MINIMUM VERTEX HULLS FOR POLYHEDRAL DOMAINS [1]
(Conference abstract)

GAUTAM DAS - University of Wisconsin

DEBORAH JOSEPH - University of Wisconsin

1. INTRODUCTION

In this paper we investigate several variations of the following problem:

> Given a collection of pairwise disjoint polygons and their spatial positions in the plane, cover each with a polygonal hull such that
> (i) the hulls are pairwise disjoint, and
> (ii) the total number of vertices of the hulls is minimized.

This problem can be readily extended to three dimensions, where polyhedrons replace polygons, or it can be restricted to special types of polygons such as rectilinear polygons or convex polygons. If the objects are sufficiently far apart, each object can be enclosed in a triangle (or tetrahedron); however, if the objects are placed closer together, then because we require that the hulls are disjoint the problem is nontrivial.

Our problem belongs to the important area of object approximations, where the goal is to approximate complex objects by simpler shapes [A, AB, CY, KL]. The situation we consider typically arises in circuit design where rectilinear polygons represent circuit components, or in robot motion planning where polyhedrons represent obstacles inside a workspace. Our goal of obtaining minimum vertex hulls, if efficiently accomplished, may speed up subsequent processing because of the reduced input size. For example, an algorithm for constructing a communication network between the objects based on the *link metric* may be sped up because this preprocessing straightens out "bends" in the objects.

Our results are the following.

Result 1: The two dimensional minimum vertex hulls problem is NP-hard. The proof follows from a reduction from the Planar-3SAT problem, [L]. The problem is NP-hard even when restricted to convex polygons, or rectilinear convex polygons.

Result 2: For the restricted case of rectilinear polygons there is an efficient approximation algorithm which constructs hulls that achieve more than half the maximum possible reduction in vertices. If n is the input size and k is the number of polygons, this algorithm runs in $O(nlogk + nloglogn)$, and uses the algorithm in [TV] as a subroutine. As an application this algorithm, along with ideas from [CKV], led to an algorithm for computing shortest rectilinear paths amidst rectilinear polygonal obstacles in $O(k(logk)^2 + nlogk + nloglogn)$ time. Other applications of the algorithm

[1] This work was supported in part by the National Science Foundation under NSF PYI grant DCR-8402375. Authors' address: Computer Sciences Department, University of Wisconsin, 1210 West Dayton St., Madison, WI 53706, USA.

include algorithms for computing rectilinear minimum spanning trees and rectilinear *Steiner trees* [GJ] between the polygons.

Result 3: There is an algorithm for constructing rectilinear minimum vertex hulls in two dimensions which is based on dynamic programming, whose running time is exponential in the *number of polygons* rather than the input size. Thus, the difficulty of the problem is related more to the number of polygons than to their shapes and positions. As we shall note later, this is *not true* for the three dimensional case. This algorithm may be useful in practice, when the number of objects is small.

Result 4: In three dimensions the minimum vertex hull problem is also NP-hard. Again, this result holds for convex polyhedrons and for rectilinear convex polyhedrons. In addition, the following more interesting result holds for three dimensions. The problem is NP-hard *even for only two nonconvex polyhedrons.* This is also true for rectilinear polyhedrons. Thus, unlike the two dimensional case, here the difficulty lies in the complex shapes that three dimensional objects can have. The assumption we make here is that the polyhedrons are of *genus* 0; that is they are topologically homeomorphic to the sphere.

Result 5: Klee [O] posed the following related problem: given two concentric convex polytopes in arbitrary dimensions, fit a minimum vertex convex polytope between them. A generalization is to allow nonconvexity in the problem. While the two dimensional problems have been solved, [AB], the higher dimensional versions remain open. As an important corollary of Result 4, we show that Klee's problem generalized for nonconvex polyhedrons is NP-hard. (Again, the assumption is that the polyhedrons are of genus 0, and the result is seen to hold even for rectilinear polyhedrons).

Our research differs from most previous research on object approximations in two important ways. First, the measures of simplicity previously studied have usually been continuous (for instance, *area* [KL], or *symmetric differences* [A]), while ours is combinatorial, as in [AB]. Second, complex objects have usually been considered in isolation, while we have a more restrictive environment where a number of neighboring objects can hinder the simplification process.

Section 2 defines some terms and notations. Section 3 provides some details of the NP-hardness proof of our two dimensional problem (part of Result 1). Section 4 describes the approximation algorithm for obtaining rectilinear hulls (part of Result 2). Section 5 describes the exact algorithm for the special case of rectilinear polygons (Result 3). Section 6 provides some details of the NP-hardness proof of our three dimensional problem (part of Result 4). Due to space limitations we omit details of the other results, which may be found in [DJ]. We conclude with a list of open problems.

2. DEFINITIONS AND NOTATIONS

A *simple polygon* is a piecewise linear, nonintersecting closed curve on the plane. A *rectilinear polygon* is one where each piece (or *edge*) is parallel to either the x or the y axis. Adjacent edges meet at *vertices.* A *convex polygon* is one where the line segment joining any pair of interior points lies wholly inside the polygon. A *rectilinear convex* polygon is one where the above is true for all pairs of interior points that have the same x co-ordinate or the same y co-ordinate. A *polygon set* is a collection of pairwise disjoint polygons on the plane. Given a polygon set, a *hull set* is another collection of pairwise disjoint polygons (or *hulls*), such that each hull encloses exactly one polygon from the polygon set. If P is a polygon set, $\#(P)$ denotes the number of vertices in P.

In three dimensions the definitions are similar. A *simple polyhedron* is a closed region in space with a piecewise planar surface. Each planar piece is known as a *facet*. Facets are adjacent at *edges*, and edges are adjacent at *vertices*. A polyhedron is of *genus* g if it is topologically homeomorphic to a sphere with g handles. Throughout we restrict the polyhedrons to be of genus 0. One reason for having this assumption is because the graphs defined by the vertices and edges of genus 0 polyhedrons are planar. So, the overall size of the input description is linearly related to the number of vertices, and thus reducing vertices reduces the input. The remaining definitions are similar to their two dimensional counterparts.

The next section describes some details of the two dimensional NP-hardness proof.

3. NP-HARDNESS IN TWO DIMENSIONS

In this section we outline a proof which shows that constructing minimum vertex hulls for a convex polygonal set (with integer co-ordinates) is NP-hard.

We shall prove NP-completeness for the decision version of our problem, which is: Given a convex polygon set P, with all vertices having integer co-ordinates, and an integer K, is there a set of hulls H, such that $\#(P) - \#(H) \geq K$? Clearly, the problem is in NP because a nondeterministically constructed H can be verified in polynomial time. For proving NP-completeness, the reduction will be from Planar-3SAT, [L], which is defined as follows. A *variable-clause* graph of a 3SAT instance (with n variables and m clauses) is a graph where (i) vertices are variables and clauses, and (ii) edges are between variable vertices and clause vertices, with the rule that if a clause C has a literal of a variable V, then $[C, V]$ is an edge. Also, an edge is marked "+" or "−", depending on which literal is in the clause. Planar-3SAT may now be formally stated as: Given a planar variable-clause graph for a 3SAT formula, is there a satisfying truth assignment?

In our reduction, we design components for variables, clauses, and edges of a given planar variable-clause graph. Each component will be a collection of convex polygons. All components will be "superimposed" on the planar graph, so that we have a global collection of convex polygons. The planarity of the drawing will ensure that these polygons remain pairwise disjoint. In this outline, we omit details on how to lay out the polygons on integer co-ordinates. The following definitions will be useful in describing the remaining steps.

Consider the two groups of polygons in Figure 1. The central quadrilateral of each group defines an empty triangular region called a *block*, which would be filled if the quadrilateral was extended into a triangle. Thus a minimum vertex hull for the quadrilateral either *selects* the block, or excludes it. Two blocks belonging to different groups can intersect as the figure shows. If the minimum vertex hulls of one group selects its block, clearly the hulls of the other group will have to exclude the intersecting block. Throughout our construction we only place such groups on the plane so that appropriate blocks intersect. To construct minimum vertex hulls it is to our advantage to select as many such blocks as possible.

Variable Component: There is one variable component per variable, and the design of each component is dependent upon the total number of clauses, m. In particular, each component consists of $2m$ polygon groups (as defined above), with the blocks intersecting in a cycle, as in Figure 2. Let one set of m alternating blocks in the cycle be called *positive* blocks, while the other set be called *negative* blocks. Clearly, a set of minimum vertex hulls for the component will select either the positive blocks only, or the negative blocks only. In the former case, the variable is set *FALSE*, and in the latter case it is set *TRUE*. Finally, we superimpose each variable component as a "macro vertex" in the variable-clause graph.

Clause Component: Since a clause has three literals, its component will consist of three polygon groups whose blocks intersect in a cycle, in a manner similar to above. Thus, a set of minimum vertex hulls for the component can select only one of the blocks. As before, we superimpose each clause component as a macro vertex in the variable-clause graph.

Edge Component: Edge components provide a means of linking up the configuration. We shall illustrate this by an example. Consider a "–" edge of the variable clause graph. The component is realized in Figure 3, and should altogether have an even number of blocks intersecting in a chain. We superimpose the chain on the edge in the graph, such that one end block intersects a negative block in the variable component, and the other intersects a literal block in the clause component. (The actual number of blocks in the chain can be shown to be polynomial). Since there are enough blocks in each variable component to acommodate all clauses, all edge components may be laid out without encountering polygon crossovers. Since an edge component has an even number of blocks, its best hull set has to select at least one of the end blocks. In that case, the block that intersects it (which may belong to either the variable or the clause component) will not be selected.

Clearly, the reduction requires only polynomial time. Let P be the set of polygons of all components and let the total number of blocks of all edge components be $2r$. Let $K = nm + m + r$. Note that the first term corresponds to the reduction in vertices if all variable components had their best hulls. Similarly, the second and third terms correspond to clause and edge components respectively. We pose the problem: Is there a set of hulls H such that $\#(P) - \#(H) \geq K$? From the reduction, we can conclude that this is true if and only if the instance of Planar-3SAT is satisfiable. Thus constructing minimum vertex hulls for a set of convex polygons is NP-hard.

The proof for the case of rectilinear convex polygons is more involved, and the reader may find it in [DJ]. The next section describes an approximation algorithm for the rectilinear problem.

4. AN APPROXIMATION ALGORITHM FOR RECTILINEAR HULLS

In this section we shall only be concerned with rectilinear polygons. We describe a polynomial time algorithm which constructs valid hulls that closely approximate minimum vertex hulls, in a sense described below.

Let P be a rectilinear polygon set. It is not hard to see that there exists a set of minimum vertex hulls which is *tight*, that is, each hull edge touches some polygon edge. This is easily proven, because any set of minimum vertex hulls can be tightened by pushing each hull edge towards the enclosed polygon in a direction perpendicular to the edge, until it hits the polygon. We may lengthen / shorten adjacent hull edges, but no new vertices will be added. For notational convenience, henceforth $MinHull(P)$ will denote a set of tight minimum vertex hulls of P. The algorithm outputs a hull set H such that $\#(P) - \#(H) \geq \frac{1}{2}[\#(P) - \#(MinHull(P))]$. The output H may not be tight; the concept of tightness is primarily used in analyzing the algorithm.

The algorithm is essentially greedy in nature; it iteratively performs local modifications to the shapes of the polygons, such that each iteration reduces the number of vertices. There are two such local operations, and both are the major iterative steps of the algorithm.

Fill-Well(P): Consider any polygon of P. Suppose its boundary has a shape similar to Figure 4, or symmetric to it. The polygonal region enclosed by its boundary between a and b, and the straight line $[a, b]$, is called a *well.* Note that a and b need not be vertices. $Fill - Well$ identifies a well that does not intersect with any polygon, and *fills* it by replacing the segment of the polygon's boundary between a and b by the straight line $[a, b]$. Clearly if a well does get filled, then $\#(Fill - Well(P)) \leq \#(P)$.

Fill-Corner(P): Consider any polygon of P. Suppose its boundary has a shape similar to Figure 5, or symmetric to it. The polygonal region enclosed by its boundary between a and b, and the straight lines $[a, d]$ and $[b, d]$, is called a *corner.* Unlike wells, note that a and b are vertices. $Fill - Corner$ identifies a corner that does not intersect with any polygon, and fills it, by replacing the segment of the polygon's boundary between a and b by the two lines $[a, d]$ and $[b, d]$. Clearly if a corner does get filled, then $\#(Fill - Corner(P)) < \#(P)$.

It is easy to see that a set of minimum vertex hulls can be constructed by applying a particular sequence of these operations. Unfortunately, it is unlikely that the exact sequence can be determined in polynomial time. However, we now outline our polynomial time algorithm which applies these operations in a sequence that guarantees the error bound claimed above in the approximate solution. The algorithm has two major steps. In the first step wells are filled, and in the second step corners are filled.

Step 1: In this step, $Fill - Well$ operations are applied exhaustively on P, converting it to P_1. At any stage, the selection of the well to be filled is arbitrary. Thus P_1 should have no further wells that can be filled. It is not hard to see that $Fill - Well$ operations are "harmless", that is $\#(MinHull(P)) = \#(MinHull(P_1))$.

Step 2: In this step, $Fill - Corner$ operations are applied exhaustively on P_1, converting it to H, the final output. It is easy to see that H should have neither wells nor corners that can be filled. Unlike Step 1 however, at any stage the selection of the corner to be filled is *not* arbitrary. The selection procedure is the key idea of the algorithm, and we describe it after the following definitions and facts.

After Step 1, the boundary of each resulting polygon is a cycle of alternating fragments that are exposed to the exterior (called *open fragments*), and fragments that are adjacent to a neighboring polygon (called *closed fragments*), as in Figure 6. It is easy to see that *all* closed fragments belong to any final set of minimum vertex hulls. Thus the algorithm has to only examine and modify open fragments. The open fragments can be further divided into *staircases.* Observe that the open edge between any two staircases also belongs to any final set of tight minimum vertex hulls. Staircases themselves are sequences of *steps*. Suppose a polygon has a corner that can be filled. Then the common boundary between the corner and the polygon has to be a sequence of adjacent steps of some staircase. Define the *size* of a staircase as the number of steps it contains. Also, a step at either end of a staircase is called a *head*.

The selection procedure in Step 2 can now be described. A head of any staircase is examined to determine if the corner that it defines can be filled. Two cases arise:

(1) *The corner cannot be filled:* Then the head is clearly a part of a final hull, and is thus removed from the staircase. (Figure 7).

(2) *The corner can be filled:* Then the corner is filled, and the updates made to the staircase are indicated in Figure 8.

We observe that in either case, the size of a staircase is decreased, thus the iteration of the selection procedure will terminate. The result of the iteration, H, is the final output. Clearly the whole process is polynomial. The following lemma proves that H indeed satisfies the claimed error bound.

Lemma: $\#(P) - \#(H) \geq \frac{1}{2}[\#(P) - \#(MinHull(P))]$.

Proof: From Step 1, we know that,

(1) $\#(P) \geq \#(P_1)$, and

(2) $\#(MinHull(P)) = \#(MinHull(P_1))$.

Since H does not have any wells and corners to be filled,

(3) $\#(H) = \#(MinHull(H))$.

Now let the total number of $Fill - Corner$ operations be c. Let the sequence of polygon sets produced during Step 2 be $P_1 = R^0, R^1, R^2, .. , R^c = H$. Consider $MinHull(R^i)$ of a set of polygons R^i. Since it is tight, in the worst case it is conceivable that it intersects the next corner to be filled as shown in Figure 9. However, we can construct a valid new hull set for R^{i+1} which has at most 2 more vertices than $MinHull(R^i)$, by "bending" the intersecting portions of $MinHull(R^i)$ outwards to avoid the corner, as shown in Figure 9. But after filling the corner, the new set of polygons R^{i+1} has at least 2 vertices less than R^i. We thus get two more conditions,

(4) $\#(MinHull(H)) \leq \#(MinHull(P_1)) + 2c$.

(5) $\#(H) \leq \#(P_1) - 2c$.

In the 5 conditions above, there are 7 quantities, 3 of which appear in the statement of the lemma. If we eliminate the other 4 quantities, the lemma follows. ∎

The algorithm can be implemented using elementary data structures and a bounded number of plane sweeps to run in $O(nlogn)$ time. A more careful implementation, using the algorithm in [TV], runs in $O(nlogk + nloglogn)$ time. Details of this and various applications of the algorithm may be found in [DJ].

The next section describes an exact algorithm for the above problem.

5. AN EXACT ALGORITHM FOR RECTILINEAR HULLS

Given a rectilinear polygon set with k polygons and a total of n vertices, we have designed an algorithm that computes their minimum vertex hulls in $P(n^k)$ time, where P is a polynomial. The algorithm works even for nonconvex rectilinear polygons. This result shows that the difficulty of the problem is related to the *number of polygons* more than their shapes and spatial positions. As we shall note later, this is *not true* for the three dimensional case. We shall only outline the algorithm, and details may be found in [DJ].

Suppose we *grid* the area exterior to the polygons by extending vertical and horizontal lines through each vertex in both directions until they either extend to infinity, or terminate at some polygon's boundary. It is not hard to see that there exists a set of minimum vertex hulls whose edges lie along the grid. If we perform a brute force search on this grid, the time will be exponential in n. To achieve the claimed time bound, we need to do better. Our algorithm is based on dynamic programming. We will assume the usual nondegeneracies in the input data such as, no two horizontal (vertical) edges share the same y co-ordinate (x co-ordinate) etc, as is usual with geometric algorithms.

We first fill all wells in the input, exactly as the algorithm in the previous section does. After that, there is a difference in how open fragments are modified by this algorithm. The process of filling wells may result in *trapped* regions between pairs of polygons, such that the region's boundary is two open fragments. Within such a region, the boundary of a minimum vertex hull for one polygon is also the boundary of a minimum vertex hull for the other polygon. This boundary fragment can be easily computed by a shortest path search along the grid within the region (where path length is measured by number of bends). For each trapped region, this computation is carried out, the new fragment is added to the set of closed fragments, and the two open fragments are removed from their set. Conceptually, the polygons grow and the trapped regions shrink.

We next break up the remaining area exterior to the polygons into $O(k)$ regions, such that each region is adjacent to at most two polygons. This is done as follows. From planar graph theory it can be shown that the remaining number of open fragments is $O(k)$. Let a *monotone* curve be a curve on the plane whose y co-ordinate monotonically increases, (or monotonically decreases). We split each remaining open fragment into a minimum number of monotone open fragments. We can show that the total number of such fragments is still $O(k)$. We now extend horizontal lines through the end vertices of each monotone open fragment in both directions until they hit other edges, or extend to infinity (Figure 10). Clearly this will break up the exterior area into $O(k)$ regions, and each region will resemble a "generalized" square, such that each vertical side is a monotone portion of some polygon's boundary. Note that each region will contain two fragments of the final hulls, a *left* fragment covering the left vertical side, and a *right* fragment covering the right vertical side.

Consider such a region $ABCD$, as in Figure 10. In the most general case, the grid divides the horizontal edges AB and CD into $O(n)$ points each. Let the sequence of points from left to right be $A = u_0, u_1, ..., u_p = B$ ($C = l_0, l_1, ..., l_q = D$). The left fragment may enter AB through any u_i, and the right fragment through any u_j, $j \geq i$. Similarly, the left fragment may exit CD through any l_r, and the right fragment through any u_s, $s \geq r$. If these entry and exit points are known, we claim (to be proven shortly) that the fragments with minimum number of vertices can be computed in polynomial time. We associate with each region a $O(n^4)$ sized table, where each entry, indexed by the quadruple $(i,\ j,\ r,\ s)$, contains the two fragments with minimum vertices. By the above claim, these tables can be computed in polynomial time. Now a combination of one entry from each table (properly connected using a minimum number of additional vertices), defines a valid set of hulls. The algorithm steps through all the combinations and outputs the one with minimum vertices. Clearly this takes $P(n^k)$ time, where P is a polynomial.

Finally, we show that the tables can be computed in polynomial time. Consider the region $ABCD$. The horizontal lines of the grid break its interior into rectangles. Let the sequence of rectangles be R_0, R_1, ..., R_m, ordered from top to bottom. We shall build the table inductively. Assume the table for the subregion $R_0 \cup R_1 \ldots \cup R_i$ has been constructed. It is not hard to see that the table for the region $R_0 \cup R_1 \ldots \cup R_i + 1$ can be constructed in polynomial time.

The next section describes some details of the three dimensional NP-hardness proof.

6. NP-HARDNESS IN THREE DIMENSIONS

In three dimensions the problem is to construct a minimum vertex hull set for a given polyhedron set. Recall that we require all polyhedrons to be of genus 0. The problem is NP-hard even for convex / rectilinear convex polyhedrons, by a trivial generalization of the proofs of Result 1. However, for simple polyhedrons, we show the problem to be NP-hard even if *only two polyhedrons are present.* This result indicates that in three dimensions, shapes and spatial positions do matter, and two objects could be interwined in a very complex manner which prevents easy computation of their minimum vertex hulls. We outline the proof below.

The decision problem is: Given two polyhedrons with all vertices having integer co-ordinates, and an integer K, is there a pair of hulls which have at least K fewer vertices? Clearly the problem is in NP. For proving NP-completeness, the reduction shall be from 3SAT, and some ideas will be borrowed from the two dimensional construction. As before, we will ignore the details involving integer co-ordinates. Our basic component will be a *prism*, which is a 6-vertex polyhedron, as in Figure 11. The empty tetrahedral region adjacent to the smaller triangular facet is called a *block*, which would be filled if the prism was extended into a tetrahedron. Our construction will consist of (among other things) such prisms with blocks intersecting appropriately. We will also position obstacles alongside all but the smaller triangular facet of each prism, so that the prism can only grow by filling its block. (Henceforth, the construction of a hull will sometimes be regarded as the "growth" of a polyhedron until it becomes the hull).

Given a (not necessarily planar) variable-clause graph of a 3SAT instance, we construct variable, clause, and edge components as in Section 3, except that we replace polygon groups by prisms. Since this is a construction in three dimensions, there is no planarity requirement. Next, we connect all prisms into a single polyhedron (called A) by attaching their larger triangular facets to a common *rail*, which is itself a long polyhedral structure (Figure 12). Clearly A is of genus 0.

Suppose we had obstacles alongside all facets of A except the smaller triangular facet of each prism. Then A can only grow by filling adjacent blocks. But intersecting blocks cannot be simultaneously filled, otherwise A will grow to have a handle, and hence have genus ≥ 1. Thus the next major step of the reduction is to construct a single polyhedron (called B), which will act as an obstacle along all appropriate facets of A. Let C be an artificial polyhedron composed of the union of A and all adjacent blocks. Clearly C may have a large genus. Let B be a large tetrahedral object which completely contains C in its interior. It has two connected surfaces, one exterior, and one interior, with the latter adjacent to C. To force it to have a single connected surface, we drill a hole (with a suitably small triangular cross section) from the exterior surface to the interior surface, such that the interior opening is adjacent to some facet of A's rail. Now B is a polyhedron with a single connected surface, though with a possibly large genus.

At this stage, the problem of finding a minimum vertex hull for only A, in the presence of B, is at least as hard as finding a satisfying assignment for the 3SAT instance. This is because the hull for A can only select nonintersecting blocks, and clearly cannot grow through the hole in B. To complete the construction, we have to satisfy two more requirements. First, we have to convert B to a genus 0 object, and second, pose the problem as finding hulls for both objects simultaneously.

To satisfy the first, we break internal handles of B by cutting portions of suitably small width (Figure 13), until the genus becomes 0. However, one complication arises. It may now be possible for A to grow by selecting intersecting blocks, and yet avoid its resulting handle by also growing through a created gap in a former handle of B. To make it expensive for A to grow through these gaps, we let each cut portion have a shape roughly like a cup with at least as many vertices as A currently has.

The reduction is now complete. The second requirement is automatically satisfied because of the following. Suppose we find a minimum vertex hull for A alone. Then we can create a minimum vertex hull for B by growing B until it is everywhere adjacent to A's hull, except at the interior opening of the hole. We omit further details and claim that this reduction is sufficient to prove that computing minimum vertex hulls for simple polyhedrons of genus 0 is NP-hard.

A similar result can be established for a pair of rectilinear polyhedrons. As a corollary, the generalized Klee's problem can also be shown to be NP-hard, even for rectilinear polyhedrons. Details may be found in [DJ].

7. OPEN PROBLEMS

In three dimensions the main open problem is whether it is hard to construct hulls for convex polyhedron sets with a *bounded* number of polyhedrons. We do not think the problem is NP-hard. Results on this problem may shed some light on Klee's original problem. It would also be interesting if some results on polyhedrons with higher genus could be established.

In neither two nor three dimensions have we designed approximation algorithms for the simple polygons (polyhedrons) problem. Good approximation algorithms, or even heuristics, would be interesting because they have practical applicability.

It seems to be necessary to define a good *general measure of simplicity*, rather than special ones such as minimum vertices, area, etc. Such a measure should be investigated from a computational complexity point of view.

8. REFERENCES

[A] H. Alt, "Approximation of Convex Polygons by Rectangles and Circles", Manuscript, (1989).
[AB] A. Aggarwal, H. Booth J. O'Rourke, S. Suri, C.K. Yap, "Finding Minimal Convex Nested Polygons", Proc. 1^{st} Annual Symp. on Comp. Geometry (1985), pp 296-303.
[CKV] K. Clarkson, S. Kapoor, P. Vaidya, "Rectilinear Shortest Paths through Polygonal Obstacles in $O(n(logn)^2)$ Time", Proc. 3^{rd} Annual Symp. on Comp. Geometry (1987), pp 251-257.
[CY] J.S. Chang, C.Y. Yap, "A Polynomial Solution for Potato Peeling and Other Polygon Inclusion and Enclosure Problems", IEEE Symp. on Foundations of Computer Sciences, (1984), pp 408-417.
[DJ] G. Das, D. Joseph, "Minimum Vertex Hulls for Polyhedral Domains", Technical Report, (Nov 1989), University of Wisconsin-Madison.
[GJ] M. Garey, D. Johnson, "Computers and Intractability", Published by W. H. Freeman and Co. (1979).
[KL] V. Klee, M.C. Laskowski, "Finding the Smallest Triangle Containing a Given Convex Polygon", Journal of Algorithms, (1985).
[L] D. Lichtenstein, "Planar Formulae and their Uses", SIAM Journal Comp., (1982), pp 329-343.
[O] J. O'Rourke, "Computational Geometry Column", SIGACT News, (1988).
[TV] R. Tarjan, C. Van Wyk, "An $O(nloglogn)$-Time Algorithm for Triangulating a Simple Polygon", SIAM Journal Comp. (1988), pp 143-178.

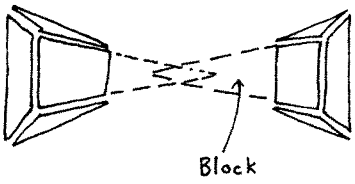

FIGURE 1.

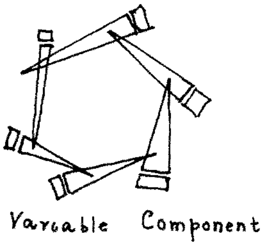

FIGURE 2.

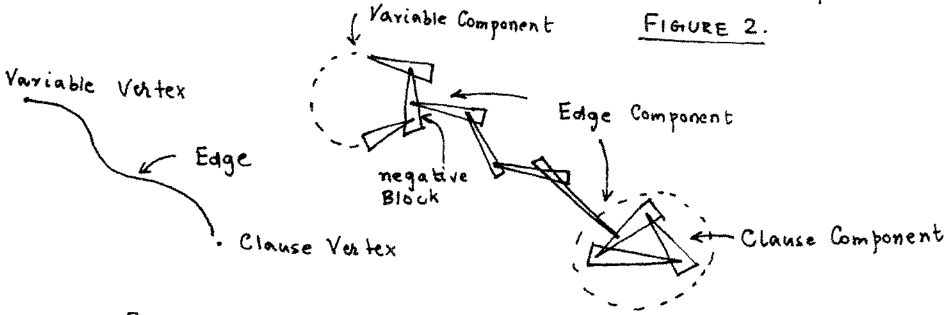

FIGURE 3.

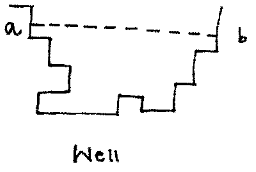

FIGURE 4.

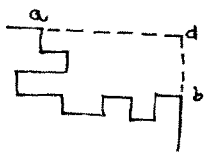

FIGURE 5.

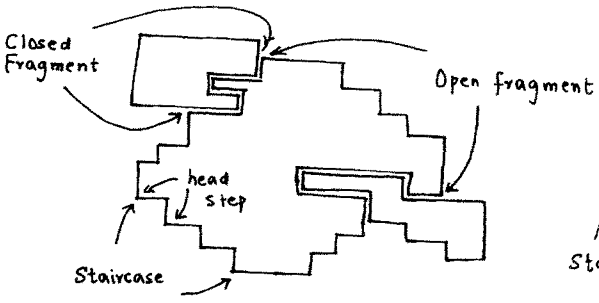

FIGURE 6.

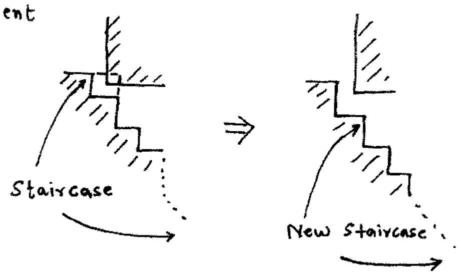

FIGURE 7.

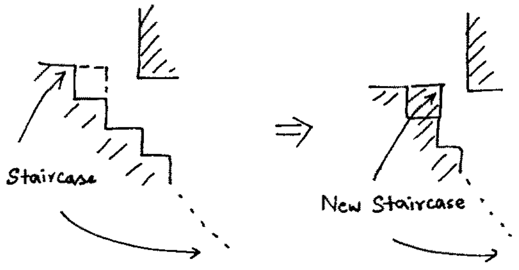

FIGURE 8.

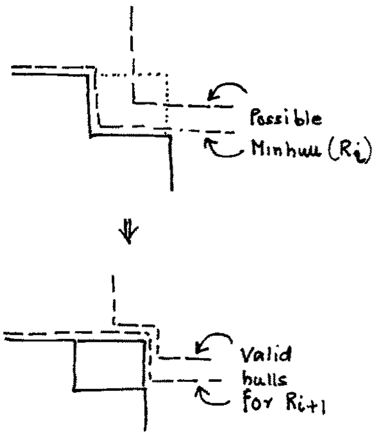

FIGURE 9.

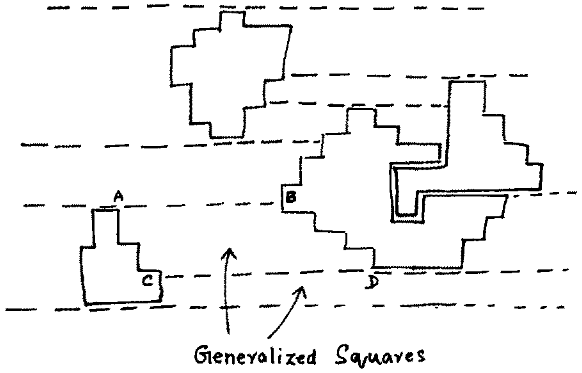

FIGURE 10.

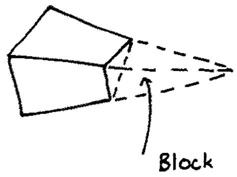

FIGURE 11.

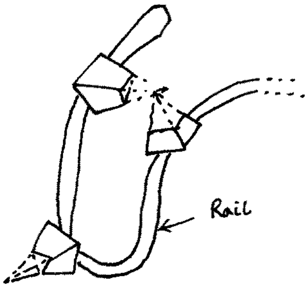

FIGURE 12.

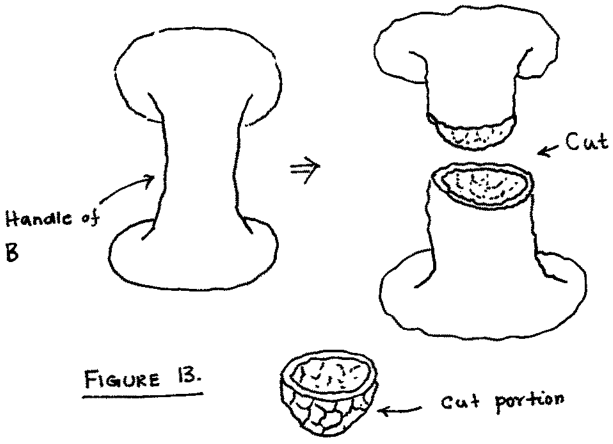

FIGURE 13.