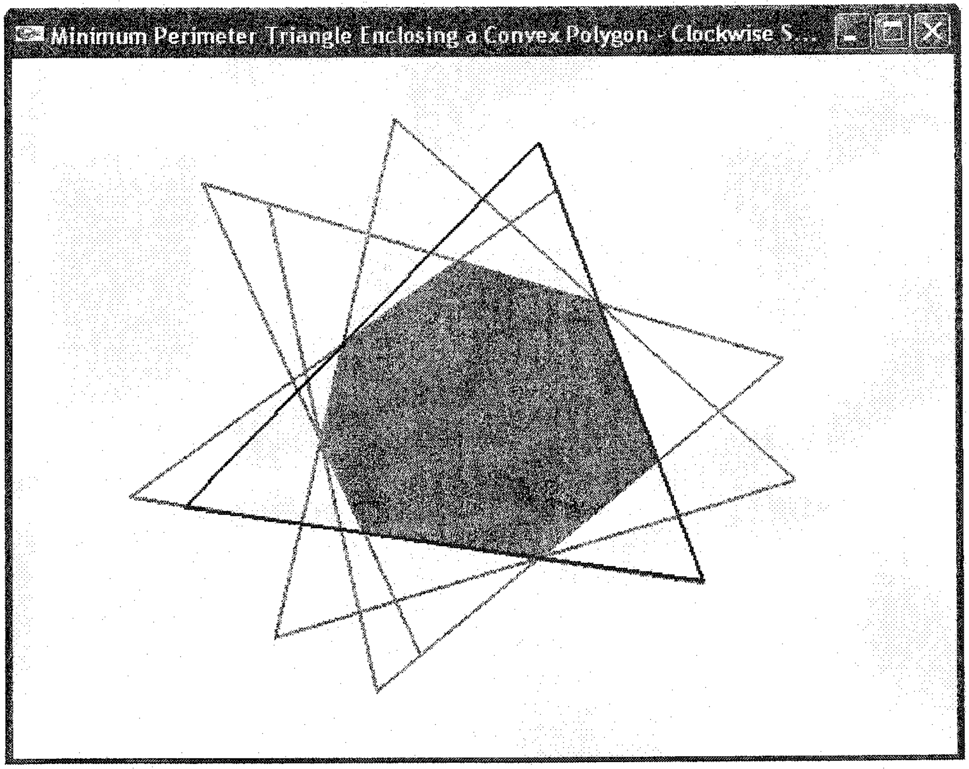

Minimum Perimeter Triangle Enclosing a Convex Polygon - Clockwise S...

Figure 6.4 The minimum perimeter triangle for the polygon is found
(Minimum triangles corresponding to all edges are shown)

Figures 6.3 and 6.4 show how our software system displays the results of the algorithm for computing the minimum perimeter triangle. For every polygon edge, all iterations are displayed (Figure 6.3). This allows us to visualize the process of minimizing the perimeter of the enclosing triangle as the algorithm progresses from one iteration to the next. At the end, the resulting minimum perimeter triangles that correspond to different polygon edges are shown together (Figure 6.4), and the minimum triangle for the polygon is chosen as a minimum among the triangles computed for every polygon edge.

6.4 Graphical Representation of Algorithmic Details: Use of OpenGL

Our software system allows step-wise visualization of geometric details of the algorithm.

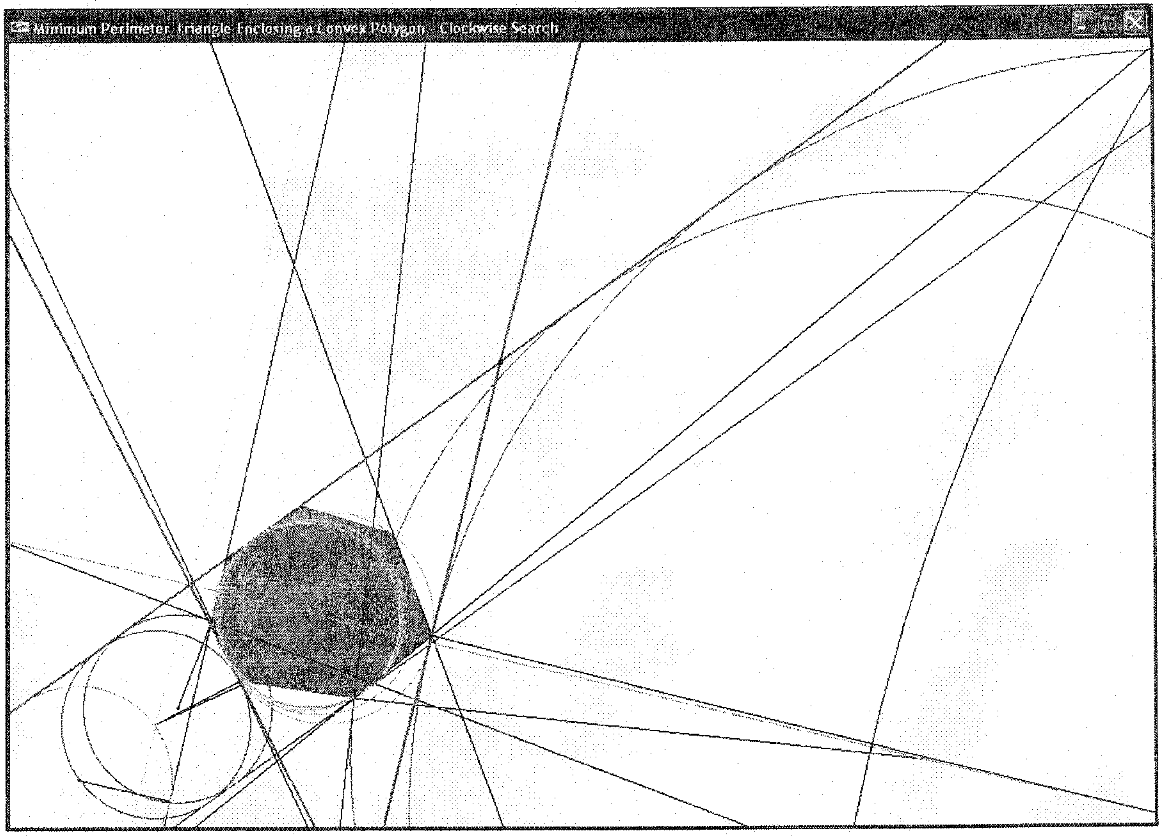

Minimum Perimeter Triangle Enclosing a Convex Polygon - Clockwise Search

Figure 6.5 Geometric details of the linear time algorithm

This visualization is made possible by using OpenGL graphics library. Figure 6.5 screen-shot shows many geometric details displayed by the software system simultaneously.

6.5 Efficiency Issues

The algorithm uses several geometric operations as it progresses in its execution. In particular, it relies on the operations such as finding an antipodal point, finding an excircle for a triangle, computing a point of tangency of a circle and a line, finding a circle inscribed in a wedge, computing a cross point of two lines, and so on. To achieve the maximum *efficiency* for our implementation, we mathematically derived optimal solutions for all operations specified or implied by the algorithm (see Chapter 3).

6.6 Robustness Issues

All coordinates and geometric equation coefficients for this implementation are floating-point numbers. Since we use finite-precision arithmetic, all comparison tests are carried out with respect to an input precision. For instance, for many algorithmic operations, the software system computes lines through their algebraic equations, and checks whether two lines are parallel. In this case, we compare the floating-point coefficients of the line equations not exactly, but rather with certain *precision*, specified by the user. This precision identifies an acceptable error limit beyond which the values being compared are considered indistinguishable. Accordingly, if the absolute value of the difference is greater than the specified precision, the values are considered distinct. Our typical precisions used for most calculations are 0.1, 0.01, and 0.001.

The same precision argument applies to the key comparison operation in the algorithm, which determines the number of wedge-switching and circle-fitting iterations necessary to either terminate the iterations and output the minimum perimeter triangle for the edge under consideration (when $BP = CQ$) or conclude that the current edge cannot generate a minimum perimeter triangle (when the height of P becomes less than of Q).

We make our implementation *robust* by means of introducing this precision property, whose value is used to anticipate and eliminate various problems throughout the program, which typically result from dealing with finite-precision arithmetic.

6.7 Implementation Challenges

The biggest challenge was establishing a tradeoff in specifying the *precision* for different purposes, e.g. for parallel line identification versus computing the difference $|BP| - |CQ|$.

Chapter 7 Evaluation of the Proposed Framework: Algebraic and Experimental Results

7.1 Theoretical Results

We have derived an algebraic expression for the perimeter function in the case of the main subsidiary problem variant. We have also provided the analysis of the behavior of this function, and determined that the function has two extremums (the minimum and the "maximum") in the case when $H > h$ and one extremum (only the minimum) when $H = h$. Due to limitations on the computing power available to us, we were unable to solve the equation for minimization of the perimeter in its general form; however, we succeeded as providing solutions in cases when numerical values of parameters h and d were plugged into the equation. Appendix B contains many such numerical results in graph and tabular formats.

7.2 Implementation and Experimental Results

The key factor in characterizing the efficiency of our implementation is the number of iterations computed for each polygon edge in the process of finding an enclosing triangle of minimum perimeter corresponding to that edge. We performed the following experiments in order to empirically determine whether there is any dependency of the number of such iterations on n. We executed our program for various arbitrary n-gons, where n varied from 4 to 30. Each polygon was specified by the floating-point Euclidian coordinates of its vertices. Table 7.1 shows our results, where "Average" and "Minimum" have the following meaning. We recorded the average of the number of iter-

n	Anticlockwise Search				Clockwise Search			
	Average		Minimum		Average		Minimum	
	0.1	0.01	0.1	0.01	0.1	0.01	0.1	0.01
4	2.75	2.75	2	2	2.00	2.00	2	2
4	1.75	1.75	2	2	1.75	1.75	2	2
4	11.0	14.0	11	14	11.0	14.0	11	14
5	3.00	3.00	3	3	5.00	5.60	3	3
5	4.40	5.40	2	2	3.40	4.20	2	2
5	2.40	2.40	3	3	3.20	3.20	6	6
8	3.25	3.25	3	3	3.25	3.25	4	4
8	2.88	2.88	3	3	6.00	6.88	4	4
8	6.50	8.00	11	14	3.38	3.63	10	12
10	2.90	2.90	3	3	4.20	4.50	3	3
10	5.30	6.30	4	4	4.60	5.40	4	4
10	4.20	4.80	9	12	3.80	4.00	8	10
13	4.08	4.31	3	3	4.08	4.08	4	4
13	3.31	3.69	3	3	4.08	4.77	4	4
13	4.62	5.08	4	4	5.31	6.15	3	3
15	4.33	4.93	3	3	4.93	5.67	4	4
15	4.67	5.20	5	8	4.93	5.60	7	7
15	5.67	6.67	9	12	5.00	5.67	10	13
18	4.56	5.22	5	5	3.44	3.78	4	4
18	5.89	6.83	4	4	4.17	4.33	3	3
18	4.22	4.56	6	6	3.06	3.22	2	2
20	5.20	5.60	4	4	4.95	5.45	3	3
20	4.30	4.60	3	3	3.95	4.25	4	4
20	4.65	5.00	4	4	4.95	5.65	3	3
23	3.91	4.43	3	3	4.57	4.91	4	4
23	4.52	5.17	4	4	4.43	4.91	4	4
23	4.57	5.13	3	3	4.61	5.17	4	4
25	4.92	6.32	3	3	4.96	5.96	3	3
25	4.88	5.72	3	3	3.76	4.16	4	4
25	5.76	6.52	4	4	4.20	4.76	4	4
28	5.07	5.79	9	11	4.29	4.75	6	7
28	5.32	6.04	3	3	4.75	5.25	4	4
28	4.54	4.75	6	6	5.79	6.43	4	4
30	5.07	5.77	4	4	4.80	5.37	4	4
30	5.13	5.83	11	14	5.43	6.17	10	13
30	5.03	5.60	10	13	5.37	6.13	4	4

Table 7.1 Experimental results showing the number of iterations for different *n*-gons

ations, "Average", that were needed to compute a minimum perimeter triangle for each polygon edge (the average was computed over all polygon edges). We also recorded the number of iterations, "Minimum", that corresponded to the edge that produced the minimum perimeter triangle for the polygon. We performed both, anticlockwise and clockwise searches of the polygon edges. The averages and the "minimum" values were computed for both of the edge searches. The number of polygon vertices *n* and the polygon vertex coordinates were chosen arbitrarily.

Our experiments indicate that the number of iterations necessary to compute a minimum perimeter triangle for a polygon edge does not depend on the number of edges *n*, and is rather determined but the geometry of the input polygon, as well as by the desired calculation precision. This result agrees with the algorithm's linear time complexity and shows that this complexity can be achieved experimentally.

Our results also show that the algorithm often finds the same minimum perimeter triangle whether we consider polygon edges in anticlockwise or clockwise order. The number of iterations may slightly differ for anticlockwise and clockwise solutions, which is particularly true for polygons with no geometric symmetry. We also observe that the final minimum perimeter triangle for the polygon is often (in our experiments) flush with the longest polygon side (especially for large *n*). Finally, we find that our resulting minimum perimeter triangle for the polygon is not dependent on our choice of precision.

The proposed implementation in its current version (version 1.0) works for $3 \leq n \leq 1000$ polygon edges and allows a maximum of 100 iterations for each polygon edge. Minimizing or eliminating the GUI component of the software system can substantially improve its potential. We retain the GUI at this time for demonstration purposes.

Chapter 8 CONCLUSION AND FUTURE WORK

8.1 CONCLUSION

First, we have derived an algebraic equation that expresses the perimeter of the enclosing triangle via several geometric parameters of the enclosed degenerate object, formed by a pair of points above a given line. Although due to its high degree, the equation cannot be solved precisely in its general form, we have been able to provide several concrete solutions to the equation by plugging the constant values for the equation parameters h and d. These numerical results (Appendices A and B) indicate that the perimeter function for the main subsidiary problem variant has exactly two extremums (when $H > h$). We further claim that the first extremum is associated with the minimum perimeter triangle configuration, whereas the second corresponds to the "maximum" perimeter configuration, both satisfying the two main properties of the minimum perimeter triangle (derived in Chapter 3). When $H = h$, we have one extremum (only the minimum).

Second, we have given an implementation of a linear time algorithm for computing the minimum perimeter triangle enclosing a convex polygon. This implementation has been shown to be reliable for various input instances. Asymptotically, our implementation is more efficient than previous solutions for accomplishing the same task. Our software system also achieves flexibility by allowing the user to specify the precision to which arithmetic comparison tests are to be performed. It appears that the precision factor determines the number of iterations needed for every polygon edge while computing its corresponding minimum perimeter triangle candidate, if one exists. Thus, our implementation controls its own constant factor by means of specifying the precision variable. We have demonstrated that the average number of iterations is small and rarely exceeds 10 for precisions of 0.1 and 0.01 and for polygons with up to 30 edges. This once again illustrates that the number of iterations for each polygon edge is at most a fraction of n, and the overall number of iterations for all edges of the polygon is on the order of n. We have shown that the resulting minimum perimeter triangle is not dependent on whether we first consider the polygon edges in clockwise or anticlockwise order. We have also observed that the final minimum perimeter triangle for the polygon is often (in our experiments) flush with the longest polygon side. Finally, we have demonstrated that the resulting minimum perimeter triangle is not dependent on our choice of precision value.

Our implementation is efficient and complies with the algorithm's linear time complexity while achieving a small constant factor. We have verified empirically that the number of iterations necessary for computing a minimum perimeter enclosing triangle corresponding to each polygon edge – when such a triangle exists – is a factor which is at most a fraction of n for each polygon edge. The sum of these factors over all iterations for n polygon edges is on the order of n, implying the linearity of the algorithm. The space complexity of our algorithm and implementation is $O(n)$ and the amount of extra space used in the implementation is $O(1)$ if we omit the GUI component.

Our use of OpenGL and MFC makes the implementation graphical and interactive.

8.2 FUTURE WORK

Future work related to the problem of finding the minimum perimeter triangle enclosing a convex polygon can be projected in the following two directions. First, we may consider the same and related problems in two dimensions (such related problems can include any problems concerning optimization of geometric objects in 2D). Second, our proposed framework may also be generalized to higher dimensions. The following two sections summarize these possible directions for future work.

8.2.1 FUTURE WORK IN TWO DIMENSIONS

A natural question is can we do better than $O(n)$? Our experimental results indicate that in many cases, especially when n is relatively large, the resulting minimum perimeter triangle is flush with the longest polygon edge. Does this mean that we only have to consider one edge? In general, the answer is no. However, in applications that allow a certain level of imprecision as a tradeoff for increased efficiency, we may want to consider computing only the minimum perimeter triangle that corresponds to the longest polygon edge. The result may well be the minimum perimeter triangle for the polygon. At any rate, it appears to be difficult to characterize the time complexity of our approach if it is only applied to one polygon edge, since the number of iterations for any edge can be on the order of n.

Another interesting question is how to generalize an enclosing triangle to an arbitrary k-gon? Also, how to generalize a convex polygon to a simple polygon? The latter question can be answered in the practical sense by convexifying a non-convex polygon (by calculating its corresponding convex hull) and then applying the algorithm to the resulting convex polygon.

Yet other important questions to investigate are what the connection to the minimum area triangle is? Which enclosing triangle is a better choice? Intuitively, the answer to the latter question is expected to be application-dependent.

8.2.2 GENERALIZATION TO HIGHER DIMENSIONS

In general, it has been difficult to generalize such problems to higher dimensions. Even in 3D (three dimensions), the problem statement for finding the minimum perimeter triangle changes to that of finding a tetrahedron of minimal surface area that encloses a given convex polyhedron. Alternatively, one may wish to minimize the total edge length of the enclosing tetrahedron. The minimum area triangle problem in 2D (two dimensions) would correspondingly transform into finding a minimum volume bounding tetrahedron in 3D. Naturally, even a 3D-extension imposes new challenges, since the approaches that perform well in 2D are not sufficient in higher dimensions, and their generalization may not be feasible. In higher dimensions (*d*-dimensions), the problem of finding the minimum perimeter triangle transforms into a problem of finding a *d*-dimensional tetrahedron that encloses a given convex polytope. Again, the approaches from 2D are not readily generalized to *d*-dimensional space.

BIBLIOGRAPHY

[Aga97] P. Agarwal, *Range searching*, In Handbook of Discrete and Computational Geometry, Eds. J. E. Goodman and J. O'Rourke, pp. 575–598, CRC Press, Boca Raton, FL, 1997.

[AAEFV98] P. Agarwal, L. Arge, J. Erickson, P. Franciosa, and J. Vitter, *Efficient searching with linear constraints*, In Proceedings of the ACM Symposium on Principles of Database Systems, Vol. 17, pp. 169–178, 1998.

[AE99] P. Agarwal and J. Erickson, *Geometric range searching and its relatives*, In B. Chazelle, J. E. Goodman, and R. Pollack, Eds., Advances in Discrete and Computational Geometry, Vol. 23 of Contemporary Mathematics, American Mathematical Society Press, Providence, RI, pp. 1–56, 1999.

[AS98] P. K. Agarwal and M. Sharir, *Efficient algorithms for geometric optimization*, ACM Computing Surveys, Vol. 30, pp. 412–458, 1998.

[ACY85] A. Aggarwal, J. S. Chang, and C. K. Yap, *Minimal area circumscribing polygons*, Special issue of Visual Computer: International Journal of Computer Graphics, 1985.

[AP88] A. Aggarwal and J. K. Park, *Notes on searching in multi-dimensional monotone arrays*, FOCS, pp. 497–512, 1988.

[AB00] S. Alstrup and G. Brodal, *New data structures for orthogonal range searching*, IEEE Symposium on Foundations of Computer Science, pp. 198–207, 2000.

[ABR00] S. Alstrup, G. Brodal, and T. Rauhe, *Optimal static range reporting in one dimension*, The IT University of Copenhagen, Technical Report, November 2000.

[AEIIM85] T. Asano, M. Edahiro, H. Imai, M. Iri, and K. Murota, *Practical use of bucketing teckniques in computational geometry*, 1985.

[BKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational geometry – algorithms and applications*, Springer-Verlag, 1997.

[BM01] B. Bhattacharya and A. Mukhopadhyay, *A linear time algorithm for computing the minimum perimeter triangle enclosing a convex polygon*, Proceedings of the 17th European Workshop on Computational Geometry (EUROCG-01), pp. 43–47, Institute of Computer Science, Freie Universität, Berlin, 26–28 March 2001.

[BM02] B. Bhattacharya and A. Mukhopadhyay, *On minimum perimeter triangle enclosing a convex polygon*, Japan Conference on Discrete and Computational Geometry, December 2002.

[Bin00] R. V. Binder, *Testing object-oriented systems: Models, patterns, and tools*, Addison-Wesley, 2000.

[BDDG85] J. E. Boyce, D. P. Dobkin, R. L. Drysdale, and L. Guibas, *Finding extremal polygons*, SIAM J. Comput., Vol. 14, pp. 134–147, 1985.

[CY84] J. S. Chang and C. K. Yap, *A polynomial solution for potato-peeling and other polygon inclusion and enclosure problems*, In 25th Annual Symposium on Foundations of Computer Science, pp. 408–416, Singer Island, Florida, IEEE, 24–26 October 1984.

[CR96] B. Chazelle and B. Rosenberg, *Simplex range reporting on a pointer machine*, Comput. Geom. Theory Appl., Vol. 5, pp. 237–247, 1996.

[DeP84] A. De Pano, *Approximations of polygons and polyhedra: Potentials for research*, Manuscript, 1984.

[DeP87] N. A. A. De Pano, *Polygon approximation with optimized polygonal enclosures: applications and algorithms*, Ph. D. Thesis, 1987.

[DA84] A. De Pano and A. Aggarwal, *Finding restricted K-envelopes for convex polygons*, Proc. of the 22nd Allerton Conference on Comm. Control and Computing, 1984.

[DS79] D. P. Dobkin and L. Snyder, *On a general method for maximizing and minimizing among certain geometric problems*, Proc. IEEE Symp., 20th FOCS, pp. 9–17, 1979.

[DB83] D. Dori and M. Ben-Bassat, *Circumscribing a convex polygon by a polygon of fewer sides with minimal area addition*, Computer Vision Graphics and Image Processing, Vol. 24, pp. 131–159, 1983.

[FS75] H. Freeman and R. Shapira, *Determining the minimum area enclosing rectangle for an arbitrary closed curve*, CACM, Vol. 18, pp. 409–413, 1975.

[GG98] V. Gaede and O. Günther, *Multidimensional access methods*, ACM Computing Surveys, 30, 2 (June), pp. 170–231, 1998.

[Gru83] P. M. Gruber, *Approximation of convex bodies*, In Convexity and Its Applications, Editor P. M. Gruber, Birkhauser, 1983.

[KS85] J. M. Keil and J.-R. Sack, *Minimum decompositions of polygonal objects*, 1985.

[KL85] V. Klee and M. C. Laskowski, *Finding the smallest triangles containing a given convex polygon*, J. Algorithms, Vol. 6, pp. 359–375, 1985.

[MM03] A. V. Medvedeva and A. Mukhopadhyay, *An implementation of a linear time algorithm for computing the minimum perimeter triangle enclosing a* *convex polygon*, Manuscript, School of Computer Science, University of Windsor, 30 April 2003.

[ORo84] J. O'Rourke, *Finding minimal enclosing boxes*, Technical Report, Dept. of Electrical Engineering and Computer Science, The Johns Hopkins University, 1984.

[OAMB86] J. O'Rourke, A. Aggarwal, S. Madilla, and M. Baldwin, *An optimal algorithm for finding minimal enclosing triangles*, Journal of Algorithms, Vol. 7, pp. 258–269, 1986.

[PS85] F. P Preparata and M. I. Shamos, *Computational geometry An introduction*, Springer-Verlag, 1985.

[RV73] A. W. Roberts and D. E. Varberg, *Convex functions*, Academic Press, 1973.

[Tou79] G. T. Toussaint, *Pattern recognition and geometric complexity*, 5th International Conference on Pattern Recognition, pp. 1324–1347, 1979.

[Tou83] G. T. Toussaint, *Solving geometric problems with the "rotating calipers"*, Proceedings IEEE MELECON '83, Athens, Greece, 1983.

[Tou85] G. T. Toussaint, editor, *Computational Geometry*, Elsevier Science Publishers B. V., 1985.

[Vit01] J. Vitter, *External memory algorithms and data structures: dealing with massive data*, ACM Computing Surveys, Vol. 33, No. 2, (June), pp. 209–271, 2001.

[Wil82] D. Willard, *Polygon retrieval*, SIAM J. Comput., Vol. 11, pp. 149–165, 1982.

[Zie99] M. Ziegelmann, *External memory computational geometry state of the art*, 1999.

REFERENCED LINKS

[WWW1] Microsoft Corporation, *Visual C++ version 6.0* (*Microsoft Visual Studio® version 6.0*), http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcedit98/HTML/vcstartpage.asp (as of May 2003).

[WWW2] Rational Software Corporation, *Rational Rose®* (*Rational Rose 2000*), http://www.rational.com (as of May 2003).

[WWW3] Richard Anderson (a personal website), *Our Northern India Tour*, Dept. of Computer Science and Engineering, University of Washington, Seattle, http://www.cs.washington.edu/homes/anderson/india/delhi.html (as of May 2003).

[WWW4] Silicon Graphics, Inc., *OpenGL®* (*OpenGL version 1.2*), http://www.opengl.org (as of May 2003).

[WWW5] Wolfram Research, Inc., *Mathematica® 4 for Students*, http://www.wolfram.com (as of May 2003).

Appendix A: Some Algebraic Results Generated with Mathematica 4

Special case of the main subsidiary problem: *h* = *H*.

```
Solve[x^4 + (x^3) *d - (d^2) / 4 == 0, x]
```

$$\Bigg\{\Bigg\{x\to -\frac{d}{4}+\frac{1}{2}\sqrt{\frac{d^2}{4}-\frac{2\,d^2}{3^{1/3}\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}+\frac{\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}{2\,3^{2/3}}}-\frac{1}{2}\sqrt{\frac{d^2}{2}+\frac{2\,d^2}{3^{1/3}\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}-\frac{\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}{2\,3^{2/3}}-\frac{d^3}{4\sqrt{\frac{d^2}{4}-\frac{2d^2}{3^{1/3}\left(-9d^4-\sqrt{3}\,d^3\sqrt{64+27d^2}\right)^{1/3}}+\frac{\left(-9d^4-\sqrt{3}\,d^3\sqrt{64+27d^2}\right)^{1/3}}{2\,3^{2/3}}}}}\Bigg\},$$

$$\Bigg\{x\to -\frac{d}{4}+\frac{1}{2}\sqrt{\frac{d^2}{4}-\frac{2\,d^2}{3^{1/3}\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}+\frac{\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}{2\,3^{2/3}}}+\frac{1}{2}\sqrt{\frac{d^2}{2}+\frac{2\,d^2}{3^{1/3}\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}-\frac{\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}{2\,3^{2/3}}-\frac{d^3}{4\sqrt{\frac{d^2}{4}-\frac{2d^2}{3^{1/3}\left(-9d^4-\sqrt{3}\,d^3\sqrt{64+27d^2}\right)^{1/3}}+\frac{\left(-9d^4-\sqrt{3}\,d^3\sqrt{64+27d^2}\right)^{1/3}}{2\,3^{2/3}}}}}\Bigg\},$$

$$\Bigg\{x\to -\frac{d}{4}-\frac{1}{2}\sqrt{\frac{d^2}{4}-\frac{2\,d^2}{3^{1/3}\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}+\frac{\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}{2\,3^{2/3}}}-\frac{1}{2}\sqrt{\frac{d^2}{2}+\frac{2\,d^2}{3^{1/3}\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}-\frac{\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}{2\,3^{2/3}}+\frac{d^3}{4\sqrt{\frac{d^2}{4}-\frac{2d^2}{3^{1/3}\left(-9d^4-\sqrt{3}\,d^3\sqrt{64+27d^2}\right)^{1/3}}+\frac{\left(-9d^4-\sqrt{3}\,d^3\sqrt{64+27d^2}\right)^{1/3}}{2\,3^{2/3}}}}}\Bigg\},$$

$$\Bigg\{x\to -\frac{d}{4}-\frac{1}{2}\sqrt{\frac{d^2}{4}-\frac{2\,d^2}{3^{1/3}\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}+\frac{\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}{2\,3^{2/3}}}+\frac{1}{2}\sqrt{\frac{d^2}{2}+\frac{2\,d^2}{3^{1/3}\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}-\frac{\left(-9\,d^4-\sqrt{3}\,d^3\sqrt{64+27\,d^2}\right)^{1/3}}{2\,3^{2/3}}+\frac{d^3}{4\sqrt{\frac{d^2}{4}-\frac{2d^2}{3^{1/3}\left(-9d^4-\sqrt{3}\,d^3\sqrt{64+27d^2}\right)^{1/3}}+\frac{\left(-9d^4-\sqrt{3}\,d^3\sqrt{64+27d^2}\right)^{1/3}}{2\,3^{2/3}}}}}\Bigg\}\Bigg\}$$

<u>d = 1/2</u>

```
NSolve[x^4 + (x^3) / 2 - 1/ 16 == 0, x]
```

```
{{x→ -0.690139}, {x→ -0.109724 - 0.457237 i}, {x→ -0.109724 + 0.457237 i}, {x→ 0.409586}}
```

<u>d = 2</u>

```
NSolve[x^4 + (x^3) *2 - 1 == 0, x]
```

```
{{x→ -2.10692}, {x→ -0.304877 - 0.754529 i}, {x→ -0.304877 + 0.754529 i}, {x→ 0.716673}}
```

<u>d = 1</u>

```
NSolve[x^4 + x^3 - 1/ 4 == 0, x]
```

```
{{x→ -1.16012}, {x→ -0.192409 - 0.598692 i}, {x→ -0.192409 + 0.598692 i}, {x→ 0.544933}}
```

Appendix B: Numerical Solutions to the Main Subsidiary Problem in Graph and Tabular Formats

Recall that the main subsidiary problem for the minimum perimeter enclosing triangle asks to find a triangle of minimum perimeter that contains two given points P and Q (located above a given line L) on two of its sides. The height of point P over line L (over the third triangle's side) is equal to or greater than the height of point Q, and P is located to the left of Q (Figure A.1).

Appendix B contains some numerical results for the main subsidiary problem that were determined algebraically. These results are displayed in tables and also in graphs of the following format:

$$Perimeter = f(x),$$

where the values for the perimeter are plotted against the values of x.

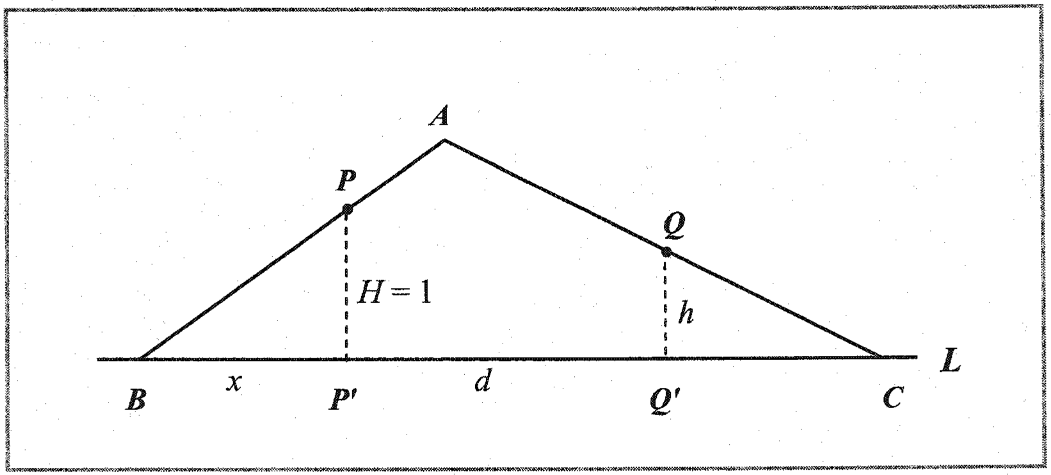


Figure A.1 Geometric meaning of h, d, and x

The following series of graphs and tables show how the perimeter of the enclosing triangle changes with respect to x, the distance from the left triangle's vertex B

to the projection of point *P* on side *BC*, *P'* (Figure A.1).

The two parameters that determine the triangle's perimeter are the distances *h* and *d* (for the sake of generosity and simplicity we have scaled all measures by *H*). The parameters *h* and *d* varied in the intervals (0, 1] and [5, 30], respectively. Specifically, the perimeter function was determined for the following combinations of values of *h* and *d*:

$$h = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1;$$
$$d = 5, 10, 15, 20, 25, 30.$$

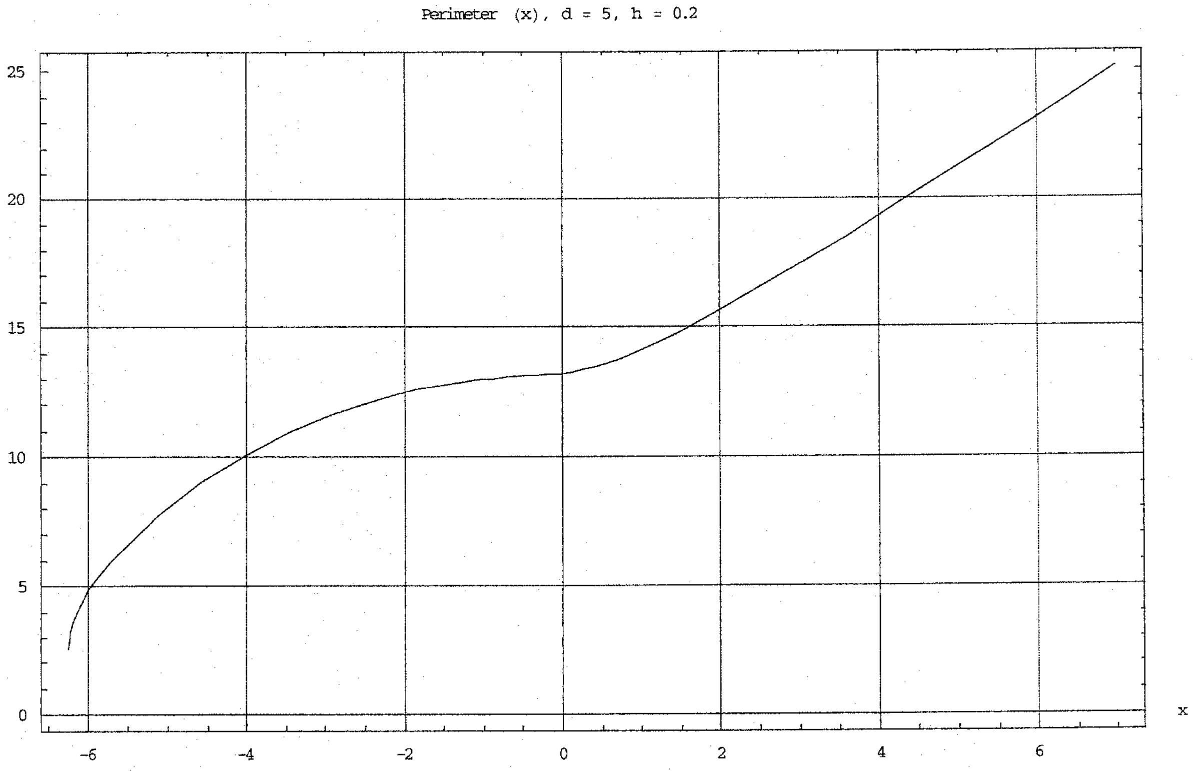
Due to a restriction imposed on the value of *d* as related to *h* (see section 4.2), $d > (1-h)\sqrt{(1/h)^2 - 1}$, some combinations of the mentioned *h* and *d* may not be represented, as they can not generate a minimum perimeter triangle. Also, the graphs are provided only for $h = 0.2, 0.4, 0.6, 0.8, 1$.

Table A.1 summarizes the numerical results for the main subsidiary problem. It lists the perimeter values that are scaled by *H*, similarly to all other values that express distances in Figure A.1. This allows providing the perimeter values for a family of triangles. Every triangle in a family has the same values for all three angles.

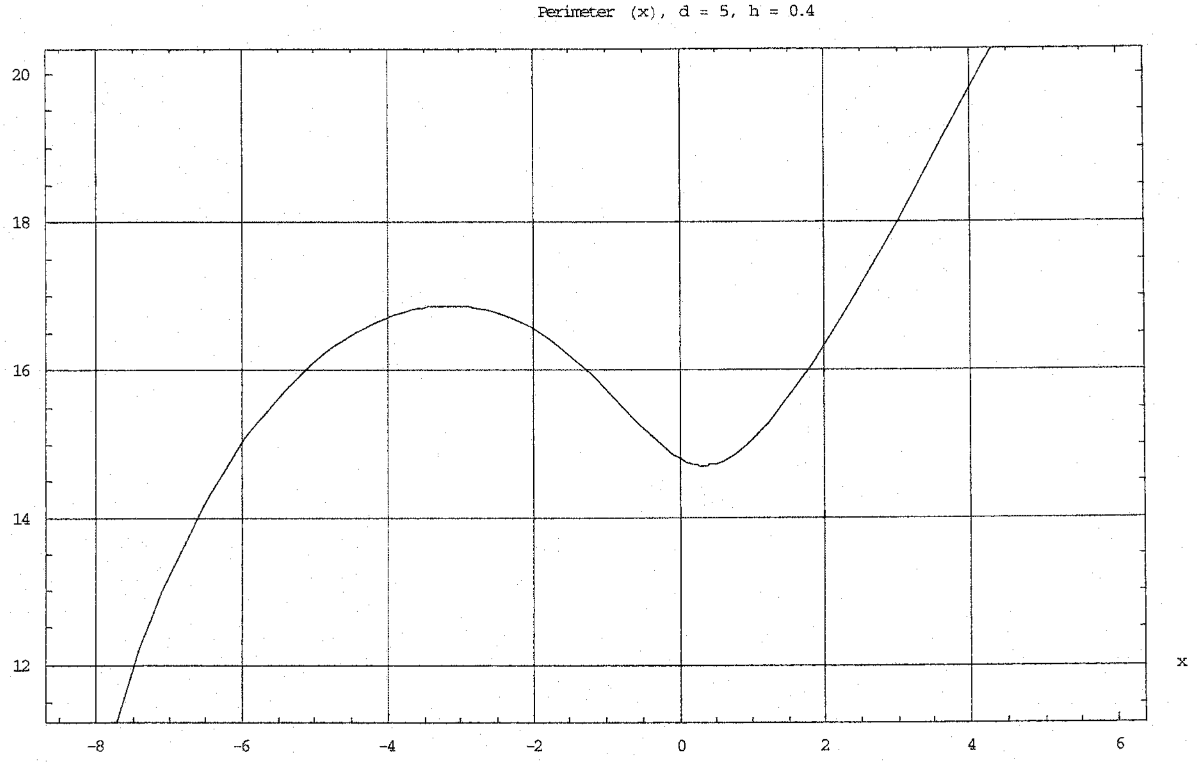
In our analysis of the perimeter function, we have found that the conditions |BP| = |CQ| and |AP| + |AQ| = |BC| are satisfied in two cases: the minimum and the maximum perimeter triangle configurations. This can be seen from the graphs in this Appendix, whose functions always (except when $h = 1$) possess two extremums (the minimum and the maximum).

The following graphs were generated using Mathematica 4 for Students [WWW5].

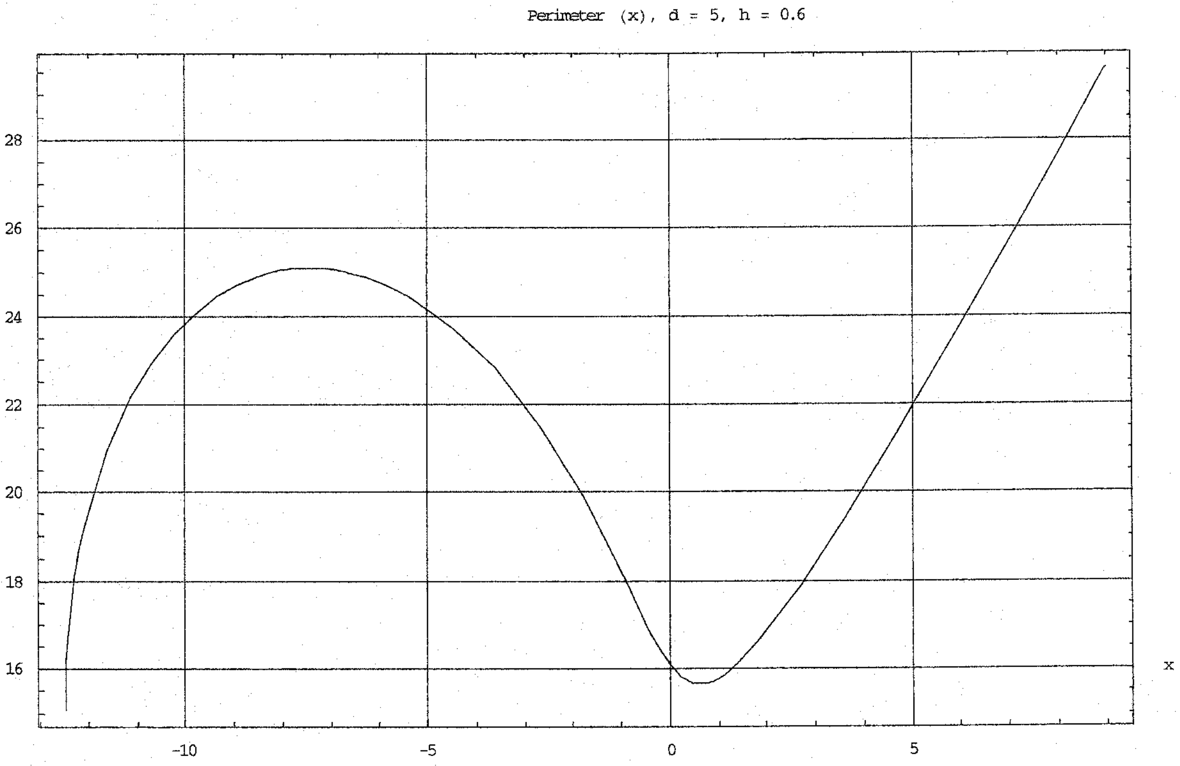
Perimeter(x), $d = 5$, $h = 0.2$ (no extremums in this case, i.e. no minimum or maximum)



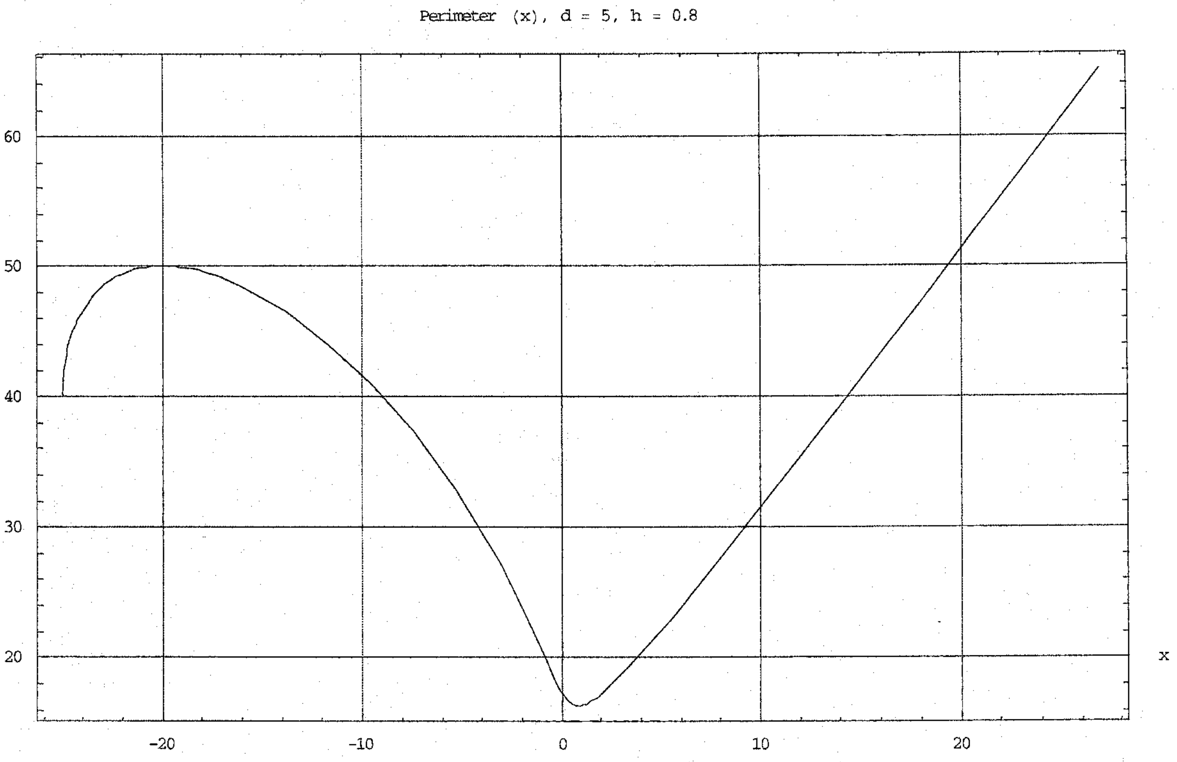
Perimeter(x), $d = 5$, $h = 0.4$



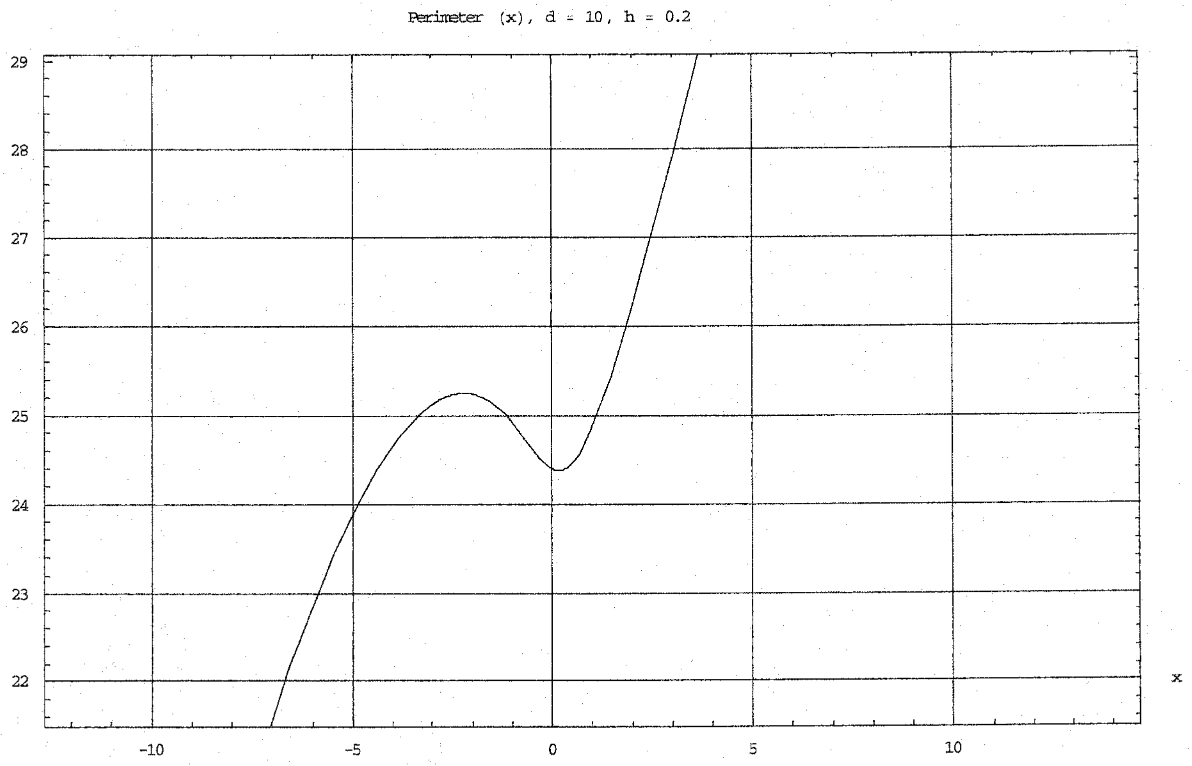
Perimeter(*x*), *d* = 5, *h* = 0.6



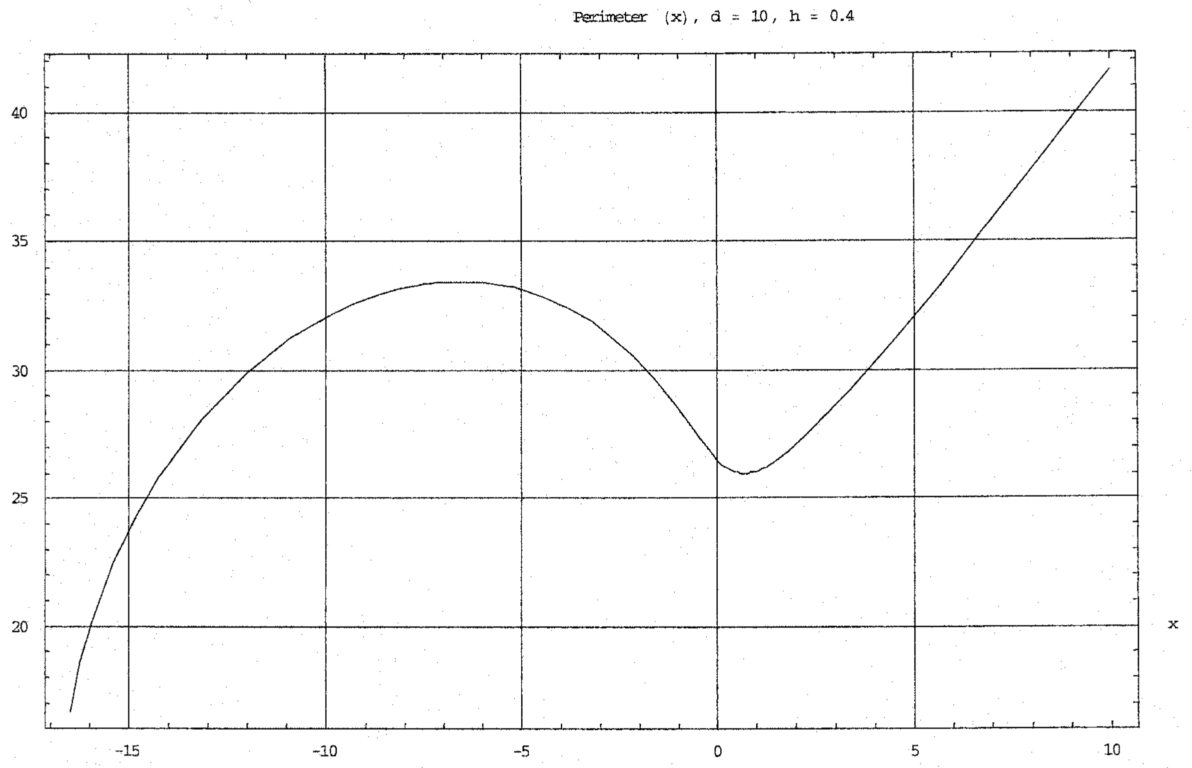
Perimeter(*x*), *d* = 5, *h* = 0.8



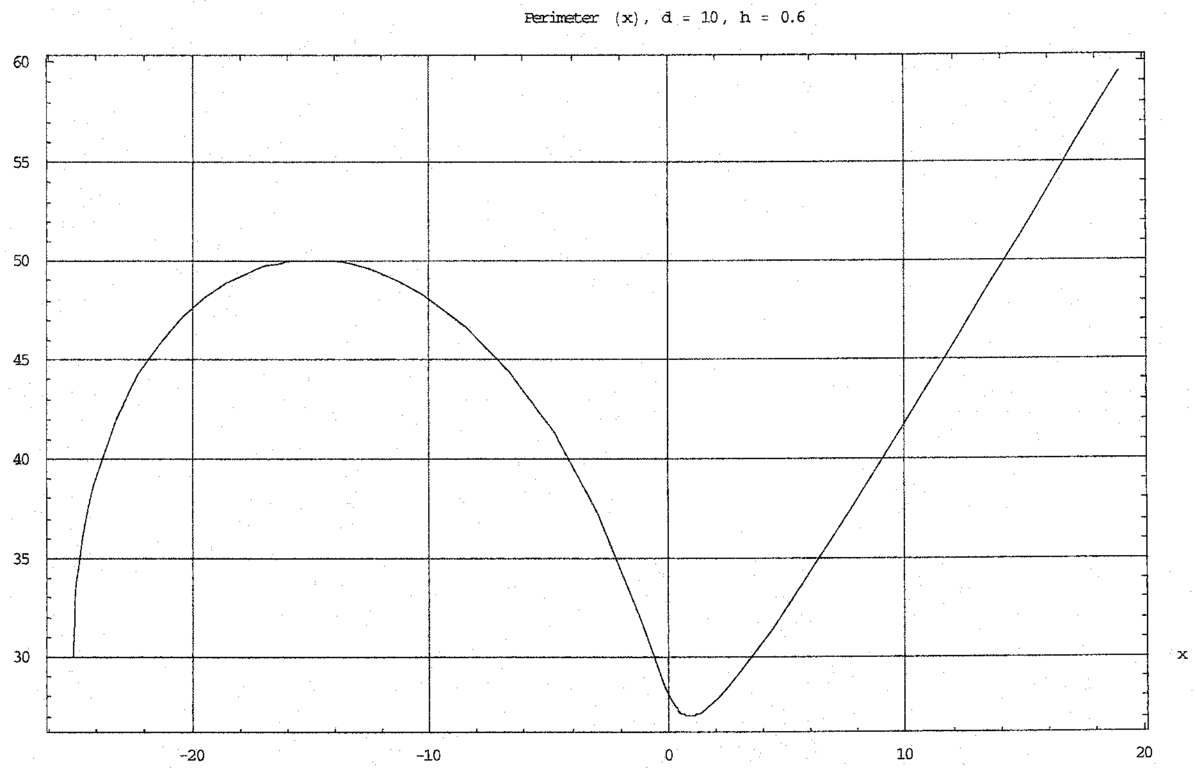
Perimeter(x), $d = 10$, $h = 0.2$



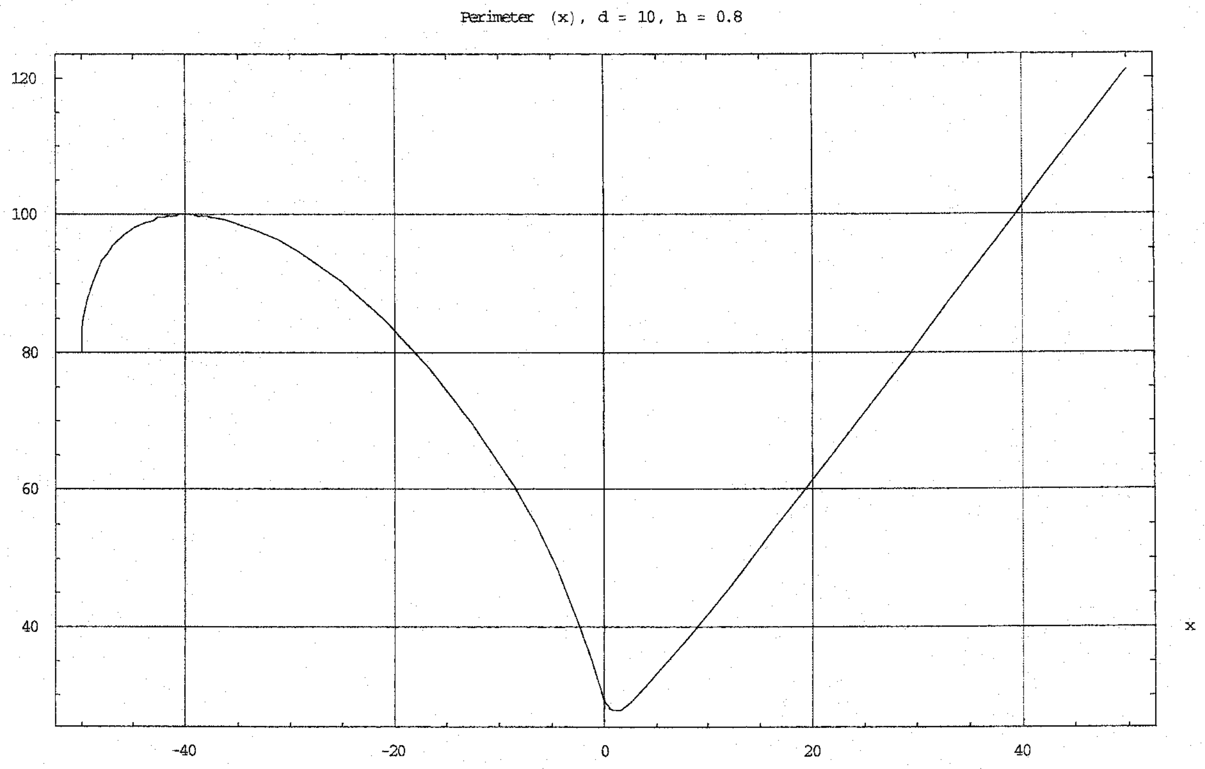
Perimeter(x), $d = 10$, $h = 0.4$



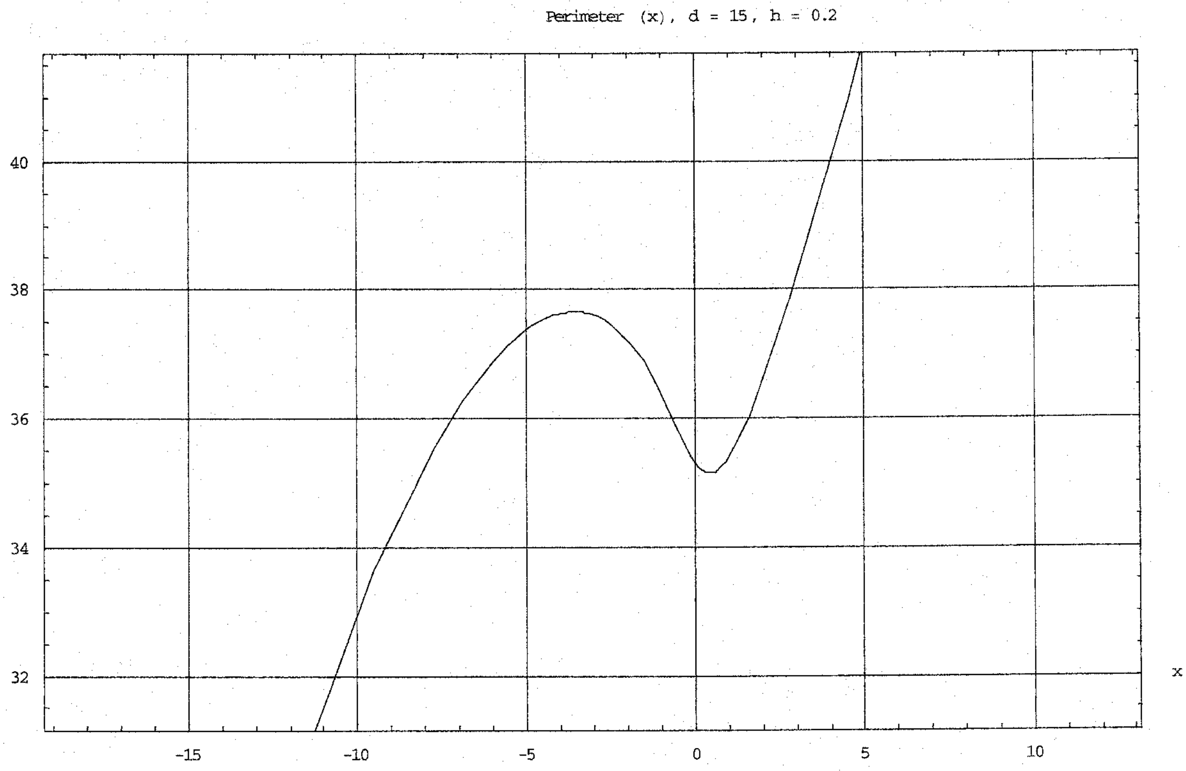
Perimeter(x), $d = 10$, $h = 0.6$



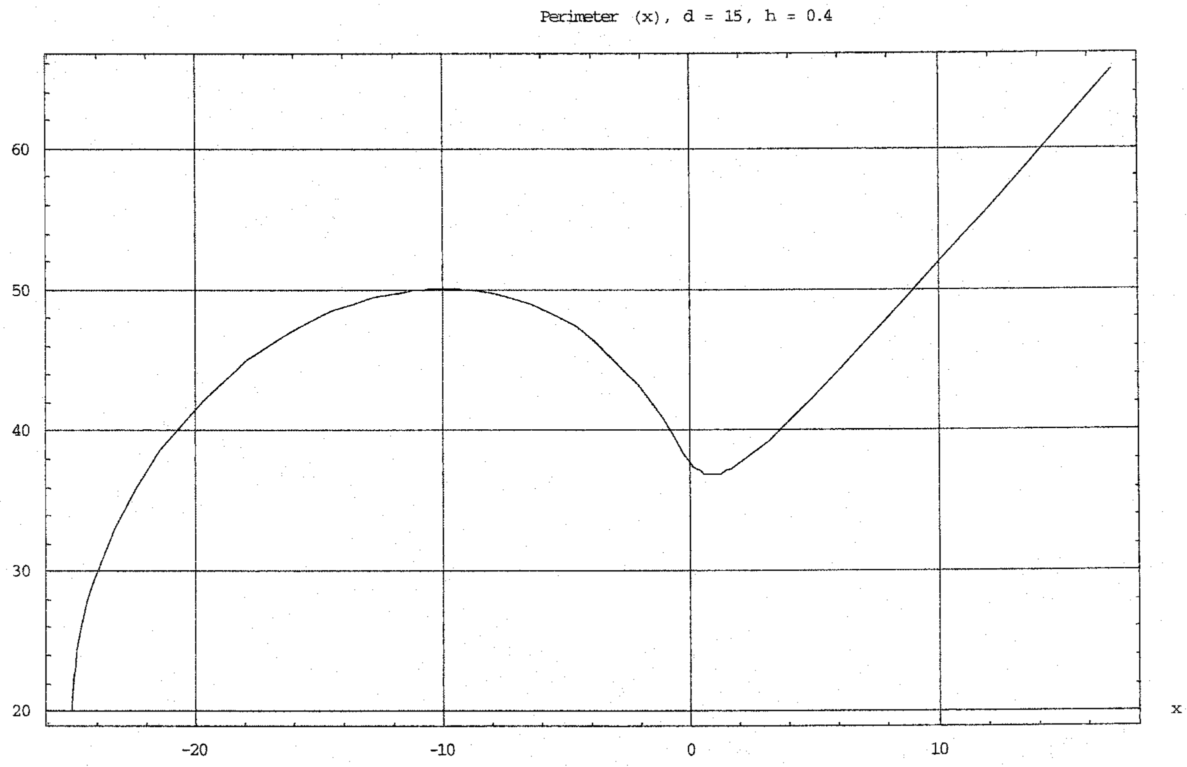
Perimeter(x), $d = 10$, $h = 0.8$



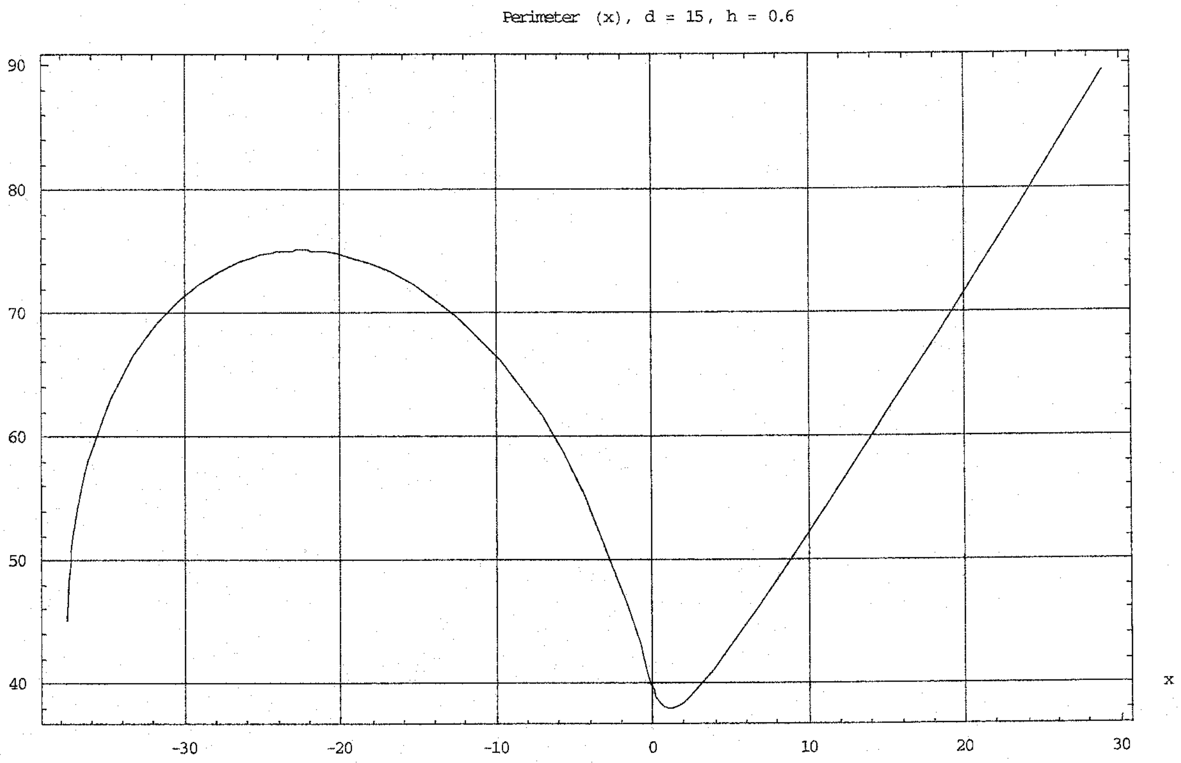
Perimeter(*x*), *d* = 15, *h* = 0.2



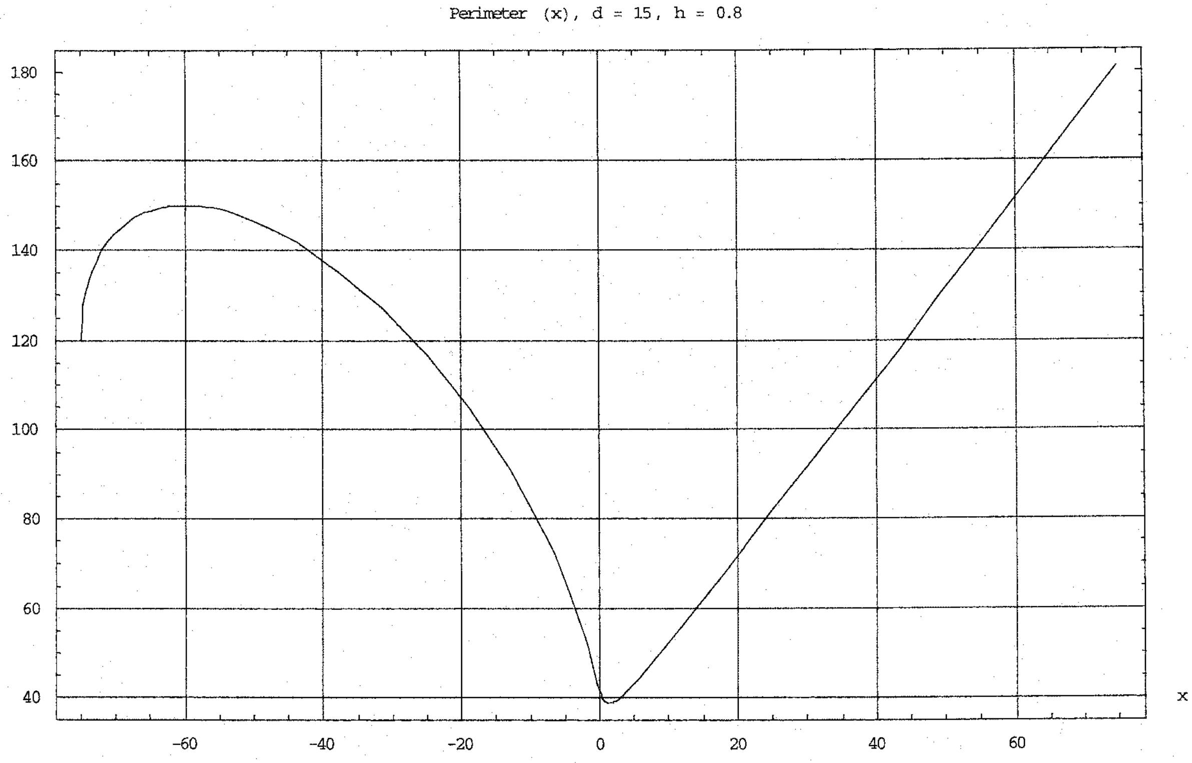
Perimeter(*x*), *d* = 15, *h* = 0.4



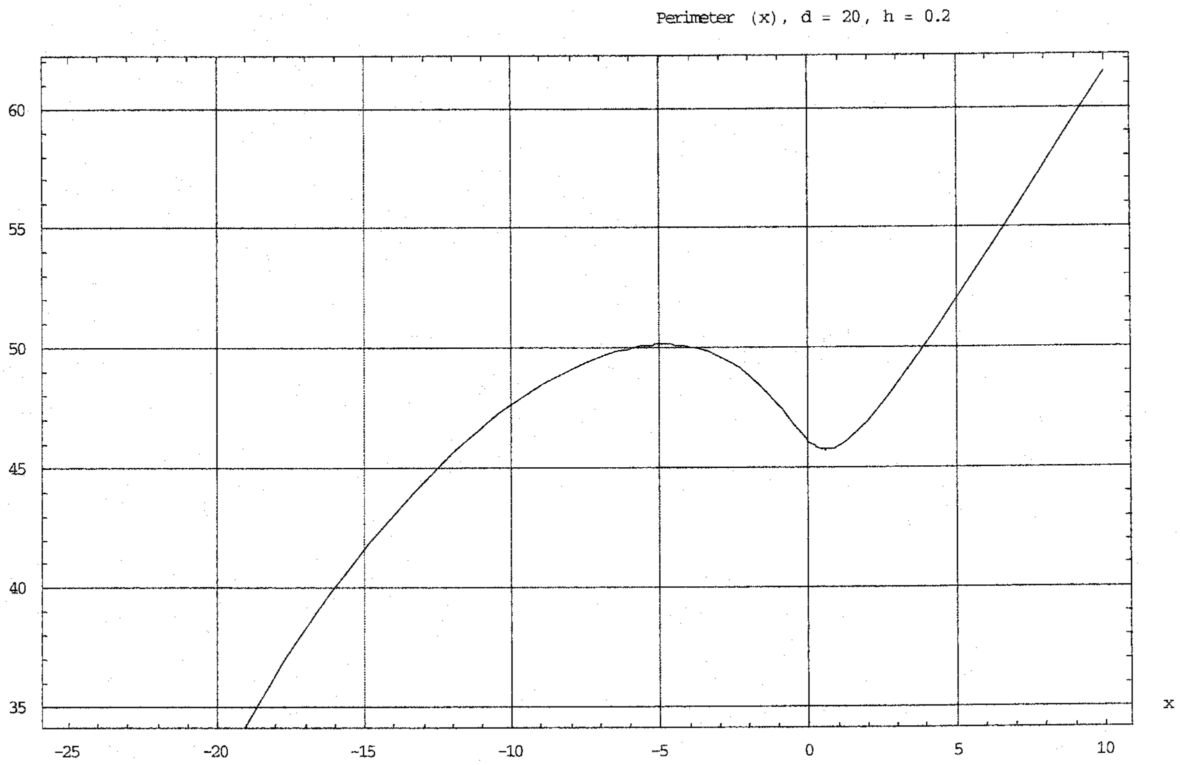
Perimeter(x), $d = 15$, $h = 0.6$



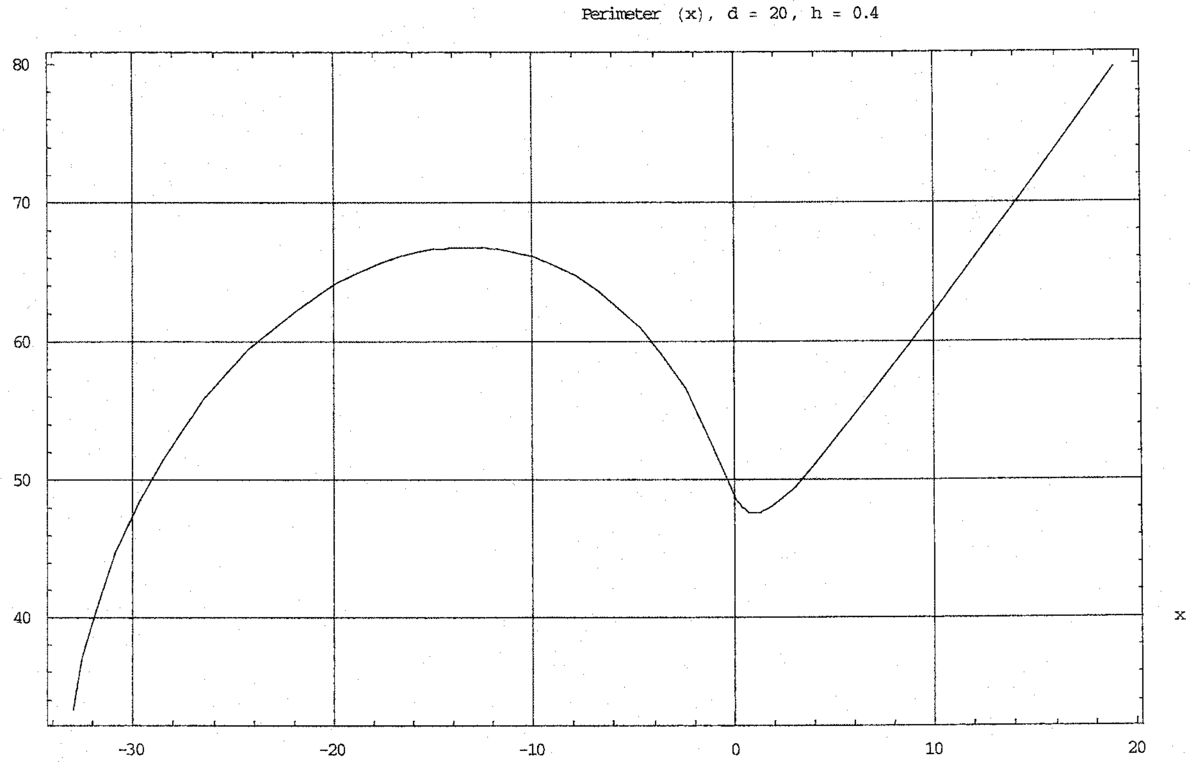
Perimeter(x), $d = 15$, $h = 0.8$



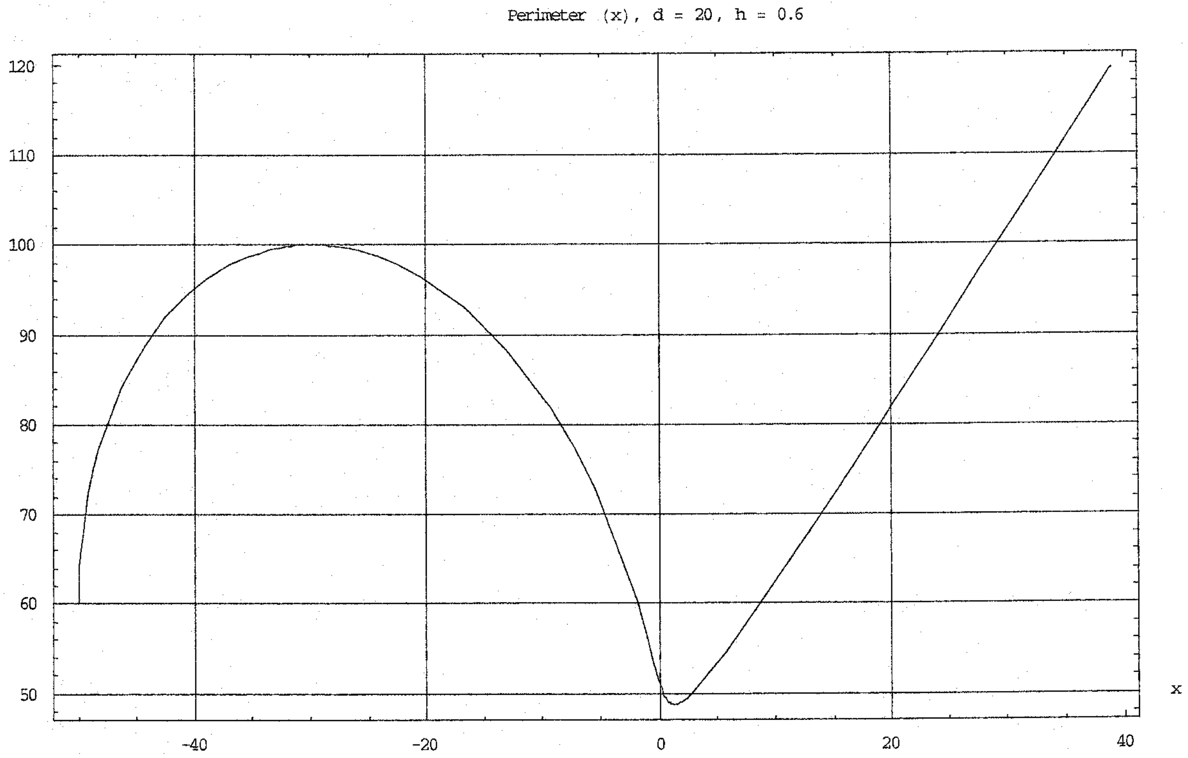
Perimeter(x), $d = 20$, $h = 0.2$



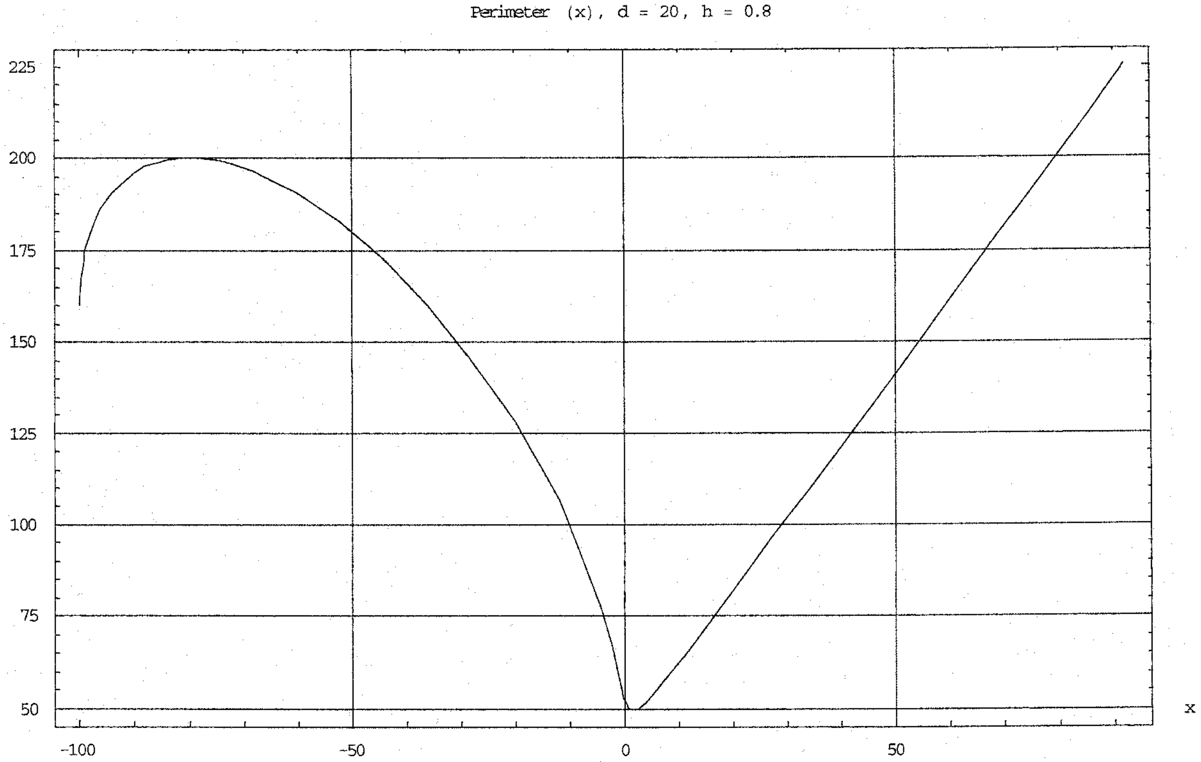
Perimeter(x), $d = 20$, $h = 0.4$



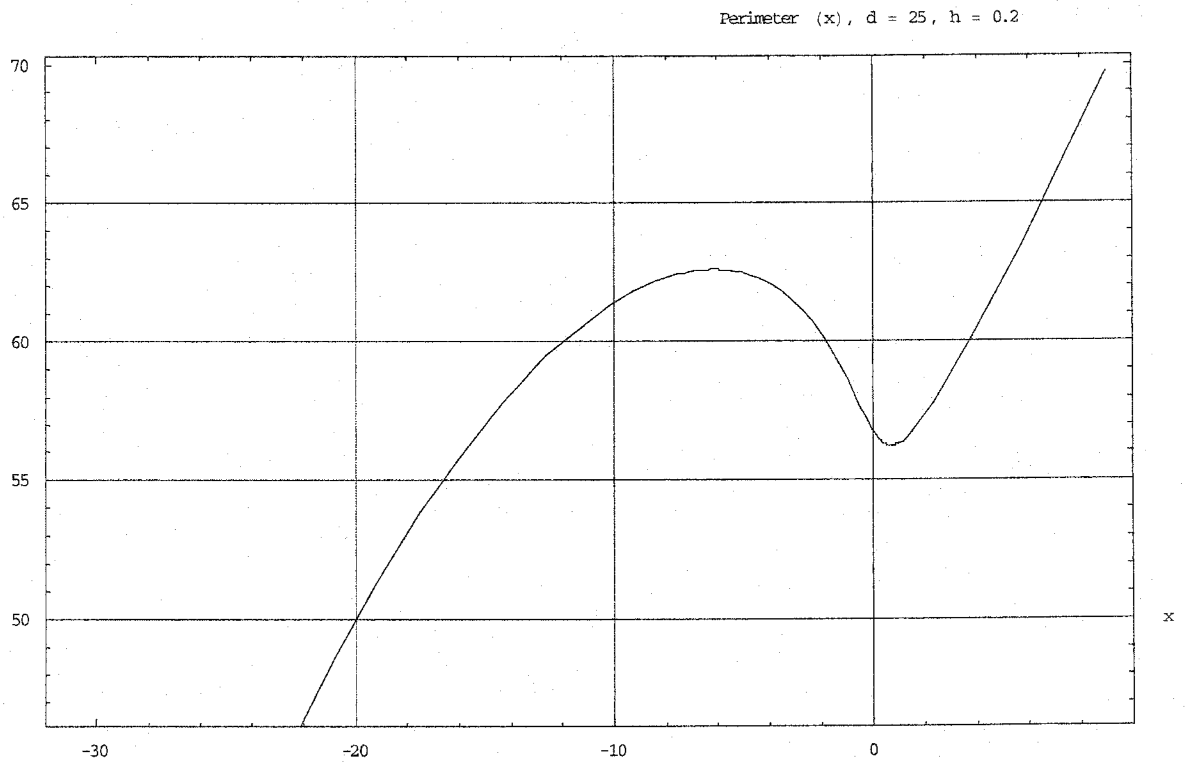
Perimeter(x), $d = 20$, $h = 0.6$



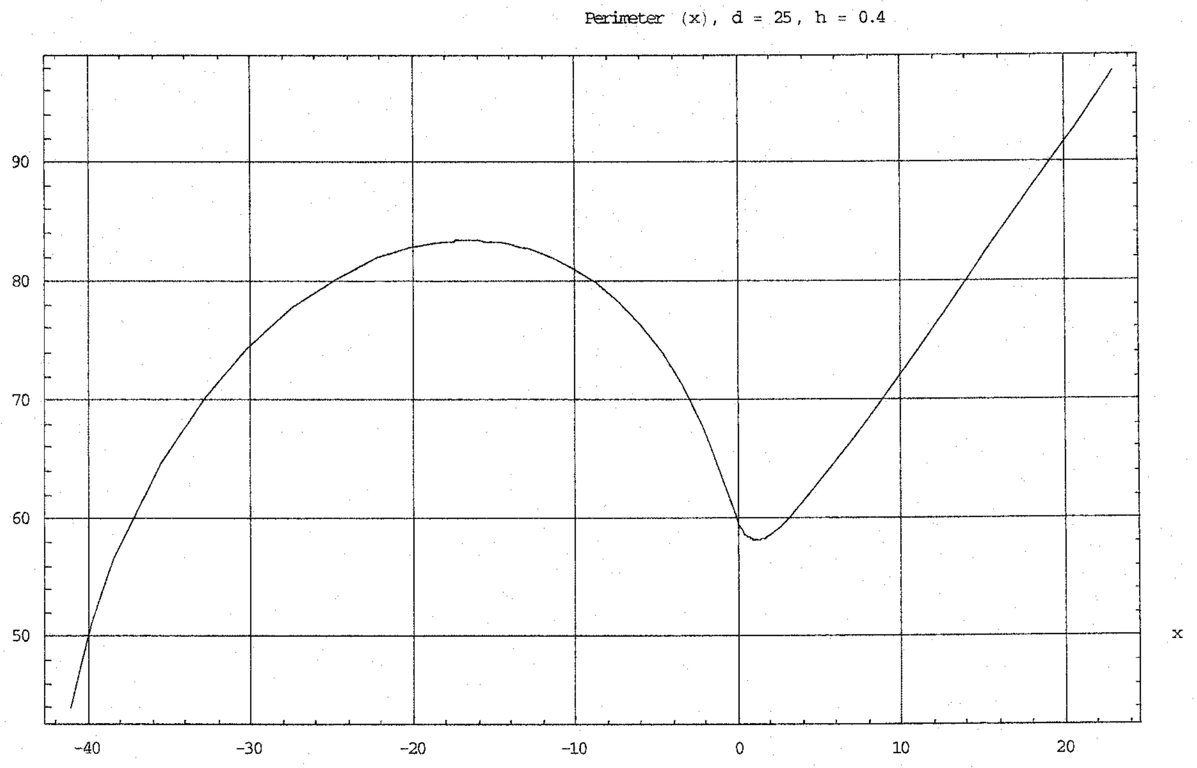
Perimeter(x), $d = 20$, $h = 0.8$



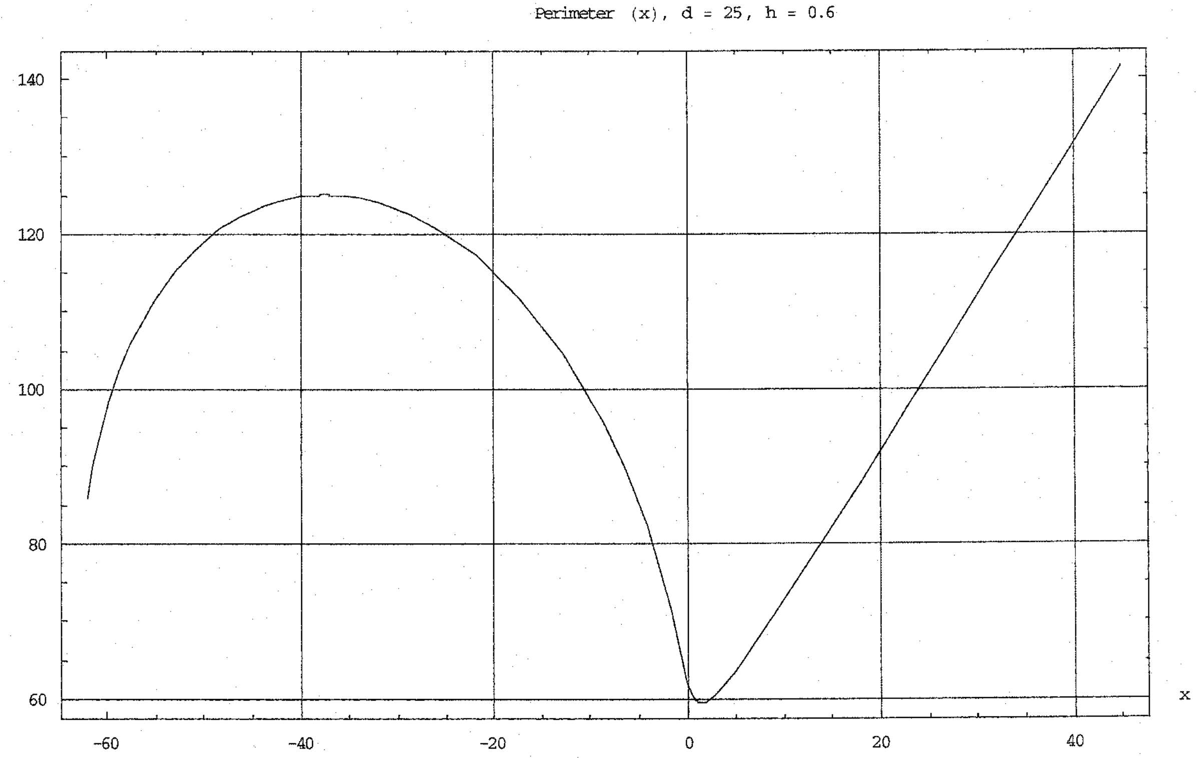
Perimeter(*x*), *d* = 25, *h* = 0.2



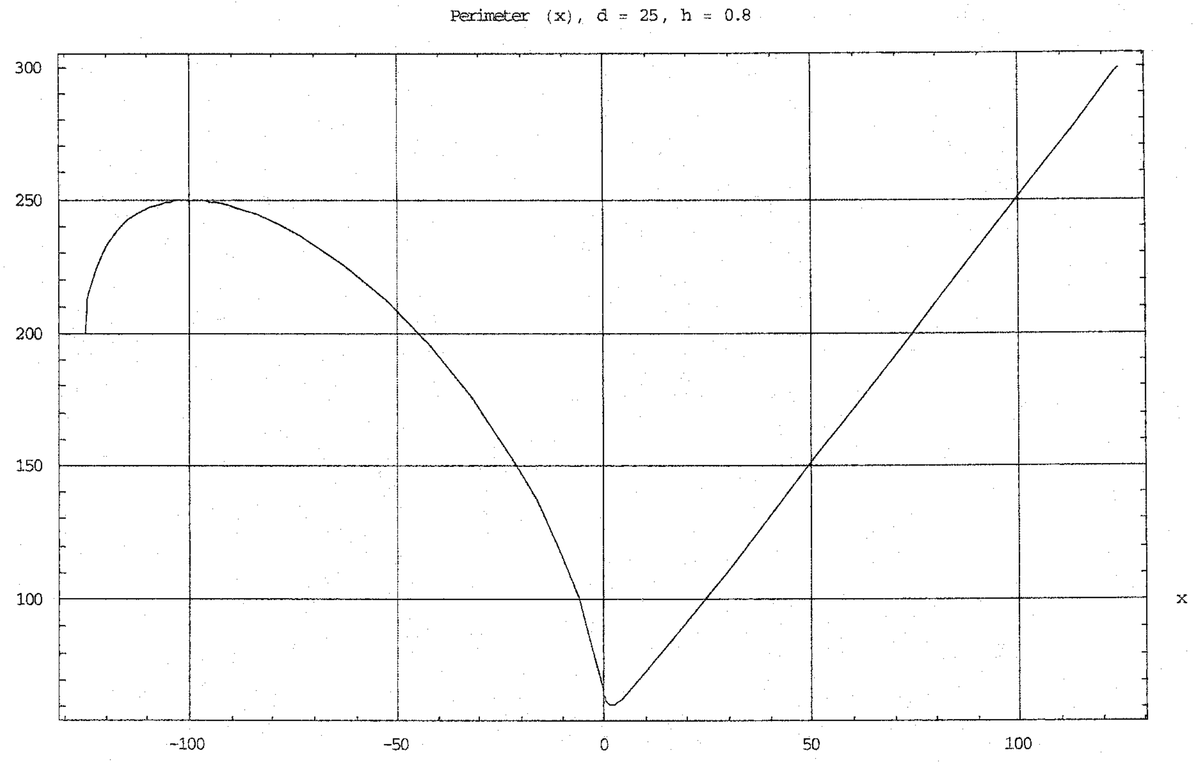
Perimeter(*x*), *d* = 25, *h* = 0.4



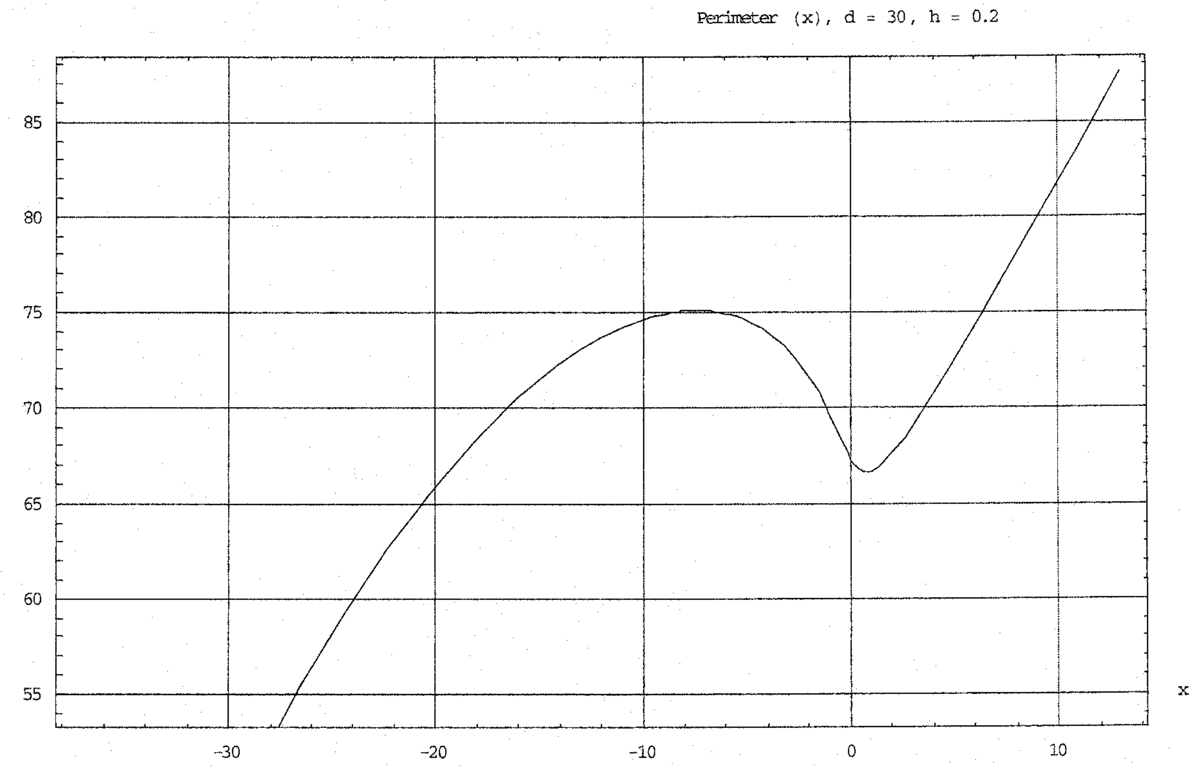
Perimeter(x), $d = 25$, $h = 0.6$



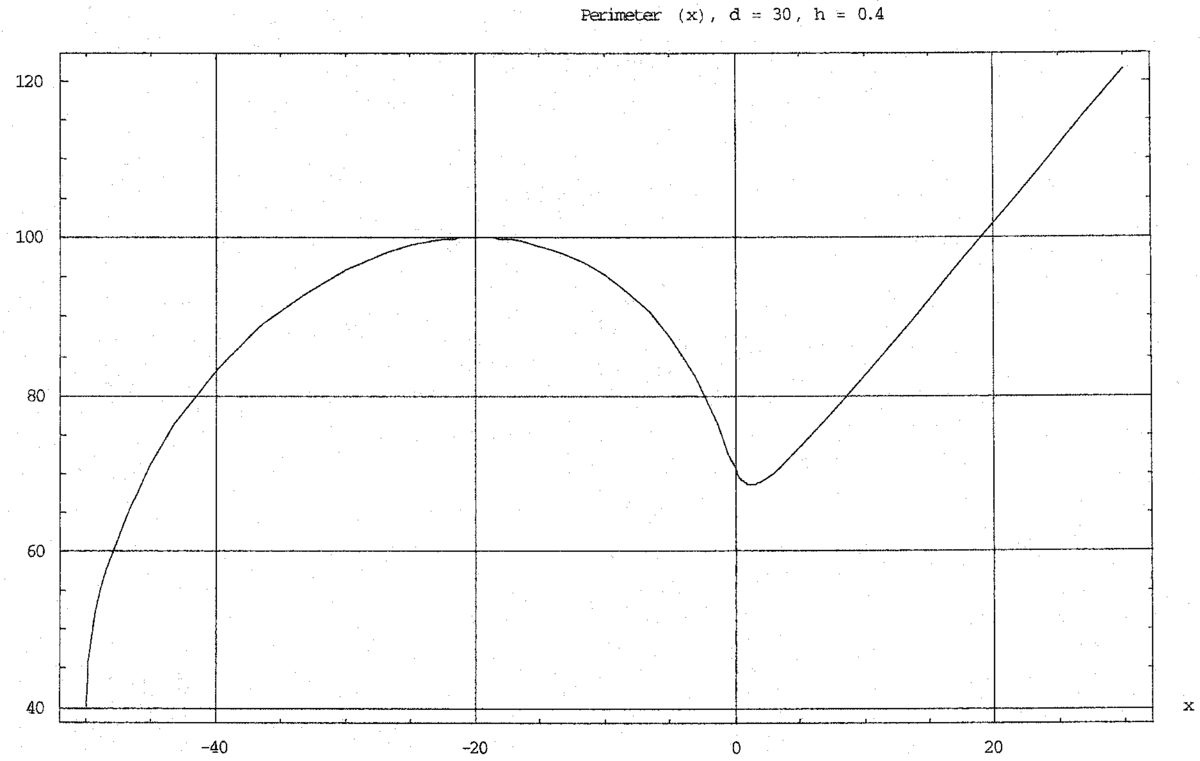
Perimeter(x), $d = 25$, $h = 0.8$



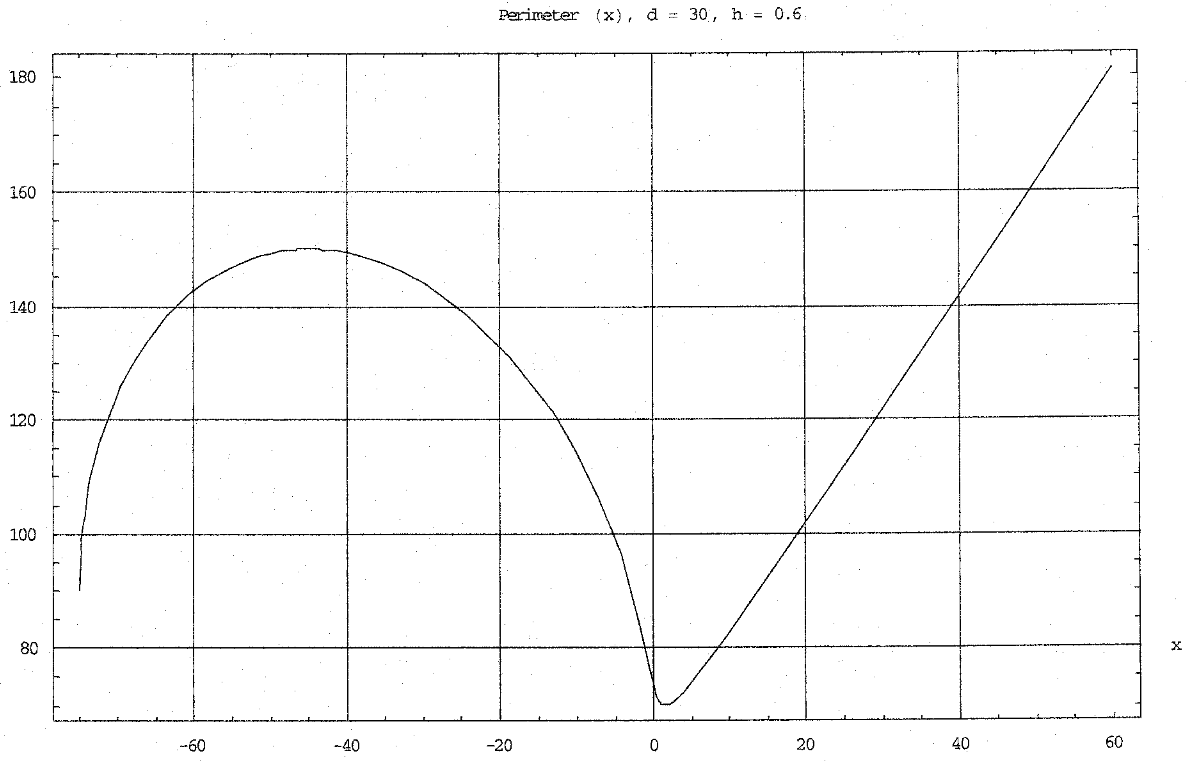
Perimeter(x), $d = 30$, $h = 0.2$



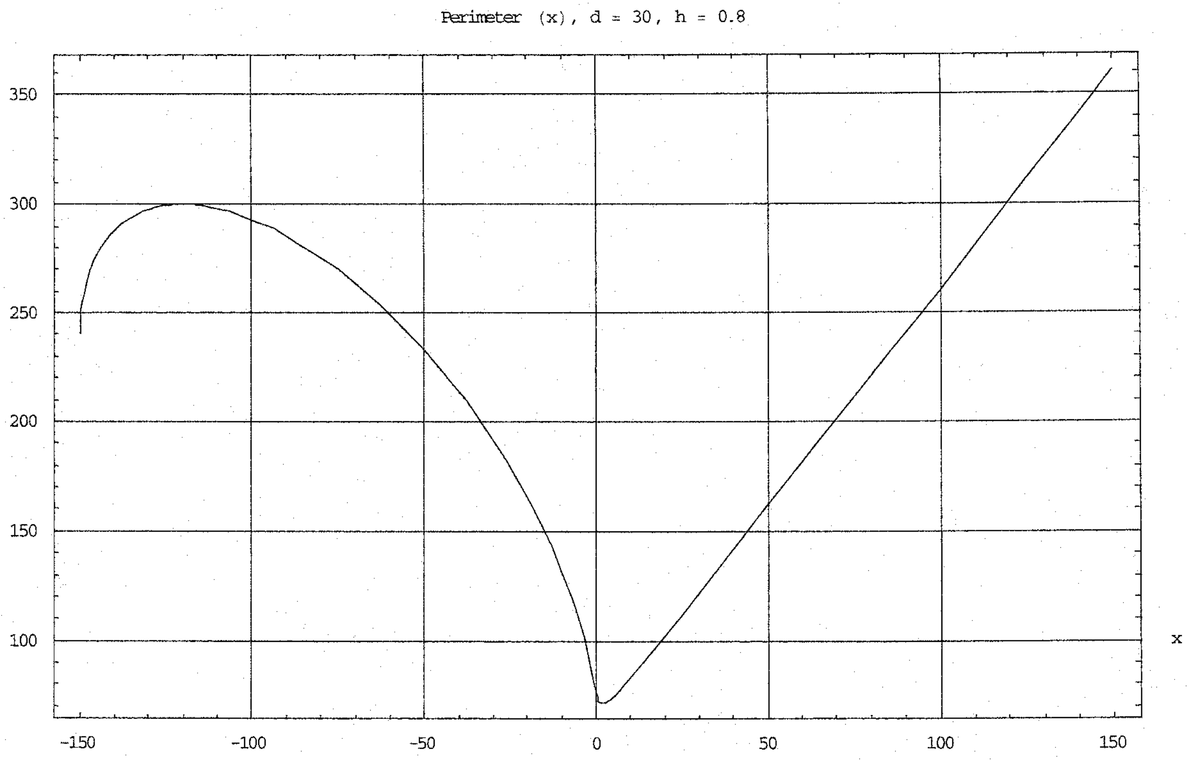
Perimeter(x), $d = 30$, $h = 0.4$



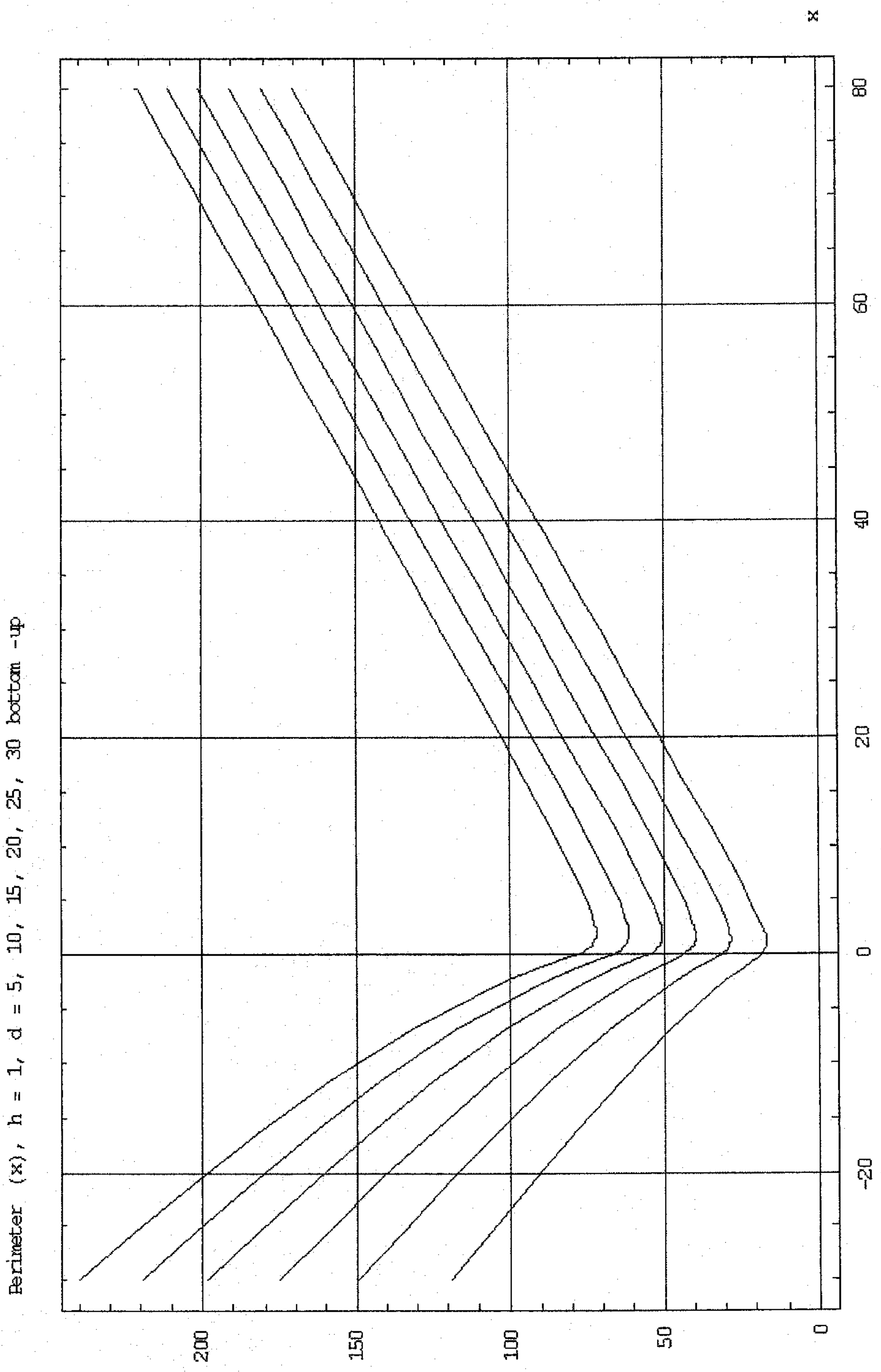
Perimeter(x), $d = 30$, $h = 0.6$



Perimeter(x), $d = 30$, $h = 0.8$



Perimeter(*x*), *h* = 1, *d* = 5, 10, 15, 20, 25, 30 bottom-up



Note that the graphs reflect the mathematical expression for the perimeter function without being restricted by the statement of Problem 2. That is, the degenerate configurations where point *P* is not interior to the triangle's side *AB* are represented as decreasing perimeter beginning at some point beyond the maximum perimeter configuration (for negative *x*, *H* > *h*). These configurations do not provide "better solutions" to the minimum perimeter problem as they violate its statement.

d	h	Perimeter (scaled by H to give a family of triangles)		x		Configuration	
		min	max	min	max	min	max
5	0.1	None	None	None	None	N/A	N/A
5	0.2	None	None	None	None	N/A	N/A
5	0.3	14.061	14.601	0.071	-1.806	Acute	Obtuse 1
5	0.4	14.706	16.879	0.328	-3.154	Acute	Obtuse 1
5	0.5	15.221	20.150	0.496	-4.898	Acute	Obtuse 1
5	0.6	15.652	25.107	0.628	-7.444	Acute	Obtuse 2
5	0.7	16.025	33.406	0.740	-11.638	Acute	Obtuse 2
5	0.8	16.351	50.045	0.839	-19.989	Acute	Obtuse 2
5	0.9	16.641	100.021	0.929	-44.997	Acute	Obtuse 2
5	1	16.899	None	1.013	None	Acute	N/A
10	0.1	None	None	None	None	N/A	N/A
10	0.2	24.378	25.248	0.178	-2.200	Acute	Obtuse 1
10	0.3	25.288	28.724	0.473	-4.134	Acute	Obtuse 1
10	0.4	25.982	33.439	0.662	-6.580	Acute	Obtuse 1
10	0.5	26.552	40.075	0.807	-9.950	Acute	Obtuse 1
10	0.6	27.040	50.053	0.929	-14.972	Acute	Obtuse 2
10	0.7	27.467	66.703	1.036	-23.319	Acute	Obtuse 2
10	0.8	27.846	100.023	1.132	-39.994	Acute	Obtuse 2
10	0.9	28.188	200.011	1.220	-89.999	Acute	Obtuse 2
10	1	28.497	None	1.303	None	Acute	N/A
15	0.1	33.647	33.702	-0.219	-1.060	Obtuse 1	Obtuse 1
15	0.2	35.144	37.662	0.421	-3.561	Acute	Obtuse 1
15	0.3	36.111	42.959	0.678	-6.329	Acute	Obtuse 1
15	0.4	36.861	50.070	0.858	-9.943	Acute	Obtuse 1
15	0.5	37.482	60.050	1.002	-14.967	Acute	Obtuse 1
15	0.6	38.016	75.036	1.124	-22.481	Acute	Obtuse 2
15	0.7	38.486	100.024	1.232	-34.991	Acute	Obtuse 2
15	0.8	38.907	150.015	1.330	-59.996	Acute	Obtuse 2
15	0.9	39.287	300.007	1.420	-134.999	Acute	Obtuse 2
15	1	39.633	None	1.505	None	Acute	N/A

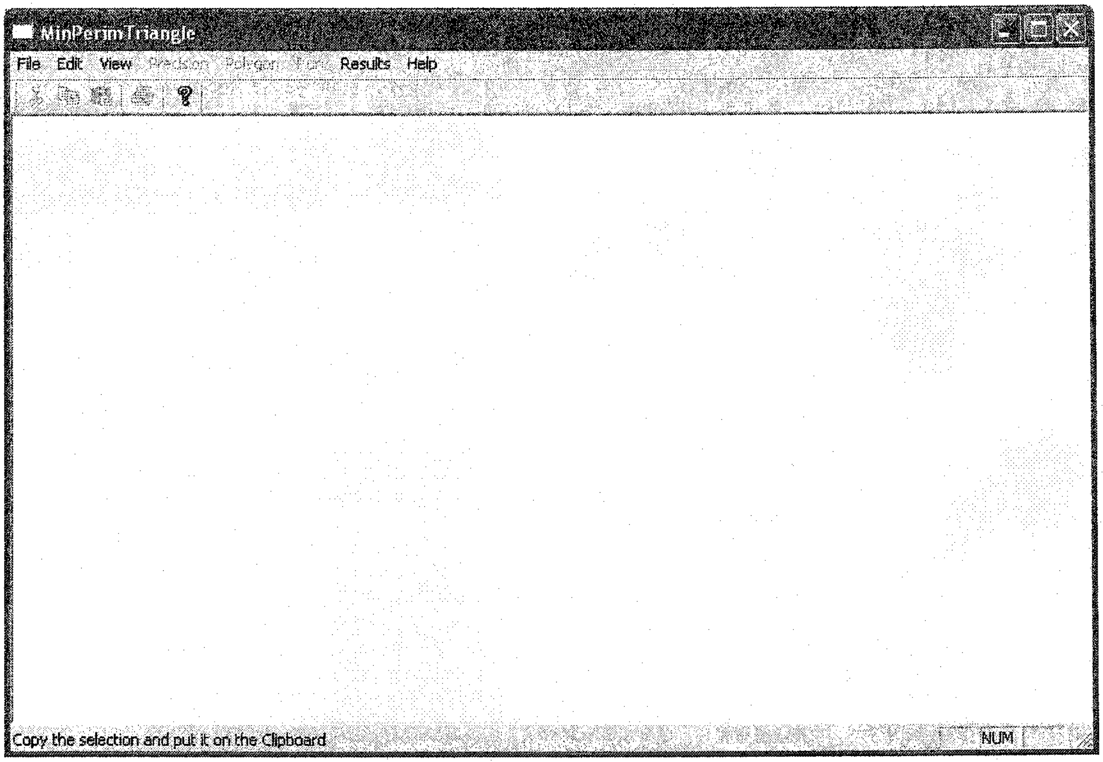
Table A.1 Numerical solutions to the main subsidiary problem summarized

d	*h*	Perimeter (scaled by *H* to give a family of triangles)		*x*		Configuration	
		min	max	min	max	min	max
20	0.1	44.195	44.705	0.094	-1.849	Acute	Obtuse 1
20	0.2	45.733	50.121	0.576	-4.861	Acute	Obtuse 1
20	0.3	46.758	57.219	0.823	-8.497	Acute	Obtuse 1
20	0.4	47.557	66.719	1.003	-13.290	Acute	Obtuse 1
20	0.5	48.222	80.038	1.149	-19.975	Acute	Obtuse 1
20	0.6	48.796	100.027	1.274	-29.986	Acute	Obtuse 2
20	0.7	49.302	133.352	1.385	-46.660	Acute	Obtuse 2
20	0.8	49.756	200.011	1.485	-79.997	Acute	Obtuse 2
20	0.9	50.168	400.005	1.578	-179.999	Acute	Obtuse 2
20	1	50.544	None	1.665	None	Acute	N/A
25	0.1	54.626	55.760	0.254	-2.496	Acute	Obtuse 1
25	0.2	56.223	62.596	0.694	-6.140	Acute	Obtuse 1
25	0.3	57.300	71.489	0.939	-10.655	Acute	Obtuse 1
25	0.4	58.143	83.375	1.121	-16.632	Acute	Obtuse 1
25	0.5	58.847	100.030	1.270	-24.980	Acute	Obtuse 1
25	0.6	59.455	125.021	1.398	-37.489	Acute	Obtuse 2
25	0.7	59.993	166.681	1.512	-58.328	Acute	Obtuse 2
25	0.8	60.476	250.009	1.615	-99.998	Acute	Obtuse 2
25	0.9	60.915	500.004	1.711	-224.999	Acute	Obtuse 2
25	1	61.317	None	1.800	None	Acute	N/A
30	0.1	64.990	66.835	0.368	-3.105	Acute	Obtuse 1
30	0.2	66.646	75.080	0.790	-7.408	Acute	Obtuse 1
30	0.3	67.772	85.765	1.037	-12.808	Acute	Obtuse 1
30	0.4	68.655	100.035	1.222	-19.971	Acute	Obtuse 1
30	0.5	69.393	120.025	1.374	-29.983	Acute	Obtuse 1
30	0.6	70.032	150.018	1.505	-44.991	Acute	Obtuse 2
30	0.7	70.598	200.012	1.622	-69.995	Acute	Obtuse 2
30	0.8	71.107	300.007	1.728	-119.998	Acute	Obtuse 2
30	0.9	71.570	600.004	1.826	-270.000	Acute	Obtuse 2
30	1	71.995	None	1.917	None	Acute	N/A

Table A.1 Numerical solutions to the main subsidiary problem summarized (Continued)

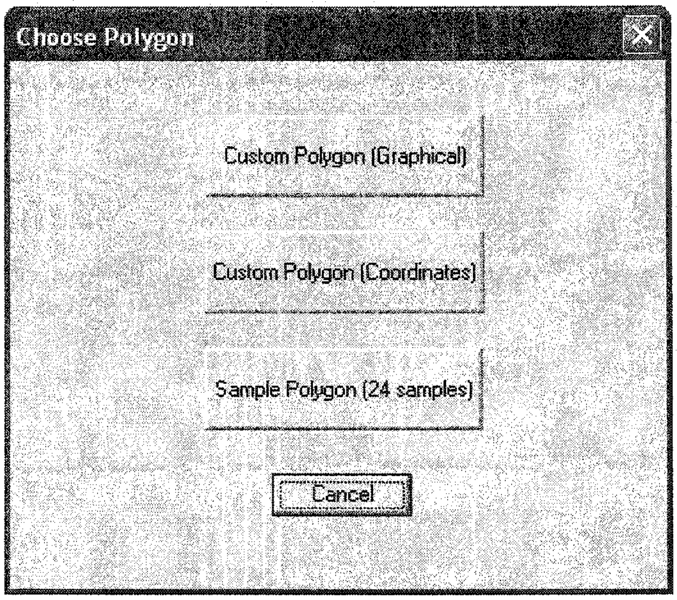
APPENDIX C: IMPLEMENTATION INTERFACE DEVELOPED USING MFC (MICROSOFT FOUNDATION CLASSES) – SCREENSHOTS

1. Software system's main window

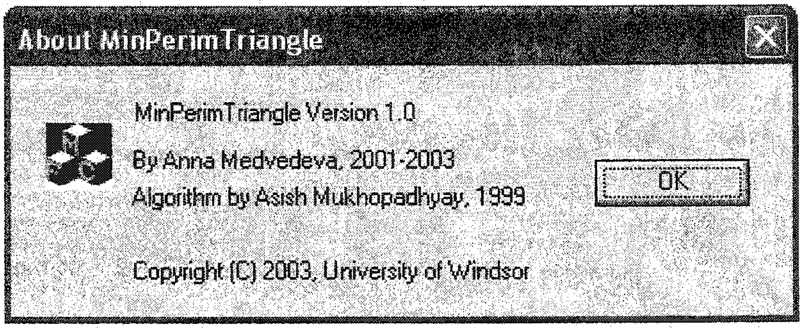


Copy the selection and put it on the Clipboard

2. "Choosing a polygon" window


Choose Polygon
Custom Polygon (Graphical)
Custom Polygon (Coordinates)
Sample Polygon (24 samples)
Cancel

3. “About” window



About MinPerimTriangle

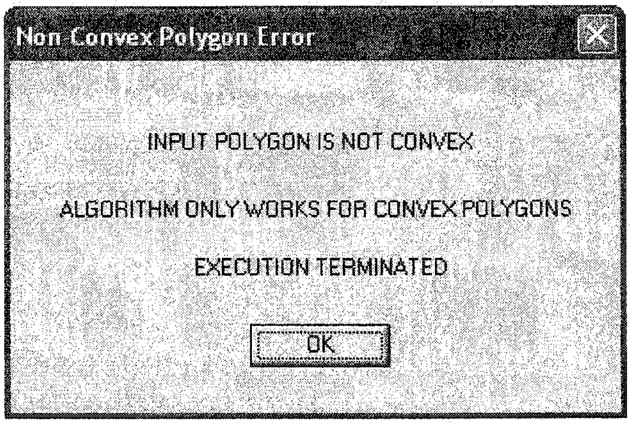
MinPerimTriangle Version 1.0

By Anna Medvedeva, 2001-2003

Algorithm by Asish Mukhopadhyay, 1999

Copyright (C) 2003, University of Windsor

4. “Non-convex polygon error” window



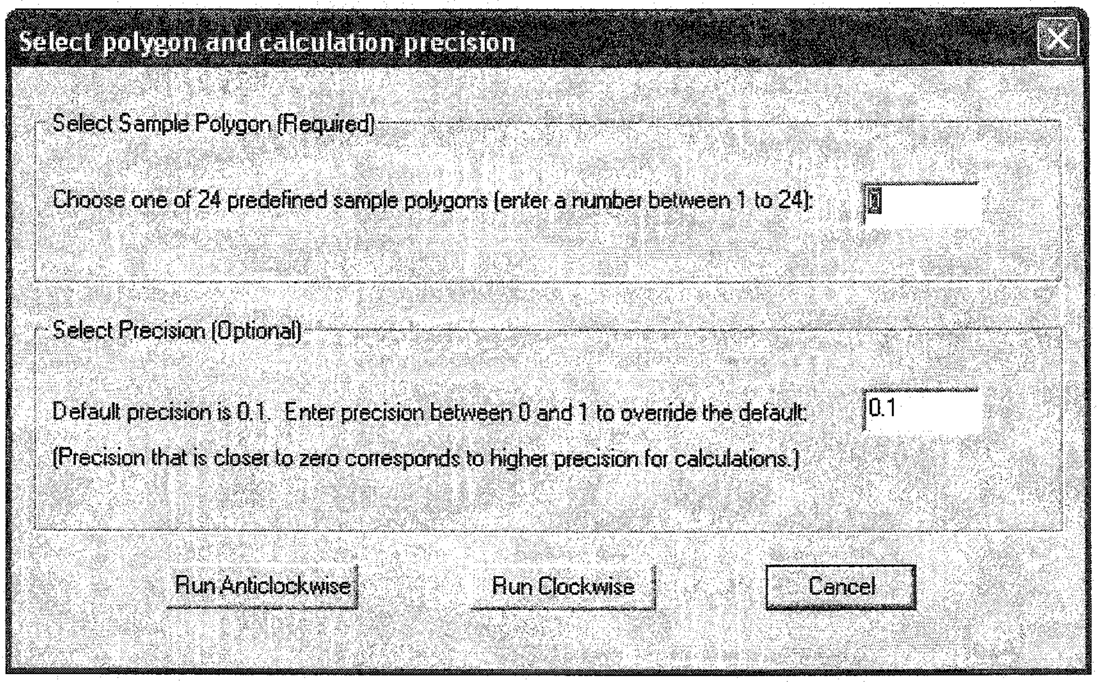
Non Convex Polygon Error

INPUT POLYGON IS NOT CONVEX

ALGORITHM ONLY WORKS FOR CONVEX POLYGONS

EXECUTION TERMINATED

5. “Selecting a sample polygon” window



Select polygon and calculation precision

Select Sample Polygon (Required)

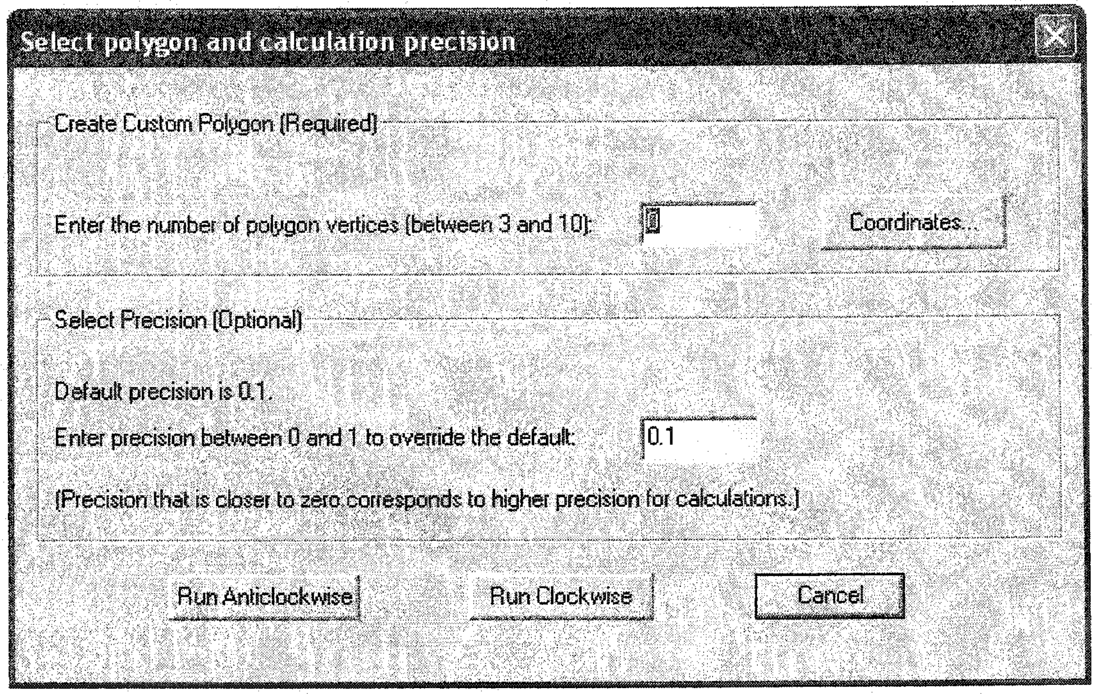
Choose one of 24 predefined sample polygons (enter a number between 1 to 24):

Select Precision (Optional)

Default precision is 0.1. Enter precision between 0 and 1 to override the default: 0.1

(Precision that is closer to zero corresponds to higher precision for calculations.)

6. “Creating a custom polygon” window



Create Custom Polygon (Required)

Enter the number of polygon vertices (between 3 and 10): 0

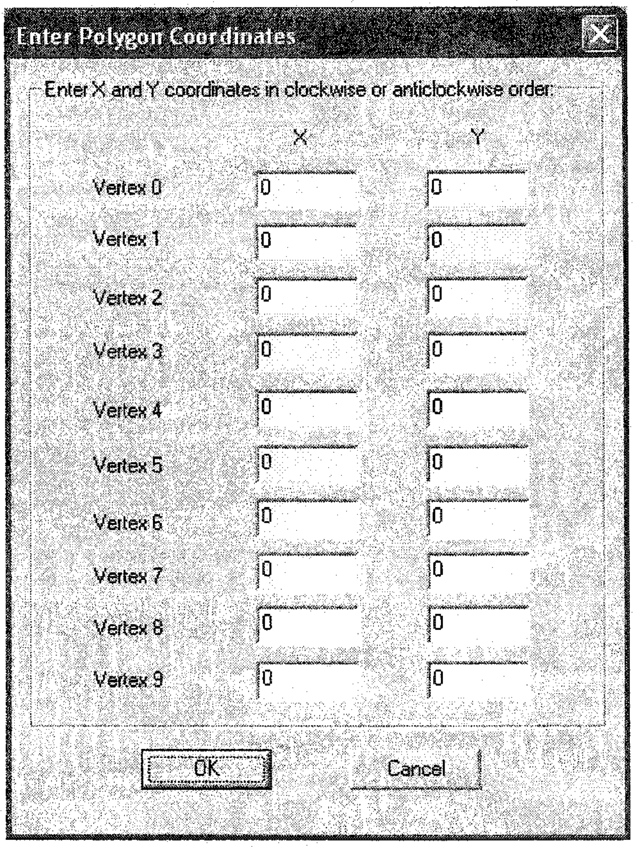
Select Precision (Optional)

Default precision is 0.1.

Enter precision between 0 and 1 to override the default: 0.1

(Precision that is closer to zero corresponds to higher precision for calculations.)

7. “Entering polygon coordinates” window



Enter X and Y coordinates in clockwise or anticlockwise order:

	X	Y
Vertex 0	0	0
Vertex 1	0	0
Vertex 2	0	0
Vertex 3	0	0
Vertex 4	0	0
Vertex 5	0	0
Vertex 6	0	0
Vertex 7	0	0
Vertex 8	0	0
Vertex 9	0	0

VITA AUCTORIS

Anna Valentinovna Medvedeva was born in Kyiv, Ukraine on December 16, 1972. In 1989, she graduated from a high school in Kyiv, Ukraine that provided specialization in Computer and Information Science and Cybernetics. In 1995, Anna received a Bachelor of Science in Radiophysics and Electronics from Kyiv University, Kyiv, Ukraine. Anna and her family immigrated to the United States in 1996. In 1999, she received a Bachelor of Science in Genetics and Cell Biology from Washington State University, Pullman, Washington, U.S.A. Anna is in the process of completing her Master's degree in Computer Science at the University of Windsor, Ontario, Canada. Her plans include pursuing a Ph. D. in Computer Science (Bioinformatics) and working in industry. Anna's future research interests include applications of Computing Science to problems that arise in Medicine and Biology. She is also planning to become an Oracle DBA Certified Professional (OCP).