Technical Report No. 2004-480 INHERENTLY PARALLEL GEOMETRIC PROBLEMS[*]

Selim G. Akl
School of Computing
Queen's University
Kingston, Ontario, Canada K7L 3N6

April 30, 2004

Abstract

A new computational paradigm is described which offers the possibility of superlinear (and sometimes unbounded) speedup, when parallel computation is used. The computations involved are subject only to given mathematical constraints and hence do not depend on external circumstances to achieve superlinear performance. The focus here is on geometric transformations. Given a geometric object A with some property, it is required to transform A into another object B which enjoys the same property. If the transformation requires several steps, each resulting in an intermediate object, then each of these intermediate objects must also obey the same property. We show that in transforming one triangulation of a polygon into another, a parallel algorithm achieves a superlinear speedup. In the case where a convex decomposition of a set of points is to be transformed, the improvement in performance is unbounded, meaning that a parallel algorithm succeeds in solving the problem as posed, while all sequential algorithms fail.

Keywords: Parallel computation, superlinear speedup, computational geometry, geometric transformation, edge replacement, edge flip, triangulation, convex decomposition, real-time computation.

1 Introduction

Parallel computation is a form of information processing whereby n processors, $n > 1$, cooperate to solve a computational problem by working on it simultaneously. The expectation is that this approach speeds up computations that would otherwise require an inordinate amount of time if performed sequentially (that is, using one processor). Over the last twenty five years, considerable progress has been achieved to fulfill the promise of parallel computation. Results, both in theory and in practice, were obtained to demonstrate that, in fact, significant improvements are possible, not only in the *speed* with which a solution is obtained, but also in the *quality* of the solution itself [8].

[*]This research was supported by the Natural Sciences and Engineering Research Council of Canada.

1.1 Speedup and Quality-up

Suppose that T_1 is the time required (in the worst case) by the best sequential algorithm for solving a given computational problem $\mathcal{P}$, and that T_n is the time required (to solve $\mathcal{P}$, also in the worst case) by an n-processor parallel algorithm. The *speedup* achieved by the parallel algorithm over the sequential one is the ratio T_1/T_n. Similarly, if V_1 is the value of the solution to $\mathcal{P}$ obtained sequentially and V_n is the value of the solution to $\mathcal{P}$ obtained in parallel, then the improvement in quality (or *quality-up*) afforded by parallel computation is V_n/V_1. Note here that the way in which the 'value' of a solution is measured depends on the problem at hand; thus, for example, the quality of a (lossless) source coding algorithm is determined by the compression rate it delivers. Similarly, the quality of a statistical measure depends on the size of the sample used to compute it. By the simulation principle (which says that any parallel algorithm can be simulated on a sequential machine), it follows that both the speedup and the quality-up offered by n processors are each at most a linear function of n. The validity of this principle is demonstrated by the fact that it holds for the vast majority of conventional computations. As it turns out, however, various computational paradigms are being discovered where the simulation principle no longer applies. In these unconventional yet realistic computations, parallel computation provides improvements in speed and quality that are often *superlinear* in n (that is, for example, of the form n^x or x^n, for $x > 1$). Sometimes the improvement is unbounded [5].

1.2 Computational Paradigms

Our previous work has uncovered two general computational paradigms within which parallel algorithms lead to a superlinear improvement in performance with respect to their sequential counterparts. These paradigms, sketched in the next section, are: computing in real time and computing within the bounds of laws of nature. In this paper, a new computational paradigm is described which (like its two predecessors) offers the possibility of superlinear (and sometimes unbounded) speedup, when parallel computation is used. Unlike its predecessors, however, it does not depend on external circumstances to achieve superlinear performance.

The new paradigm concerns computations that are subject to given mathematical constraints. Specifically, we are interested in transformations on mathematical objects. Given an object A with some property, it is required to transform A into another object B which enjoys the same property. If the transformation requires several steps, each resulting in an intermediate object, then each of these intermediate objects must also obey the same property. Our focus in this paper is on rectilinear Euclidean geometry in the two-dimensional plane. We are concerned with specific geometric objects, namely, triangulations (of polygons and point sets), and convex decompositions of sets of points. We show that in transforming one triangulation into another, a parallel algorithm achieves a superlinear speedup. In the case where a convex decomposition is to be transformed, the improvement in performance is unbounded, meaning that a parallel algorithm succeeds in solving the problem as posed, while all sequential algorithms fail.

1.3 Computational Models

In terms of models of computation used, we select the Random Access Machine (RAM) and the Parallel Random Access Machine (PRAM) [3], for their simplicity and widespread use, this choice being of no consequence as far as the results in this paper are concerned. The RAM is the standard sequential model, consisting of a single processor capable of reading a datum from memory, performing an elementary arithmetic or logical operation (such as addition, for example), and writing a datum back into memory, all in one computational step. The PRAM is equipped with N processors, $N > 1$, sharing a common memory from which they can read and to which they can write simultaneously. Each PRAM processor is identical to the RAM processor.

The remainder of this paper is organized as follows. In section 2, we review briefly the two existing paradigms for superlinear performance. The new paradigm is described in section 3. Some concluding thoughts are presented in section 4.

2 Two Existing Paradigms for Superlinear Performance

Over the last few years, two paradigms have emerged that allow for superlinear performance in parallel computation, namely, computing in a real-time environment with deadlines, and computing under the influence of laws of nature. The growing presence of these new paradigms within the field of computing, and their increasing applications in society (with ubiquitous and mobile computing, for example, becoming more prevalent), lent timeliness and importance to their theoretical study. Such study uncovered the fact that each improvement in performance obtained through these paradigms is consistent and provable, in the sense that it occurs in every instance of the computational problem under consideration. In addition, this improvement is independent of any discrepancies between the sequential and parallel computers used. These characteristics distinguish the results summarized in this section from previous ones, (see [18, 36, 37, 43, 48, 52, 58], for example), where a superlinear speedup occurs only occasionally, or is achieved either because the sequential algorithm used is inefficient, or because the size of the memory on the sequential computer is restricted. An additional distinguishing feature of these paradigms is that, in some circumstances, they lead to a superlinear improvement in the *quality* of the computed solution.

2.1 Computing in the Presence of Real-Time Deadlines

The notion of *real time* is an intuitive one. The term evokes simultaneity, liveness, and immediacy; it even conveys a sense of urgency and the need for an instantaneous action. Yet, *real-time computation* is difficult to capture. The numerous definitions in the literature [19, 50, 63] are witness to this fact (for an in-depth survey and analysis see [20]). One reason for this situation is that, in today's fast-paced society where computers can perform billions of operations per second, *every* computation seems to be performed in real time. Whether we are dealing with a bank machine or running a flight simulator, we have become accustomed to expect the result of our computations *now*. In a major textbook on the subject [47], the authors acknowledge the difficulty in delineating what computations to characterize as real-time ones. They proceed to define real-time computations as those computations in which failure to produce the answers after a certain time would be catastrophic, involving loss of human life. This state of affairs provided the initial motivation for our work on a unifying definition of real-time computation that avoids the extremes just discussed. A model was sought that can be adapted to apply at any point on the above spectrum, depending on the circumstances and requirements of a particular application. One such model was recently derived that uses a complexity-theoretic approach based on formal languages [22, 23, 27, 25]. For any real-time computation, this model allows for the construction of a language such that the computational resources required in order to accept this language are of the same order as the resources that are required to carry out the given real-time computation. A second approach is proposed in [4], where a general framework is described for the study of real-time algorithms. In it, state space traversal is used as a model for computational problems in a real-time environment, and algorithms are designed to solve these problems using a method known as discrete steepest descent. Both approaches were used to obtain parallel algorithms for real-time computation. They include algorithms for data accumulation [21, 26] and data correction [24]. Most important for our theme here, however, were the algorithms that demonstrated superlinear performance improvement in several applications [7, 12, 11, 10, 54, 55, 56, 57]. For example, we show that for a computation involving a one-way function of n variables, an n-processor parallel algorithm leads to a speedup exponential in n. Similarly, a quality-up exponential in n is obtained when using n processors to find the zero of a continuous function numerically. In fact, in a cryptographic problem, the improvement in quality due to parallelism is unbounded. A number of papers by other authors followed, reporting similar results [35, 64].

2.2 Computing Under the Influence of Natural Laws

We now turn to a second paradigm in which a parallel algorithm using n processors leads to a performance that is far beyond an improvement in speed or quality linear in n. Here we are concerned with computations whose characteristics are akin to certain unique phenomena that are observed in different domains of science. The focus is on systems whose computations are subject to natural law—typically, systems whose variables are altered unpredictably whenever one of these variables is measured or modified. Examples of such computational environments include those in which *Heisenberg's uncertainty principle* of quantum physics is witnessed, or those in which *Le Châtelier's principle* of chemical systems under stress manifests itself. Our recent investigations have uncovered practical computations that are inherently parallel *in the strong sense*, meaning that they are efficiently executed in parallel, but impossible to carry out sequentially [13, 14, 15]. Specifically, we showed that a resistance-inductance-capacitance circuit (a linear dynamical system) undergoes significant perturbations that affect its dynamical behavior when some of its parameters (such as current, voltage, and so on) are measured sequentially. By contrast, these perturbations could be eliminated when the measurements are performed in parallel. Similarly, for the Belousov-Zhabotinskii chemical reaction (a nonlinear dynamical system), we showed that measurement disturbs the equilibrium of the system and causes it to go from a stable into an unstable state. If, however, several measurements are performed in parallel, the resulting perturbations cancel each other out and the system remains in a stable state. These results confirm an earlier existence proof provided in [9] regarding such computations. There we showed that, given a system in equilibrium, the task of measuring the system's parameters, computing new values for them, and setting them to their new values before the system settles on its own into a new but undesirable state of equilibrium, can be performed in parallel but not sequentially. A further study of the properties of such computations is needed and may have profound consequences on the theory and practice of computing.

3 Computing Subject to Mathematical Constraints

The two computational paradigms described in the previous section rely on external factors in order to achieve superlinear performance when parallel computation is used. Thus, in real-time computation the conditions governing the computational environment are, in a fashion, fully determined by the human in charge and by the model of computation used. For example, if it is deemed that the arrival rate of the data is too high, it is possible for the people responsible for the computation to slow down the arrival rate, or to extend the deadline by which a solution is to be delivered, or to use a faster computer, and so on. Similarly, for the paradigm of computing under the influence of natural law, it is the laws of nature that prevail: the outcome of a computation is governed by those natural laws that are in vigor within the computational environment. In this section we propose a paradigm that does not depend on any outside conditions to attain superlinear performance in parallel. Rather, the conditions on the computation are fully dictated by the mathematical properties of the objects being operated upon.

Let A be a given physical object with a certain property p, and let B be another object also satisfying p but for which only a mathematical definition is available. Such mathematical definition may take, for example, the form of a graph, a set of equations, and so on. It is required to construct B by applying a sequence of simple transformations to A such that each intermediate object (on the path leading from A to B) also satisfies p. We will be asking whether the transformation from A to B can be performed as defined, and if so, what is the number of steps required in the worst case.

It is interesting to note here that this question is reminiscent of the *edit distance* problem for strings (see, for example, [29]). A simple version of this problem is sated as follows. Two strings u_1 and u_2, made up of symbols from a finite alphabet, are given. Also provided is a set of allowed operations on such strings, such as replacing a symbol by another, deleting a symbol, inserting a symbol, reversing the order of two adjacent symbols, and so on. What is the minimum number of consecutive operations needed to transform u_1 into u_2?

One reason for studying these transformations is that they serve as a mathematical model for the following general problem. A large data (or state) space is to be searched. In this space, it is possible to go from one datum (or state) to a nearby datum (or state), called its *neighbor*, by a simple transformation. It so happens that a single computer can only 'see' a small subset of this space at any given time. Thus, such a computer can only perform a search of the local neighborhood at its disposal, going from one datum (or state) to another using simple transformations. This explains why we insist that each intermediate object must satisfy the same property (or be of the same type) as A and B. Another motivation for this condition is that in some cases one may wish to interrupt the sequence of transformations at any given time, and ask for the object reached thus far by the algorithm to be returned. This circumstance would be an example of what is sometimes referred to in the literature as an "anytime" algorithm [65].

In what follows, we examine transformations on two types of geometrical objects in the plane, namely, triangulations and convex decompositions (also known as convex subdivisions).

3.1 Triangulations

Recall that a triangulation is a planar graph all of whose faces are triangles. A set of points in the plane can be triangulated by connecting pairs of points with (straight-line) edges until no edge can be added without creating an intersection with an existing edge. Similarly, the interior of a polygon can be triangulated by connecting its vertices with diagonals, such that each diagonal lies entirely inside the polygon, until no diagonal can be added without intersecting an existing diagonal.

Suppose that A and B are two grids, each in the shape of a triangulation of a polygon with $2n$ vertices, as shown in Fig. 1 for $n = 4$.

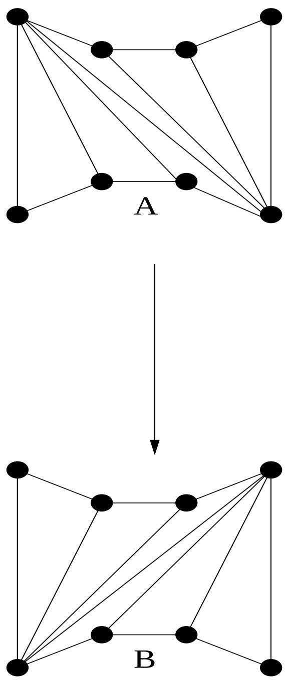

Figure 1: It is required to transform A into B.

It is required to obtain B from A by applying a sequence of transformations such that each intermediate grid is also a triangulation. A transformation is defined as the removal of *at least one* edge and the replacement of every removed edge by *exactly one other*, as shown in Fig. 2.

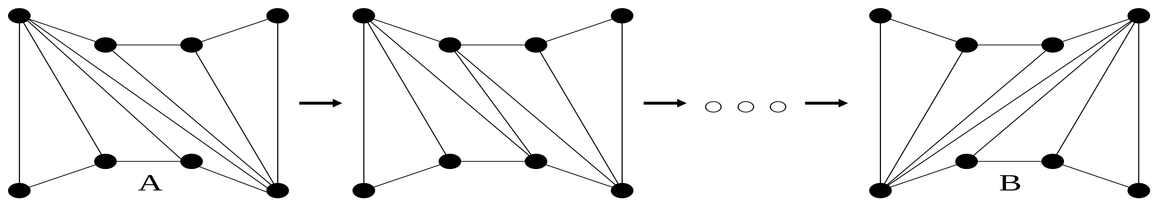

Figure 2: Transforming triangulation A into triangulation B.

3.1.1 Sequential Approach

A sequential algorithm can only replace one edge of a triangulation by another edge (such that the resulting figure is still a triangulation of the given polygon). It is shown in [41] that for triangulations with $2n$ vertices of the form A and B, respectively, $(n-1)^2$ transformations (each replacing one edge by another) are *required* by a sequential algorithm. For $n = 4$, a sequence of 9 transformations is illustrated in Fig. 3.

In order to see why the bound holds, note that the $2n$-vertex polygon to be triangulated consists of two chains, an n-vertex upper (convex) chain and an n-vertex lower (concave) chain (as shown in Fig. 1 for $n = 4$). When the polygon is triangulated, each triangle has two vertices either on the upper chain or on the lower chain. Let us associate each triangle having two vertices on the lower chain with a 0 and each triangle having two vertices on the upper chain with a 1, as shown in Fig. 4. Triangulation A may now be thought of as the string

$$000\ldots000111\ldots111$$

while triangulation B may be thought of as the string

$$111\ldots111000\ldots000.$$

There are $n-1$ 0s and $n-1$ 1s in each string. In order to get from the first string to the second, each 1 has to move $n-1$ positions to the left (and each 0 $n-1$ positions to the right). This requires $(n-1)^2$ steps.

Clearly, this $\Omega(n^2)$ lower bound also applies to the more general case in which a triangulation of a set of n planar points needs to be transformed into another such triangulation. Furthermore, it is a tight bound since any triangulation (of a polygon or point set) with $O(n)$ edges can be transformed into another in $O(n^2)$ steps [41].

3.1.2 Parallel Approach

Suppose that we add (temporarily) the following condition to the problem of transforming triangulation A into triangulation B using a parallel algorithm: Every time an edge (common to two adjacent triangles t_a and t_b in a triangulation) is to be replaced, the edge replacing it must be the other diagonal of the (convex) quadrilateral formed by t_a and t_b. It should be clear that this condition holds implicitly in the sequential case since, once the outgoing edge has been selected, the choice for the incoming edge is unique. Also, it should be clear that two edges can be replaced simultaneously (by two other edges) provided that they are

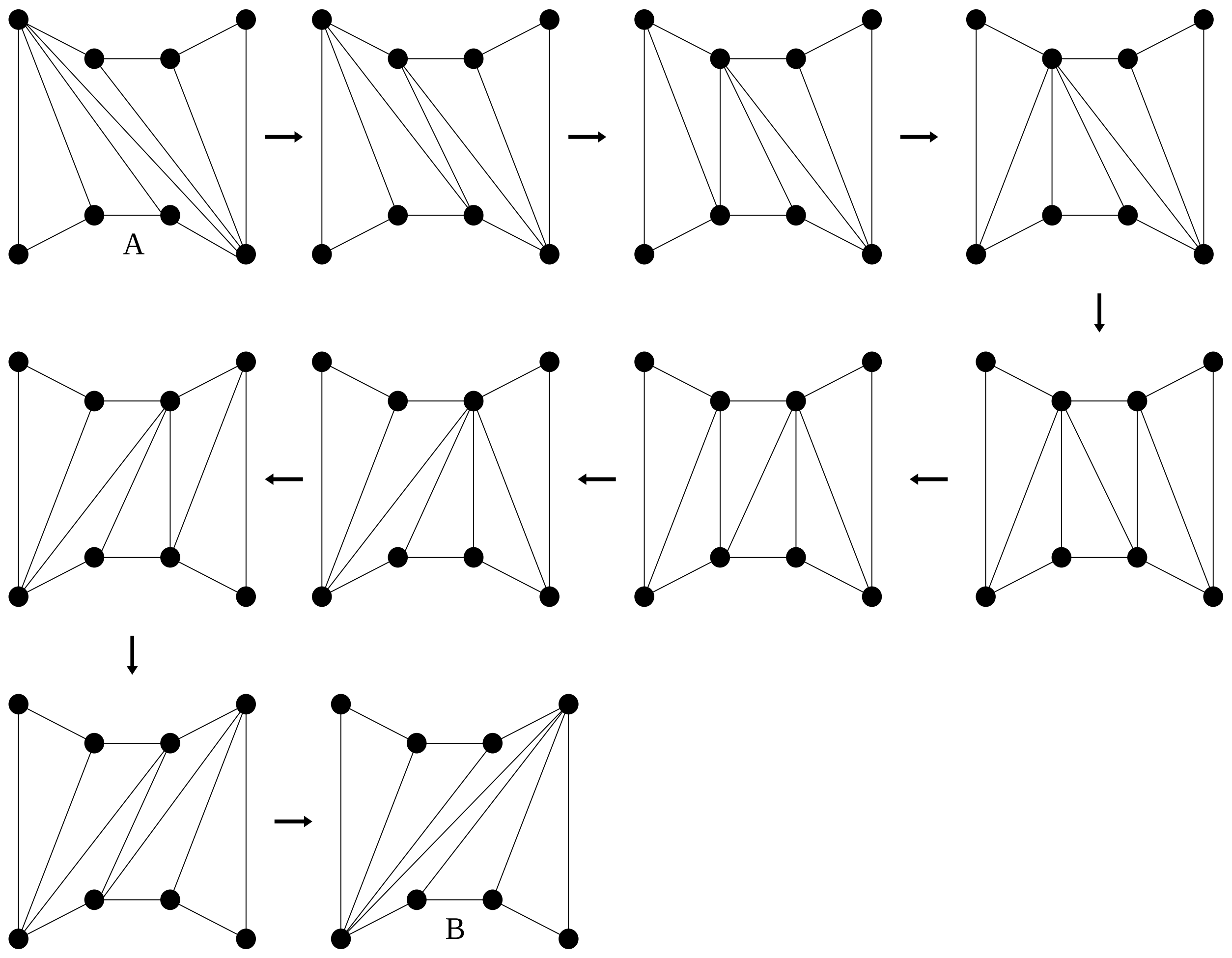

Figure 3: Transforming A into B sequentially.

not the sides of the same triangle; in this case the two edge replacements are said to be *independent*.

In this special case, a parallel algorithm solves the problem in $O(n)$ steps, each involving possibly several independent edge replacements [33]. To show why this is true, we once again use the mapping from triangulations of polygons to strings of length $2(n-1)$, consisting of $n-1$ 0s and $n-1$ 1s. Consider the string representing triangulation A.

$$000\ldots000111\ldots111$$

It contains a single 01 pair that can be flipped. Once this is done, we obtain the string

$$000\ldots001011\ldots111$$

containing two 01 pairs, *which can be flipped simultaneously*, leading to the string

$$000\ldots010101\ldots111$$

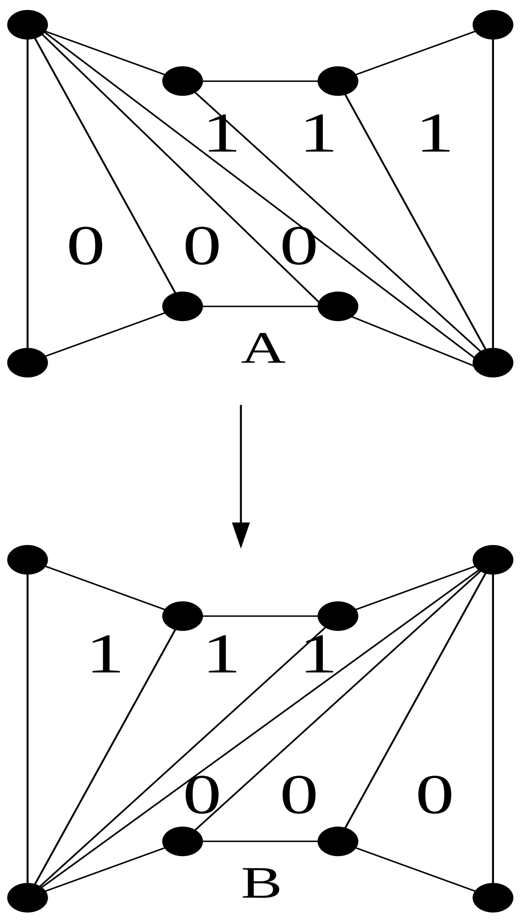

Figure 4: Mapping a triangulation into a binary string.

which now contains three pairs 01 that can be flipped simultaneously. This continues until we reach the string

$$010101\ldots010101$$

made up of $n-1$ pairs 01. After flipping these, the number of 01 pairs drops to $n-2$, $n-3$, ..., 1, until

$$111\ldots111000\ldots000$$

is reached, representing triangulation B. The number of strings generated from the initial one is $2n-3$. Therefore, with exactly $n-1$ processors, this requires $2n-3$ steps, since at most this many processors are needed to perform simultaneously all flips required to transform one string into another. It is shown in [33] that this bound holds for all triangulations of polygons as well as all triangulations of point sets. Given that the sequential solution requires $(n-1)^2$ steps, the speedup afforded by the parallel solution here is on the order of n, that is, linear in the number of processors. (Note that triangulations of convex n-gons can be transformed in parallel in a number of steps on the order of $\log n$. Given that the sequential number of steps in this case is linear in n, the speedup is on the order of $n/\log n$, which is sublinear in the number of processors.) Similar results appear in [45, 46].

Suppose now that we lift the restriction added at the beginning of this section. Thus the edges removed in going from one triangulation into another must only satisfy the requirement that the resulting figure remains a triangulation (no intersections and every face is a triangle). This change makes no difference in the sequential case as pointed out at the beginning of this section. It is clear, however, that $2n - 3$ processors operating in parallel can now solve the problem of transforming triangulation A to triangulation B (see Fig. 1) *in one step*: Each processor replaces one edge of A by an edge of B. The speedup in this case is on the order of n^2, that is, quadratic in the number of processors. This result holds for all types of triangulations.

3.2 Convex Decompositions

The parallel algorithm at the end of the previous section is reminiscent of the furniture-moving paradigm [6]:

> "[A] large piece of furniture [..] needs to be moved from one place to another. One mover working alone is unable to lift, push, or drag the item and, in order to move it, must take it apart, transport each of the parts individually, and then put them back together at the indicated spot. The job requires one hour. On the other hand, four movers working together can simply lift the piece of furniture and put it in its new location in 15 seconds."

This observation motivated us to search for other instances of this phenomenon. A result beyond superlinear speedup is provided by convex decompositions of point sets, that is, planar graphs each of whose faces is a convex polygon. Consider the convex decompositions A and B in Fig. 5. Given A, it is required to transform it to B. A transformation involves replacing at least one edge by another. Every intermediate figure (on the path leading from A to B) must be a convex decomposition.

It can be verified that *no sequential algorithm* can transform A into B by a sequence of transformations, each involving a single edge replacement, such that every intermediate graph is a convex decomposition [53]. As shown in Fig.6, the removal of one edge and its replacement by another inevitably leads to a concavity, thus violating the requirement. In fact, a stronger result is obtained in [53] for the decompositions A and B in Fig. 5, consisting of $3n$ vertices: the smallest number of edges that can be removed and replaced without creating a concavity is exactly n. Clearly, no sequential algorithm can replace n edges simultaneously. Therefore, since the problem cannot be solved sequentially, we say that the sequential number of steps is *infinite*.

A parallel algorithm, on the other hand, easily performs the transformation in one step, using the furniture moving approach: All edges in A to be replaced are lifted simultaneously and the edges replacing them deposited to create B. Therefore, in this case the speedup is *unbounded*.

It is interesting to note here that one sequential operation suffices to destroy an existing mathematical property of the object under consideration. Indeed, the decomposition is no

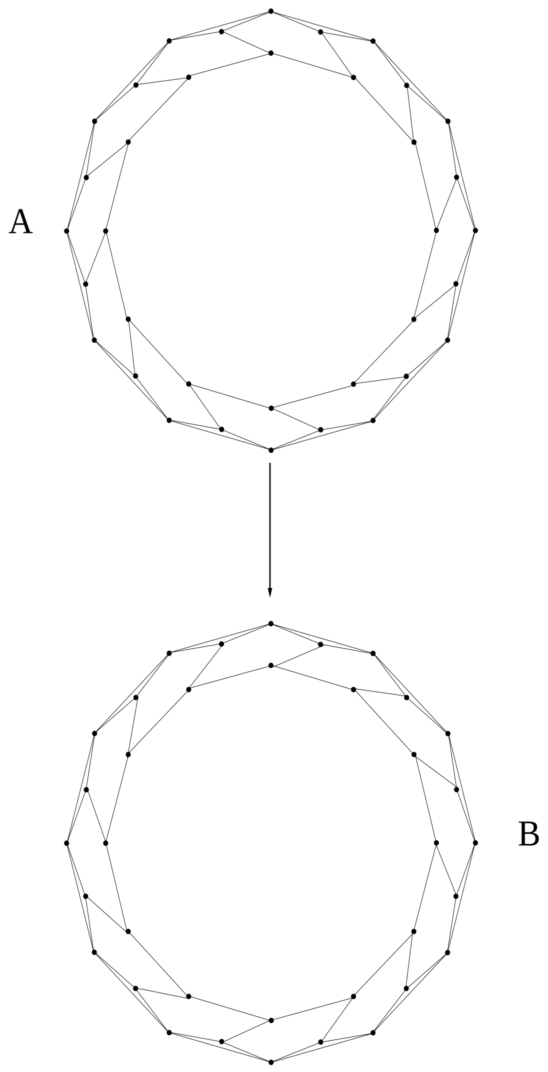

Figure 5: Transforming convex decomposition A into convex decomposition B.

longer convex after a *single* edge replacement. This situation is surprisingly similar to that in the paradigm described in section 2.2, namely, computing under the influence of natural laws. There, a physical system loses its state of equilibrium if *one* of its parameters is measured or modified.

4 Conclusion

Mathematically constrained computing is a new paradigm conducive to superlinear performance in parallel computation. Problems belonging to this paradigm are *inherently parallel* in the sense that they are quickly solvable in parallel, but very slow to solve sequentially. Typically, for a problem of size n, the speedup afforded by a parallel algorithm using n processors over a sequential one is a superlinear function of n. Some problems cannot be solved at all sequentially, in which case it is fair to say that the speedup is unbounded.

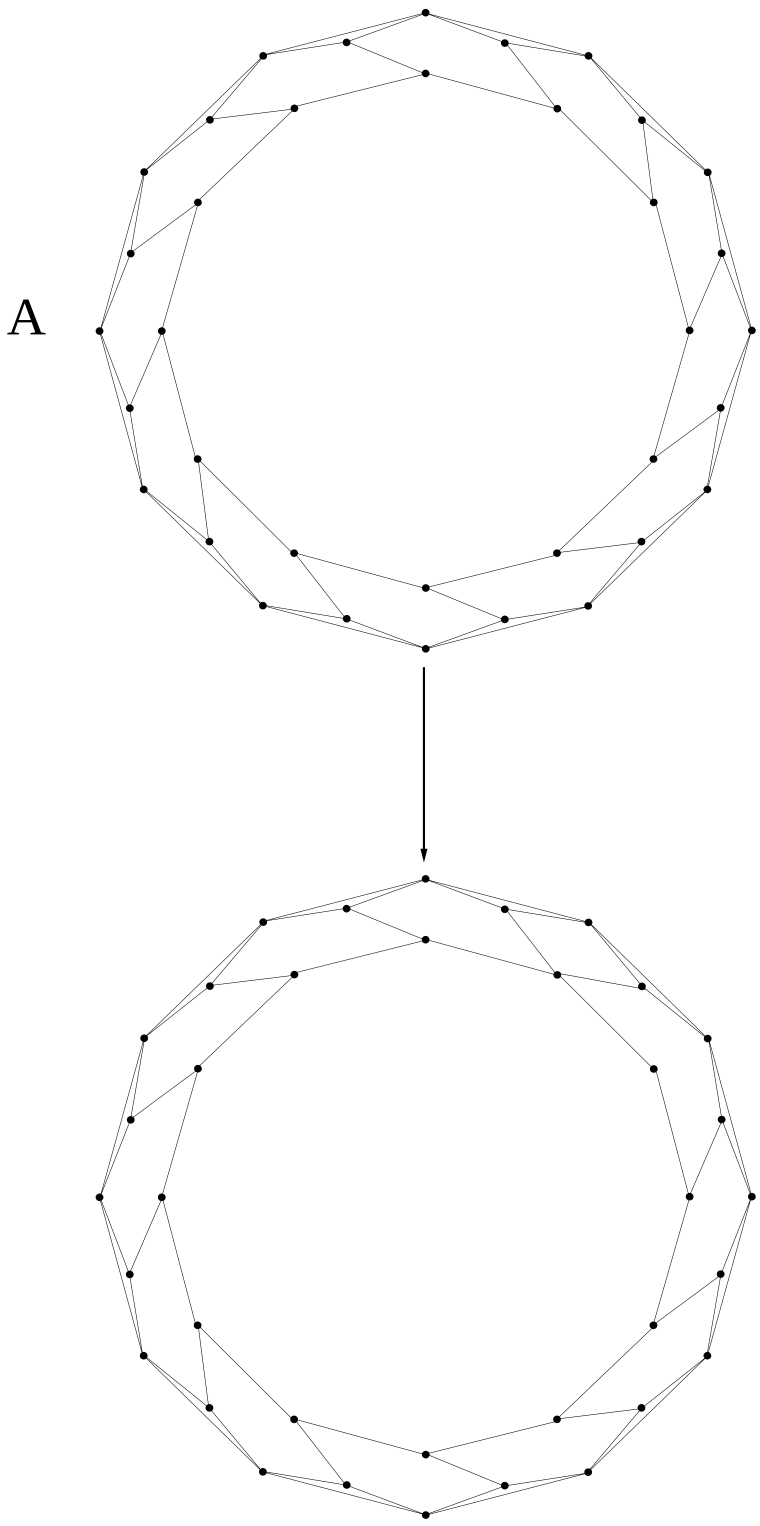

Figure 6: A sequential approach creates a concavity.

In this paper the focus has been on one family of such computations, namely, inherently parallel geometric transformations. Two examples were presented. Both are concerned with transforming one geometric figure into another of the same class using edge replacements. An edge replacement operation was defined as the removal of (at least) one edge and the insertion of a new edge (for each edge removed), such that, when the substitution is completed, the figure thus created is of the same class as the original figure. It is shown here that, in the worst case, when transforming one triangulation into another, the speedup is quadratic in the size of the triangulation. By contrast, for transforming a convex decomposition into another, we prove (by counterexample) that the problem is not solvable in all cases sequentially, but always solvable in parallel.

It is interesting to note here that transforming geometric figures through edge replacement (sometimes called *flips*) has several applications. Flips in triangulations, in particular, have received considerable attention. They have been used in computational geometry to construct various types of triangulations in two and higher dimensions [31, 32, 33, 41, 44, 45, 46, 49] and to compute the visibility graph of a set of objects in the plane [59]. In combinatorics, their applications include the enumeration of rooted triangulations [16] and performing rotations on binary trees (taking advantage of the bijection between triangulations of a convex n-gon and binary trees with n-2 internal nodes) [40, 51, 61]. Flips are also used in graphics in a variety of ways: from compression techniques for visualization and transmission [38, 39, 60] to adapting to a new topology [28, 30, 62]. Last but not least, they arise quite naturally in the finite elements method of numerical analysis, one of the most important applications of triangulations [34]. In all of these domains, one typically begins with a triangulation (often arbitrary), which is gradually transformed into another 'better' (or simply different) triangulation through a sequence of edge replacements (flips).

Several avenues offer themselves for future research. For example, we are currently investigating the problems of transforming a number of rectilinear planar graphs (such as, for example, Hamilton paths, perfect matchings, sets of cycles, and so on [42]). Recall that, a cycle in a graph is a sequence of consecutive edges that starts at a node of the graph and ends at the same node. By this definition, each node is incident on exactly two edges. Also, for a set $\mathcal{P}$ of n points in the plane,

1. a rectilinear planar Hamilton path H on $\mathcal{P}$ is a connected graph whose nodes are the points in $\mathcal{P}$ and whose edges are a sequence of non-intersecting straight-line segments, such that each point is connected to exactly two other points, except for the first and last points, each of which has only one neighbor,

2. a rectilinear planar perfect matching M on $\mathcal{P}$ consists of a set of non-intersecting straight line segments each connecting a pair of points of $\mathcal{P}$ such that each point is incident on exactly one segment (assuming n is even), and

3. a rectilinear set of cycles C on $\mathcal{P}$ consists of a set of non-intersecting cycles made up of straight-line segments connecting the points of $\mathcal{P}$ such that each cycle has length i, where $1 \leq i \leq n$. When $i = 1$, the cycle is a loop connecting a point to itself, and when $i = n$, there is only one cycle, called a *Hamilton cycle.*

We are interested in finding out whether any of these geometric objects can be transformed into another of the same type using edge replacements such that each intermediate object is of the same type. Put differently, given $\mathcal{P}$, let $\mathcal{G}_{\mathcal{H}}$ be the (meta)graph each of whose nodes represents a rectilinear planar Hamilton cycle defined on $\mathcal{P}$, such that two nodes of $\mathcal{G}_{\mathcal{H}}$ are connected if and only if the two Hamilton cycles they represent can be obtained from one another through a single edge replacement. (Meta)graphs $\mathcal{G}_{\mathcal{M}}$ and $\mathcal{G}_{\mathcal{C}}$ are defined similarly. The question now is whether $\mathcal{G}_{\mathcal{H}}$ (or $\mathcal{G}_{\mathcal{M}}$, or $\mathcal{G}_{\mathcal{C}}$) is connected. Both possible answers to this question lead potentially to interesting considerations:

1. Suppose that $\mathcal{G}_{\mathcal{H}}$ (or $\mathcal{G}_{\mathcal{M}}$, or $\mathcal{G}_{\mathcal{C}}$) is shown to be connected. In this case, two new questions suggests themselves:

   (a) What is the diameter of the (meta)graph, that is, the length of the shortest path separating its furthest two nodes? If the diameter is superlinear, then a sequential algorithm requires a number of steps superlinear in n to transform one geometric object into another. By contrast, a parallel algorithm performs the transformation in constant time. This would it provide another example of superlinear speedup.

   (b) Does the (meta)graph contain a Hamilton cycle (that is, a cycle that goes through every node exactly once)? This property would signify that from any one geometric object – for instance a rectilinear planar Hamilton path, or a rectilinear planar perfect matching, and so on – *all* other objects of the same type defined on $\mathcal{P}$ can be generated using single-edge replacements.

2. On the other hand, suppose that $\mathcal{G}_{\mathcal{H}}$ (or $\mathcal{G}_{\mathcal{M}}$, or $\mathcal{G}_{\mathcal{C}}$) is not connected. This means that there are at least two geometric objects such that neither can be transformed into the other through single-edge replacements. In this case, the transformation is *impossible* sequentially, while eminently computable in parallel. This would imply another instance of unbounded speedup.

A rectilinear planar spanning tree T on $\mathcal{P}$ is a connected graph with no cycles whose nodes are the points in $\mathcal{P}$ and whose edges are non-intersecting straight-line segments. It has been shown that $\mathcal{G}_{\mathcal{T}}$ is connected for three different definitions of the edge replacement operation:

1. In [17] an edge is removed and replaced by another with no additional restrictions (save for the usual requirement that the resulting figure be a rectilinear planar spanning tree).

2. In [1] an edge is replaced by another provided that the replacement reduces the total length of the tree.

3. In [2] an edge is replaced by moving one of its endpoints along an adjacent edge.

However, the question of whether $\mathcal{G}_{\mathcal{T}}$ is Hamiltonian remains open. To our knowledge, it is also open for the (meta)graph of triangulations of a set of points and for that of triangulations of a polygon.

The next logical step would be to investigate more general geometric objects, such as rectilinear *outerplanar graphs*. These are graphs that have a crossing-free straight-line embedding in the plane, such that all vertices are on the same face.

Finally, it is an open question whether other branches of mathematics, besides geometry, offer the possibility of superlinear speedup within the context of the (computing-subject-to-mathematical-constraints) paradigm introduced in this paper. If other examples of such computations are found, will mathematical transformations, analogous to the geometric transformations described here, play a major role in achieving the improved performance through parallelism?

Acknowledgments

I am grateful to David Rappaport for bringing reference [33] to my attention. David, along with Henk Meijer, generously shared with me their result in [53] and engaged me in interesting discussions on the subject. James Stewart showed me how edge flips are used in his graphics work [62]. I value their friendship.

References

[1] O. Aichholzer, F. Aurenhammer, and F. Hurtado, Sequences of spanning trees and a fixed tree theorem, *Computational Geometry: Theory and Applications*, Vol. 21, Nos. 1–2, 2002, pp. 3–20.

[2] O. Aichholzer and K. Reinhardt, A quadratic distance bound on sliding between crossing-free spanning trees, *Proceedings of the Twentieth European Workshop on Computational Geometry*, Seville, Spain, 2004.

[3] S.G. Akl, *Parallel Computation: Models And Methods*, Prentice Hall, Upper Saddle River, New Jersey, 1997.

[4] S.G. Akl, Discrete steepest descent in real time, *Parallel and Distributed Computing Practices*, Vol. 4, No. 3, 2001, pp. 301 - 317.

[5] S.G. Akl, Parallel real-time computation: Sometimes quality means quantity, *Computing and Informatics*, Vol. 21, No. 5, 2002, pp. 455 - 487.

[6] S.G. Akl, Secure file transfer: A computational analog to the furniture moving paradigm, *Parallel and Distributed Computing Practices*, Vol. 5, No. 2, 2003, pp. 193 - 203.

[7] S.G. Akl, Parallel real-time computation of nonlinear feedback functions, *Parallel Processing Letters*, Vol. 13, No. 1, 2003, pp. 65 - 75.

[8] S.G. Akl, Superlinear performance in real-time parallel computation, *The Journal of Supercomputing*, Vol. 29, 2004, pp. 89 - 111.

[9] S.G. Akl, Coping with uncertainty and stress: A parallel computation approach, to appear in *Parallel and Distributed Computing Practices.*

[10] S.G. Akl and S.D. Bruda, Parallel real-time optimization: Beyond speedup, *Parallel Processing Letters*, Vol. 9, No. 4, December 1999, pp. 499 - 509.

[11] S.G. Akl and S.D. Bruda, Parallel real-time numerical computation: Beyond speedup III, *International Journal of Computers and their Applications*, Special Issue on High Performance Computing Systems, Vol. 7, No. 1, March 2000, pp. 31 - 38.

[12] S.G. Akl and S.D. Bruda, Improving a solution's quality through parallel processing, *The Journal of Supercomputing*, Vol. 19, No. 2, 2001, pp. 219 - 231.

[13] S.G. Akl and W. Yao, Parallel computation helps in real-time control, manuscript, 2003.

[14] S.G. Akl and W. Yao, Parallel computation and measurement uncertainty in nonlinear dynamical systems, Technical Report No. 2003-470, School of Computing, Queen's University, Kingston, Ontario, September 2003, 15 pages.

[15] S.G. Akl and W. Yao, An application of parallel computation to dynamical systems, Technical Report No. 2003-466, School of Computing, Queen's University, Kingston, Ontario, June 2003, 10 pages.

[16] D. Avis, Generating rooted triangulations without repetitions, *Algorithmica*, Vol. 16, 1996, pp. 618–632.

[17] D. Avis and K. Fukuda, Reverse search for enumeration, *Discrete Applied Mathematics*, Vol. 65, 1996, pp. 21–46.

[18] R.S. Barr and B.L. Hickman, Parallel simplex for large pure network problems: Computational testing and sources of speedup, *Operations Research*, 42, 1994, 65–80.

[19] A. Bestavros and V. Fay-Wolfe, Eds., *Real-Time Database and Information Systems*, Kluwer Academic Publishers, Boston, 1997.

[20] S.D. Bruda, *Parallel Real-Time Complexity Theory*, Ph.D. thesis, Department of Computing and Information Science, Queen's University, Kingston, Ontario, April 2002.

[21] S.D. Bruda, and S.G. Akl, The characterization of data-accumulating algorithms, *Theory of Computing Systems*, Vol. 33, January 2000, pp. 85 - 96.

[22] S.D. Bruda, and S.G. Akl, Pursuit and evasion on a ring: An infinite hierarchy for parallel real-time systems, *Theory of Computing Systems*, Vol. 34, No. 6, 2001, pp. 565 - 576.

[23] S.D. Bruda, and S.G. Akl, On the necessity of formal models for real-time parallel computations, *Parallel Processing Letters*, Vol. 11, Nos. 2 & 3, June & September 2001, pp. 353 - 361.

[24] S.D. Bruda, and S.G. Akl, A case study in real-time parallel computation: Correcting algorithms, *Journal of Parallel and Distributed Computing*, Vol. 61, No. 5, May 2001, pp. 688 - 708.

[25] S.D. Bruda, and S.G. Akl, On the relation between parallel real-time computations and logarithmic space, *Proceedings of the Fourteenth Conference on Parallel and Distributed Computing and Systems*, Cambridge, Massachusetts, November 2002, pp. 102 - 107.

[26] S.D. Bruda, and S.G. Akl, On limits on the computational power of data-accumulating algorithms, *Information Processing Letters*, Vol. 86, No. 4, 2003, pp. 221 - 227.

[27] S.D. Bruda, and S.G. Akl, Real-time computation: A formal definition and its applications, *International Journal of Computers and Applications*, Vol. 25, No. 4, 2003, pp. 247 - 257.

[28] D. Clark and M. Bailey, Visualization of height field data with physical models and texture photomapping, *Proceedings of the IEEE Visualization'97 Conference*, Phoenix, Arizona, October 1997, pp. 89–94.

[29] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, McGraw-Hill, New York, 2001.

[30] P. Desnoguès and O. Devilliers, A locally optimal triangulation of the hyperbolic paraboloid, *Proceedings of the Seventh Canadian Conference on Computational Geometry*, Laval, Québec, August 1995, pp. 49–54.

[31] H. Edelsbrunner and N.R. Shah, Incremental topological flipping works for regular triangulations, *Algorithmica*, Vol. 15, 1996, pp. 223–241.

[32] S. Fortune, Voronoi diagrams and Delaunay Triangulations, in: *Computing in Euclidean Geometry*, Second edition, D.Z. Du and F.K. Hwang, Eds., World Scientific Publishing, Singapore, 1995, pp. 225–265.

[33] J. Galtier, F. Hurtado, M. Noy, S. Pérennes, and J. Urrutia, Simultaneous edge flipping in triangulations, *International Journal of Computational Geometry and Applications*, Vol. 13, No.2, 2003, pp. 113–133.

[34] P.L. George and H. Borouchaki, *Triangulation de Delaunay et Maillage – Applications Aux Éléments Finis*, Hermes Science Publishing, London, England, 1997.

[35] F. Guerriero and M. Mancini, A cooperative parallel rollout algorithm for the sequential ordering problem, *Parallel Computing*, Vol. 29, 2003, pp. 663–677.

[36] J.L. Gustafson, Reevaluating Amdahl's law, *Communications of the ACM*, 31, 1988, 532–533.

[37] D.P. Helmbold and C.E. McDowell, Modeling speedup(n) greater than n, *IEEE Transactions on Parallel and Distributed Systems*, 1, 1990, 250–256.

[38] H. Hoppe, Progressive meshes, *Computer Graphics*, Vol. 30, pp. 99–108, 1996.

[39] H. Hoppe, Optimization of mesh locality for transparent vertex cashing, *Computer Graphics*, Vol. 33, pp. 269–276, 1999.

[40] F. Hurtado and M. Noy, Graph of triangulations of a convex polygon and tree of triangulations *Computational Geometry: Theory and Applications*, Vol. 13, No. 3, 1999, pp. 179–188.

[41] F. Hurtado, M. Noy, and J. Urrutia, Flipping edges in triangulations, *Discrete and Computational Geometry*, Vol. 22, 1999, pp. 333–346.

[42] K. Islam, H. Meijer, and S.G. Akl, On the complexity of certain geometric transformations, manuscript in preparation, School of Computing, Queen's University, 2004.

[43] R. Janssen, A note on superlinear speedup, *Parallel Computing*, 4, 1987, 211–213.

[44] B. Joe, Construction of three-dimensional Delaunay triangulations using local transformations, *Computer Aided Geometric Design*, Vol. 8, 1991, pp. 419-453.

[45] I. Kolingerova and J. Kohout, Pessimistic threaded Delaunay triangulation by randomized incremental insertion, *Proceedings of the International Conference on Computer Graphics and Vision (GraphiCon'2000)*, Moscow, Russia, August - September 2000, pp. 76–83.

[46] J. Kohout, Parallel incremental Delaunay triangulation, *Proceedings of the Fifth Central European Seminar on Computer Graphics*, Budmerice, Slovakia, 2001, pp. 85–94.

[47] C.M. Krishna, and K.G. Shin, *Real-Time Systems*, Mc-Graw Hill, New York, 1997.

[48] T.H. Lai and S. Sahni, Anomalies in parallel branch and bound algorithms, *Communications of the ACM*, 27, 1984, 594–602.

[49] C.L. Lawson, Software for C_1 surface interpolation, in: *Mathematical Software III*, J. Rice, Ed., Academic Press, New York, 1977, pp. 161–194.

[50] H.W. Lawson, *Parallel Processing in Industrial Real-Time Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[51] J. Lucas, The rotation graph of binary trees is Hamiltonian, *Journal of Algorithms*, Vol. 9, 1988, pp. 503–535.

[52] R. Mehrotra and E.F. Gehringer, Superlinear speedup through randomized algorithms, *Proceedings of the International Conference on Parallel Processing*, 1985, 291–300.

[53] H. Meijer and D. Rappaport, Simultaneous edge flips for convex subdivisions, manuscript in preparation, School of Computing, Queen's University, 2004.

[54] M. Nagy, and S.G. Akl, Real-time minimum vertex cover for two-terminal series-parallel graphs, *Proceedings of the Thirteenth Conference on Parallel and Distributed Computing and Systems*, Anaheim, California, August 2001, pp. 526 - 534.

[55] M. Nagy, and S.G. Akl, Locating the median of a tree in real time, *Proceedings of the Fourteenth Conference on Parallel and Distributed Computing and Systems*, Cambridge, Massachusetts, November 2002, pp. 108–113.

[56] M. Nagy, and S.G. Akl, Computing nearest neighbors in real time, *Proceedings of the Fifteenth Conference on Parallel and Distributed Computing and Systems*, Marina Del Rey, California, November 2003, pp. 518–524.

[57] N. Nagy, and S.G. Akl, The maximum flow problem: A real-time approach, *Parallel Computing*, Vol. 29, No. 6, 2003, pp. 767–794.

[58] D. Parkinson. Parallel efficiency can be greater than unity. *Parallel Computing*, Vol. 3, 1986, pp. 261–262.

[59] M. Pocchiola and G. Vegter, Topologically sweeping visibility complexes via pseudotriangulations, *Discrete and Computational Geometry*, Vol. 16, No. 4, 1996, pp. 419–453.

[60] W.J. Schroeder, A topology modifying progressive decimation algorithm *Proceedings of the IEEE Visualization'97 Conference*, Phoenix, Arizona, October 1997, pp. 205–213.

[61] D.D. Sleator, R.E. Tarjan, and W.P. Thurston, Rotations distance, triangulations and hyperbolic geometry, *Journal of the American Mathematical Society*, Vol. 1, 1988, pp. 647-682.

[62] A.J. Stewart, Constructing long triangular strips using tunneling and mesh transformations, manuscript, School of Computing, Queen's University, 2004

[63] M. Thorin, *Real-Time Transaction Processing*, Macmillan, London, 1992.

[64] M. Toulouse, T.G. Crainic, and B. Sansó, Systematic behavior of cooperative search algorithms, *Parallel Computing*, Vol. 30, 2004, pp. 57–79.

[65] S. Zilberstein, Using anytime algorithms in intelligent systems, *AI Magazine*, Fall 1996, pp. 73–83.