1999

Embeddability of pseudoline arrangements and point configurations to Eucliean plane

Javid Huseynov

Rochester Institute of Technology
Department of Computer Science

Embeddability of Pseudoline Arrangements and Point Configurations to Euclidean Plane

by
Javid Huseynov

A thesis, submitted to
the Faculty of the Department of Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by:

Prof. Stanisław Radziszowski

Prof. Peter Anderson

Prof. Andrew Kitchen

July 26, 1999

MS Thesis Report Copyright Release

I, hereby signed Javid Huseynov, do permit to reproduce this report for any non-commercial or non-profit use.

Signed by:

Javid Huseynov

July 28, 1999

Thesis Abstract

The present thesis explores embeddability (realizability) properties of pseudoline arrangements, perhaps, the most important mathematical structures in computational geometry. The underlying theme is the use of combinatorial representation of arrangements vs. usual geometric representation to approach the embeddability problem. The combinatorial representation chosen is that of counterclockwise systems (CC-systems) introduced by D.E. Knuth in 1992. CC systems were defined as sets of ordered triples of points that incorporate and obey counterclockwise relations on points in the plane.

If a CC system can arise from actual points in the plane, it is called *realizable*. Not all CC-systems are realizable in plane, since they are defined by axioms that involve at most five points. The most famous nonrealizable configuration on nine points corresponds to so-called theorem of Pappus, which is a legal CC-system. Any heuristic solution to realizability question is proven to be NP-hard.

The primary goal of this thesis is to consider embeddings of CC-systems to the Euclidean plane and to exploit properties coherent to embeddable systems. As a major effort in this task we have developed and implemented algorithms (brute force as well as breadth-first search) for finding the convex hull of a CC-system and geometrically testing the realizability of the system using linear programming methods.

Recently, Goodman, Pollack, Wenger, and Zamfirescu[1] [2] have proved two conjectures of Grünbaum, showing that any arrangement of pseudolines on 8 points in the plane can be embedded into a projective plane, and that there exists a universal topological projective plane in which every arrangement of pseudolines is stretchable (realizable). Thus, we have conducted our realizability tests on systems arising from 9 or more points. Pappus system mentioned above and other non-realizable extensions of it on more than 9 points were studied as well.

We have also developed a graphical interface (Java application) for creating, dynamically updating, visualizing and storing CC-systems. Visual geometric embedding of a particular system point-by-point is another feature provided by this interface.

[1]GOODMAN J.E., POLLACK R., WENGER R. and ZAMFIRESCU T. *Every Arrangement Extends To A Spread*, in *Proceedings of the Third Annual Canadian Conference on Computational Geometry*, 1991, pp. 191-194.

[2]GOODMAN J.E., POLLACK R., WENGER R. and ZAMFIRESCU T. *There Is A Universal Topological Plane*, in *Proceedings of the Eighth Annual ACM Symposium on Computational Geometry*,Berlin, June 1992, pp. 171-176

Contents

1 Acknowledgments **3**

2 Introduction and background **4**
2.1 Pseudoline arrangements 5
2.2 Point configurations and CC-systems 6
2.2.1 Definitions and axioms 6
2.2.2 The definition of tournaments; transitive and vortex-free tournaments. 8
2.2.3 Convex hull 11
2.2.4 Algorithms for finding convex hulls 12
2.2.5 Complexity of problems involving CC-systems 13
2.3 Oriented matroids and their relation to CC-systems 15

3 Realizability of pseudoline arrangements and point configurations **17**
3.1 Previous research 17
3.2 The Pappus and the Desargues configurations as examples of nonrealizable CC-systems 18
3.3 Our approach 19

4 Algorithms and software system description **20**
4.1 Convex hull algorithms 20
4.2 Realizability solver 22
4.2.1 Algorithms and their application 22
4.2.2 Basic testing of the realizability solver 25
4.2.3 Testing results on 8-point CC-systems 26
4.2.4 Testing results on 9-point CC-systems 26
4.2.5 Testing results on 10-point systems with complementary symmetry 28
4.3 Graphical user interface for geometric embedding 28
4.4 Functional specification 30
4.4.1 Realizability solver functions (in C) 30
4.4.2 CC-system generator 30
4.5 User's manual 31
4.5.1 Realizability solver 31
4.5.2 Graphical user interface 32
4.5.3 CC-system generator 32

5 Conclusion

1 Acknowledgments

I am grateful to my MS thesis advisor, Prof. Stanisław Radziszowski, who guided me through my thesis research. I would also like to thank Prof. Peter Anderson and Prof. Andrew Kitchen for agreeing to be my readers.

2 Introduction and background

Forgive me should I cause offence
and give me leave to make this point.
– Nizami

Mathematicians in the nineteenth century showed possibility of creating consistent geometries in which Euclid's Parallel Postulate was no longer true. Absence of parallels leads to spherical, or elliptic, geometry; abundance of parallels leads to hyperbolic geometry. By mid-century the English mathematician Arthur Cayley had constructed analytic models of these three geometries that had a common descent from projective geometry, which one may think of as the formalization of the renaissance theory of perspective.

The present thesis explores embeddability (realizability) properties of pseudoline arrangements and point configurations. Realizable point configurations are the ones that can arise from points in Euclidean plane. The underlying theme is the use of combinatorial representation of point configurations vs. usual geometric representation to approach the realizability problem and to explore properties coherent to realizable systems.

The combinatorial representation chosen is that of *counterclockwise systems (CC-systems)*. The latter were introduced by Donald Knuth in 1992, and his work "Axioms and Hulls" is used as one of the main references in our thesis. CC-systems were defined as sets of ordered triples of points that incorporate and obey counterclockwise relations on points in the plane, formalized by five axioms described in section 2.2.1 of the present thesis.

CC-systems are also known to be equivalent to configurations arisen in another part of mathematics, namely, uniform acyclic oriented matroids of rank 3. There is a two-to-one correspondence between CC-systems on a set of labeled points and all such oriented matroids on the same set. Additional information on theory of oriented matroids and their equivalence to CC-systems is given in section 2.3.

Unfortunately, axioms, although simple, can lead to issues that are very difficult to decide; realizability is one of them. In fact, any heuristic solution to the problem of realizability would be NP-hard [35], and our work was an attempt to test the realizability in finite time using geometric methods. For this purpose, we studied and implemented the convex hull algorithms for CC-systems, axiom testing, and applied linear programming methods for geometric realizability solver. These algorithms are described in detail in section 4.

The geometric representations of point configurations and line arrangements were successfully implemented in the course of present thesis. A graphical user interface using Java was developed to create, edit, visualize and store CC-systems. This graphical user interface is described in subsection 4.3 of present thesis.

2.1 Pseudoline arrangements

Pseudoline arrangements are, perhaps, the most important mathematical structures in computational geometry. The *arrangement* $\mathcal{A}(\mathcal{L})$ of a finite set $\mathcal{L}$ of lines is the partition of the plane formed by $\mathcal{L}$, whose *vertices* are intersection points of lines in $\mathcal{L}$, *edges* are maximal connected (open) portions of lines in $\mathcal{L}$ not containing any vertex, and *faces* are maximal connected (open) portions of the plane not meeting any vertex or edge [1]. The notion of arrangement can be easily generalized to curves in the plane, or to hyperplanes or hypersurfaces in higher dimensions. The *combinatorial complexity* of an arrangement in the plane is the number of its vertices, edges and faces, and in higher dimensions it is the total number of faces in all dimensions.

A *pseudoline* is a simple curve in the plane that goes to infinity in two directions [35]. A *collection of pseudolines* is a set of pseudolines such that any two members of the set intersect at most once, and cross if they intersect. A *pseudoline arrangement* is the partition of the plane induced by a collection of pseudolines. A line or pseudoline arrangement is *uniform* if no three lines intersect in a point and no two lines are parallel.

Arrangements of lines and pseudolines have been studied from various mathematical points of view for a very long time. The first systematic exposition of arrangements is due to B. Grünbaum. Arrangements in higher dimensions have received considerably less attention, and not much is known about them except for some basic results. Edelsbrunner [7] also provides an extensive survey of known results in this area.

The algorithmic study of arrangements, however, has started only recently, primarily because of their significance in many fundamental geometric problems. The arrangement of a set of n lines in the plane can be computed in time $O(n^2 \log n)$ using a line sweep approach; this bound has been improved to $O(n^2)$ by Edelsbrunner. In fact, Edelsbrunner and Guibas [10] presented another $O(n^2)$ algorithm to construct the arrangement of a given set of n lines using topological sweep, which requires only $O(n)$ working storage.

Stretchability is an arrangement of pseudolines such that lines can

all be made straight without changing the topological configuration of the arrangement itself. Shor [35] presented a similar definition for stretchability, claiming that a pseudoline arrangement is *stretchable* (or *realizable)* if there is an arrangement of lines with the same combinatorial structure. The stretchability concept in line arrangements is the same as that of realizability of CC-systems. In fact, many sources refer to stretchable line arrangements as realizable. Line arrangements associated with realizable CC-systems are *stretchable* in the Euclidean plane.

Goodman and Pollack [20] have proved two conjectures of Grünbaum, showing that any arrangement of pseudolines formed by up to 8 points in the plane can be embedded into a projective plane. Later, along with Wenger and Zamfirescu [23], they also proved that there exists a universal topological projective plane in which every arrangement of pseudolines is stretchable. By Folkman and Lawrence's theorem, this plane contains every finite (simple) oriented rank 3 matroid.

In our thesis we mainly concentrated on studying realizable CC-systems (point configurations). Realizable CC-systems are equivalent to stretchable line arrangements and realizable oriented matroids of rank 3.

2.2 Point configurations and CC-systems

2.2.1 Definitions and axioms

As mentioned in the introduction, CC-systems were defined as sets of ordered triples of points that incorporate and obey counterclockwise relations on five points in the Euclidean plane. Formally, a CC-system is a system of three point counterclockwise predicates[1]. Counterclockwise predicate on points p, q, r with Cartesian coordinates $(x_p, y_p), (x_q, y_q), (x_r, y_r)$ corresponds to the sign of the determinant [29]:

$$pqr \iff \begin{vmatrix} x_p & y_p & 1 \\ x_q & y_q & 1 \\ x_r & y_r & 1 \end{vmatrix} > 0$$

Counterclockwise predicates on three points satisfy the following axioms:

- **Axiom 1**. Cyclic symmetry: $pqr \Rightarrow qrp$
- **Axiom 2**. Antisymmetry: $pqr \Rightarrow \neg\, prq$

[1]The *counterclockwise* predicate pqr is true when the circle through points (p, q, r) is traversed counterclockwise when we encounter the points in cyclic order $p, q, r, p, \ldots$

- **Axiom 3**. Nondegeneracy: $pqr \bigvee prq$

  In this axiom we assume that no three points are collinear; collinear points are described by the relation $|pqr| = 0$, which violates the counterclockwise predicate defined by the sign of the determinant $|pqr| > 0$.

- **Axiom 4**. Interiority: $tqr \wedge ptr \wedge pqt \Rightarrow pqr$.

  Obviously, if point t lies to the left of directed lines pq, qr, and rp, then t is inside the triangle pqr, thus:

  $|pqr| = |tqr| + |ptr| + |pqt|$.

- **Axiom 5**. Transitivity: $tsp \wedge tsq \wedge tsr \wedge tpq \wedge tqr \Rightarrow tpr$.

  In this axiom, the first three hypotheses state that the points p, q, r are in the halfplane to the left of ts, the last two hypotheses state that q is to the left of tp and r is left of tq. Thus, r is left of tp, and t is a convex combination of p, q, r. This means that determinant $|tst|$ is a convex combination of the positive determinants $|tsp|, |tsq|, |tsr|$, which is impossible because $|tst| = 0$ [29].

  Similarly, a dual transitivity axiom can be deduced:

- **Axiom 5′**. Dual Transitivity: $stp \wedge stq \wedge str \wedge tpq \wedge tqr \Rightarrow tpr$.

  Let's assume that this axiom fails, so that

  $$stp \wedge stq \wedge str \wedge tpq \wedge tqr \wedge trp$$

  We can prove that

  $$spq \Longrightarrow srp.$$

  It certainly holds if pqr is true, since Axiom 5 tells us that $pqs \wedge pqt \wedge pqr \wedge pst \wedge ptr \Longrightarrow psr$. Assuming $spq \wedge spr \wedge prq$, we must have rqs, otherwise Axiom 5 would say that $rsq \wedge rsp \wedge rst \wedge rqp \wedge rpt \Longrightarrow rqt$. But rqs causes another problem, since Axiom 5 also carries the implication $qsr \wedge qst \wedge qsp \wedge qrt \wedge qtp \Longrightarrow qrp$.

  Thus, $spq \Longrightarrow srp$ and similarly:

  $$spq \Longrightarrow srp \Longrightarrow sqr \Longrightarrow spq$$

  and we have either $spq \wedge srp \wedge sqr$ or $sqp \wedge spr \wedge srq$. Both of these arguments contradict Axiom 5, since

  $$stp \wedge stq \wedge str \wedge spq \wedge sqr \Longrightarrow spr$$
  $$stq \wedge stp \wedge str \wedge sqp \wedge spr \Longrightarrow sqr.$$

So, we simply proved that Axioms 1, 2, 3 and 5 imply 5′. If we complement the value of each triple containing s, we can see that 1, 2, 3, and 5′ imply 5.

Knuth defined his axioms on 5-point systems obeyed by the counterclockwise relation on points. In our work, we extended the notion of CC-system and its axioms to larger point systems. The axioms are simple and can be generally applied to system on any number of points. However, as the number of points in the system grows, the question of *realizability* of this system in Euclidean plane takes an important position. Moreover, complexity issues also become more sensitive with realizing larger point systems.

A realizable system is the one that can arise from coordinates in the Euclidean plane. The axioms presented are not strong enough to prove that a given system is realizable. Knuth actually showed the possibility of constructing a CC-system that satisfies all the axioms but cannot arise from actual points in the plane. Examples of such systems are the Pappus and the Desargues configurations presented below in our subsection on realizability. Detailed information on realizability research is also given in section 3.

Additional research and algorithms are needed to prove that the given point configuration is a realizable CC-system. Definition and implementation of such algorithms and other ways of solving the realizability question form the basis of our thesis.

2.2.2 The definition of tournaments; transitive and vortex-free tournaments.

For each point p, a directed graph (tournament) with arcs $q \to r$ can be formed, if and only if pqr holds. This shortly means that for every point p in a CC-system the relationship of the remaining points q, r can be described using simple directed graphs, or tournaments. These tournaments arise from counterclockwise relations (predicate) of triples in a CC-system.

If we consider a set of triples on $\{a, b, c, d, e\}$.

$$abc, dab, dbc, dca, eab, ebc, eca, ead, ebd, ecd.$$

corresponding tournaments for every point in the set would be as follows:

$$a: \begin{matrix} b \to c \\ \downarrow \nearrow \uparrow \\ d \to e \end{matrix} \quad , \quad b: \begin{matrix} c \to a \\ \downarrow \nearrow \uparrow \\ d \to e \end{matrix} \quad , \quad c: \begin{matrix} a \to b \\ \downarrow \nearrow \uparrow \\ d \to e \end{matrix} \quad , \quad d: \begin{matrix} a \to b \\ \uparrow \swarrow \uparrow \\ c \leftarrow e \end{matrix} \quad , \quad e: \begin{matrix} a \to b \\ \uparrow \searrow \downarrow \\ c \to d \end{matrix}$$

Indeed, either $q \to r$ or $r \to q$ is true for pair of distinct vertices q and r. A system of triples satisfying axioms 2 and 3 on an n-element set

corresponds to a set of n tournaments, and Axiom 1 concludes that $q \to r$ appears in the tournament for p iff $r \to p$ and $p \to q$ appear respectively in tournaments for q and r.

The definition of tournaments within the framework of CC-systems is very important in our thesis. This allowed us to apply certain graph theory algorithms to analyze CC-systems. In particular, the application of Breadth-First Search (BFS) method for CC-systems and other algorithms for convex hull removal makes use of tournaments of points in these systems. We defined a tournament as a matrix arising from directed edges connecting the points in this digraph.

A tournament with no 3-point cycles is called *transitive*. Transitive tournaments take an important role in our approach to realizability of CC-system. As it will be shown in further sections, points that correspond to such tournaments are extreme and form the convex hull of a CC-system. Transitivity factor of the tournaments draws from Axiom 4 (interiority) of CC-systems as indicated below. We applied both of these characteristics in implementing our algorithms for convex hull removal and realizability tester.

Axiom 4 claims that any 3-cycle $p \to q \to r \to p$ in a tournament for t must correspond to a triple pqr and such relationship also affects tournaments of p, q, and r, which do not involve point t. According to Axiom 5, the tournament for t must not contain four vertices forming a 3-cycle pqr and a source s:

$$t: \quad \begin{array}{c} s \to p \\ \downarrow \diagdown\!\!\diagup \uparrow \\ q \to r \end{array} \quad .$$

Axiom 5' claims that no derived tournament should contain a 3-cycle and a sink. Thus, we notice an independence of axioms, particularly, Axioms 5 and 5' cannot be derived based on Axioms 1–4. Therefore, only all five axioms together are strong enough to define a CC-system.

Vortex-free tournament is a directed graph in which either $p \to q$ or $q \to p$ holds for all $p \neq q$, such that no four points form a vortex (a subtournament containing a 3-cycle with source or sink) [29]. To better describe vortex-free tournaments, Knuth introduced the idea of *signed points,* namely $a_1, \ldots, a_n$ and their complements $\bar{a}_1, \ldots, \bar{a}_n$. The original points are said to be positive, and their complements are said to be negative. The operation of changing sign is called *negation*, and we define

$$\bar{\bar{a}} = a \qquad \text{and} \qquad |a| = |\bar{a}| = a\,.$$

The negation of a signed point reverses the direction of graph arcs touching it.

Lemma (Knuth). A tournament is vortex-free iff it can be obtained from a transitive tournament by negating a subset of its points.

Proof. Negating one point of an in-vortex (sink) produces an out-vortex (source). Thus, negation preserves vortex-freeness, and subsequently, transitive tournament by repeated negation must be vortex-free.

Suppose we have a point a in vortex-free tournament. Negating every point p so that $p \to a$ would yield a vortex-free transitive tournament such that $a \to p$ for every $p \neq a$. □

Corollary (Knuth). A tournament on n points is vortex-free if and only if there is a string $\alpha_1\alpha_2\ldots\alpha_n$ containing each point or its complement, such that

$$\alpha_j \to \alpha_k \qquad \text{for} \quad 1 \le j < k \le n\,.$$

Moreover, it is possible to construct such a string by examining the direction of only $O(n \log n)$ arcs.

Proof (Knuth). The condition above is just a rephrasing of the lemma, in terms of signed points.

To construct a suitable string $\alpha_1\alpha_2\ldots\alpha_n$, choose α_1 to be any signed point. Then if a partial string $\alpha_1\ldots\alpha_k$ has been constructed representing a vortex-free subtournament on k points for some k with $1 \le k < n$, let p be any point distinct from $|\alpha_1|, \ldots, |\alpha_k|$, and let $\alpha = p$ or $\bar{p}$ according as $\alpha_1 \to p$ or $p \to \alpha_1$. We know from the lemma that there exists j in the range $1 \le j \le k$ such that $\alpha_i \to \alpha$ for $1 \le i \le j$ and $\alpha \to \alpha_i$ for $j < i \le k$; the value of j can be determined by using binary search to examine the direction of at most $\lceil \lg k \rceil$ arcs. This yields a string $\alpha'_1\ldots\alpha'_{k+1} = \alpha_1\ldots\alpha_j\,\alpha\,\alpha_{j+1}\ldots\alpha_k$ that represents a subtournament of $k+1$ points; and the process can therefore continue with k replaced by $k+1$ and $\alpha_1\ldots\alpha_k$ replaced by $\alpha'_1\ldots\alpha'_{k+1}$, until $k = n$. □

CC-systems are combinatorial representations that correspond to arrangements of points in the plane and satisfy all five axioms. Thus, every point p in a CC-system on n points has an associated vortex-free tournament defined by a string $\alpha_1\alpha_2\ldots\alpha_{n-1}$ on the remaining points. This string is, in fact, easy to interpret: it represents the order in which the remaining points are encountered when a straight line through p rotates counterclockwise through 180°. The positive elements of $\alpha_1\alpha_2\ldots\alpha_{n-1}$ are the points to the left of the initial position of this sweep line; the negative elements are those to the right.

2.2.3 Convex hull

The convex hull of a CC-system is a set of all ordered pairs ts of distinct points such that tsp holds for all $p \notin \{s,t\}$. By Axiom 5 if ts is in the convex hull, the tournament of point t is transitive. The points in convex hull can also be called as *extreme points* of a CC-system, and these points are associated with transitive tournaments. In [29] Knuth proved the following lemma.

Lemma *(Knuth).* A convex hull of a CC-system on $n \geq 2$ points consists of ordered pairs that form a cycle,

$$t_1\,t_2, t_2\,t_3, \ldots, t_{m-1}\,t_m, t_m\,t_1, \quad \text{where} \quad m \geq 2$$

The proof of this lemma uses both Axiom 4 and Axiom 5, since if points p,q,r,s,t violate Axiom 4, they define a system of triples with no convex hull, which is impossible. On the other hand, if p,q,r,s,t violate Axiom 5, they define a system with ts in the convex hull but not pt, qt, rt, or st. Therefore, Axioms 4 and 5 are necessary and sufficient to obtain a ternary relation in which all subsets have a convex hull satisfying the lemma above, assuming that Axioms 1 – 3 hold.

If $\{t_1t_2, \ldots, t_{m-1}t_m, t_mt_1\}$ is the convex hull of a CC-system, we can prove that

$$t_it_jt_k \quad \text{whenever} \quad i < j < k$$

This is assumed by the definition of convex hull when $j = i+1$ or $j = k-1$. Otherwise, it can be proven by induction on $k-i$ that $t_{i+1}\,t_j\,t_k$ and $t_i\,t_j\,t_{k+1}$ are true; the tournament for t_i would then contain a vortex $t_{k-1} \to t_k \to t_j \to t_{k-1}$ out of t_{i+1} if we had $t_i\,t_k\,t_j$. Thus, the cyclic sequence of extreme points $(t_1, \ldots, t_m)$ forms an m-gon. The counterclockwise relation $t_i\,t_j\,t_k$ holds if and only if

$$i < j < k \quad \text{or} \quad j < k < i \quad \text{or} \quad k < i < j$$

Theorem *(Knuth).* Suppose points $(t_1, \ldots, t_m)$ of a CC-system form an m-gon, and let p be another point. If p lies outside the m-gon, say $pt_1\,t_m$ then there exist indices $1 \leq j \leq l < m$ such that

$$pt_k\,t_{k+1} \quad \text{if and only if} \quad j \leq k \leq l$$

On the other hand, if $p \in \Delta t_it_jt_k$ for some $i < j < k$, we have pt_mt_1 and pt_kt_{k+1} for $1 \leq k < m$.

Proof. If pt_1t_m, the ordered pairs ts with $t \neq p$ and $s \neq p$ in the convex hull of $\{p, t_1, \ldots, t_m\}$ are precisely the pairs t_kt_{k+1} such that pt_kt_{k+1} holds. Since the convex hull is a cycle, these pairs must be consecutive and the full convex hull must include also pt_j and $t_{l+1}p$, where j and l are defined by conditions above.

If $p \in \Delta t_it_jt_k$, the tournament for p includes the cycle $t_i \rightarrow t_j \rightarrow t_k \rightarrow t_i$, so p is not an extreme point. The convex hull of $p, t_1, \ldots, t_m$ must therefore be a cycle that does not involve p, and the only suitable cycle is $\{t_1t_2, \ldots, t_{m-1}t_m, t_mt_1\}$ $\square$.

The convex hull of a CC-system is a plane formed by points of the cycle. Thus it's possible to realize the remaining (non-hull) points of the realizable system in this plane (obviously Euclidean). Finding the convex hull of the system also allows us to apply formal geometric rules to points (assigning coordinates, using linear programming techniques on halfplanes, etc.). The approach we have taken to the realizability question is based on the convex hull of the system, and described in detail in section 3.

2.2.4 Algorithms for finding convex hulls

Based on the theorem from the previous section, Knuth introduced an efficient incremental algorithm to find the convex hull of any CC-system. Suppose we have a CC-system of points numbered $\{a_1, \ldots, a_N\}$ in any order, where $N \geq 2$; and the convex hull $(t_1, t_2, \ldots, t_m)$ is represented by separate t_1 and the ordered list $(t_2, \ldots, t_m)$ in a binary search tree. Initially $m = n = 2$, $t_1 = a_1$, and $t_2 = a_2$. If $n < N$, increase n by 1, set $p = a_n$, and update the convex hull as follows:

Case 1, pt_1t_m.
  set $j = 1, 2, \ldots$ **until** $j = m - 1$ **or** pt_jt_{j+1}
  set $l = m - 1, m - 2, \ldots,$ **until** $l = j$ **or** pt_1t_{l+1}.
  delete $\{t_{l+2}, \ldots, t_m\}$ **from the tree.**
  if $(j == 1)$
    insert p **at the right of the tree**
  else
    delete $\{t_2, \ldots, t_{j-1}\}$ **from the tree and replace** t_1 **by** p.

Case 2, pt_mt_1.
  let t_k **be the root of the tree**
  start a tree search
  if $(k == m \;||\; t_1pt_k)$

decrease k so that the new t_k is the left child of old t_k
else
increase k so that the new t_k is the right child of old t_k

It follows that t_1pt_j holds iff $j \geq k$. If $k > 2$ and $pt_{k-1}t_k$, we have $p \in \Delta t_1 t_{k-1} t_k$ and the convex hull does not need to be updated. Otherwise we have discovered that pt_kt_{k-1}, hence p lies outside the m-gon $(t_1, \ldots, t_m)$. Set $j = k-1, k-2, \ldots$, until $j \leq 2$ or $pt_{j-1}t_j$. Set $l = k, k+1, \ldots$, until $l = m$ or pt_lt_{l+1}. Then delete $\{t_{j+1}, \ldots, t_{l-1}\}$ from the tree and insert p between t_j and t_l.

If the binary search tree is maintained as a balanced tree of some kind, the total running time of the algorithm does not exceed $O(N \log N)$, since the total number of tree operations is at most N searches, N insertions, and N deletions.

In [29] Knuth also presented another incremental algorithm where the current convex hull is maintained in a doubly linked circular list $(t_1, \ldots, t_m)$, and the counterclockwise tests are controlled by a binary branching structure from which no deletions need to be made. While the first algorithm takes $O(N \log N)$ time on any given N-point CC-system, the second one takes the same time on any given system that is input in random order.

In our thesis we have implemented convex hull algorithms based on breadth-first search (similar to the first algorithm by Knuth) and a brute-force triple search algorithm, which finds all cycles in a tournament (digraph) of any given point in the system. Implementation of these algorithms is described in depth in section 4 of this thesis.

2.2.5 Complexity of problems involving CC-systems

In resolving the problems involving CC-systems, we deal with partial information-to know when certain sets of triples imply others. Thus, there is a need for efficient ways to solve decision problems involving vertices of a CC-system. In many cases, axioms may lead to situations that are very difficult to decide. Knuth proves that the decision on whether or not given values of fewer than $\binom{n}{3}$ triples can be completed to a full set of values that satisfy Axioms 1 – 5 is NP-complete. In fact, Knuth even goes further by proving the following:

Corollary. The problem of deciding whether the values of a given set of triples are consistent with Axioms 1 – 5 is NP-complete.

Proof (Knuth). Given any vortex-free tournament on $a_1, \ldots, a_n$, let a_0 be another point, and define $a_0 \to a_k$ for all k. Then let $a_ia_ja_k$ be true if

and only if at least two of the relations $a_i \to a_j$, $a_j \to a_k$, $a_k \to a_i$ are true. This rule defines a system of triples in which the original tournament is the tournament associated with a_0. We claim that it is, in fact, a CC-system. Axioms 1 – 3 certainly hold.

Suppose $t \in \Delta pqr$, meaning that we have tpq, tqr, and trp. Then the inequalities

$$[t \to p] + [p \to q] + [q \to t] \geq 2$$
$$[t \to q] + [q \to r] + [r \to t] \geq 2$$
$$[t \to r] + [r \to p] + [p \to t] \geq 2$$

can be added to give

$$[p \to q] + [q \to r] + [r \to p] + 3 \geq 6.$$

Hence $p \to q \to r \to p$, and Axiom 4 has been verified.

Moreover, the given system has the property that when when tpq and $p \to q$ then $t \to p \iff t \to q$. Therefore if $t \in \Delta pqr$ we have either

$$t \to p \ \wedge \ t \to q \ \wedge \ t \to r$$

or

$$p \to t \ \wedge \ q \to t \ \wedge \ r \to t.$$

But the above is impossible in a vortex-free tournament; so $t \in \Delta pqr$ can occur only when t is the special point a_0. Axiom 5 now follows immediately.

In the proof above $a_i a_j a_k$ is true as defined if and only if the points (a_i, a_j, a_k) form a counterclockwise triple in the complex plane. This argument shows that it is NP-hard to determine whether or not a given set of triples is part of a *realizable* CC-system. Thus, the problem of realizability in general can be considered as NP-hard.

The question of realizability has been studied in depth by Bokowski and Sturmfels [3]. Peter Shor in [35] also proved that stretchability (realizability) problem is NP-hard. He showed NP-hardness by reducing a variant of the NP-complete 3-SAT problem to the stretchability problem. The 3-SAT problem as defined by Garey and Johnson is:

Given a Boolean expression in conjunctive normal form containing only three variables in each clause, is there an assignment of the variables making the expression true?

Shor used a variant of 3-SAT called *monotone* 3-SAT, where all clauses are defined either as positive (containing only non-negated variables) and negative (only negated variables). Given such a 3-SAT formula, he further constructed a pseudoline arrangement which is stretchable if and only if the formula is satisfiable. In the pseudoline arrangement, clauses correspond to modified Pappus configurations, variables correspond to certain triples of points, and variables are linked to clauses by modified Desargues configurations.

Moreover, Shor also showed that there exists a symmetrical pseudoline arrangement which is stretchable, but which is not stretchable to a symmetrical line arrangement. Mnëv [31] has shown that the topology of the realization space of a pseudoline arrangement can be the same as the topology of any semialgebraic variety (solution space of a set of polynomial inequalities and equations over the reals). Mnëv's result also implies that determining the stretchability of a pseudoline arrangement is equivalent to the existential theory of the reals.

2.3 Oriented matroids and their relation to CC-systems

In 1935 H. Whitney introduced the concept of a matroid, which unifies many configurations studied in pure and applied mathematics—in particular, in algebra, geometry, combinatorics, and optimization theory. Ordinary matroids may be viewed as an abstraction of finite geometric configurations which are embedded into some vector space over a field. In 1978, oriented matroids were introduced by R. G. Bland, M. Las Vergans, J. Folkman, and J. Lawrence as combinatorial abstractions of finite geometric configurations in vector spaces over some ordered field. Since oriented matroids, as well as ordinary matroids, are important in many areas—as in algebraic and computational geometry combinatorics, topology, operations research, and chemistry—researchers in various fields were led to questions concerning oriented matroids.

According to Knuth, CC-systems are equivalent to other configurations that have arisen in context of computational geometry, namely, uniform acyclic oriented matroids of rank 3. There is two-to-one correspondence between CC-systems on a set of labeled points and all oriented matroids defined on the same set.

The axioms for matroids are quite different from axioms for CC-systems. A uniform acyclic matroid of rank 3 is a collection of *circuits,* which are sets of signed points p, q, r, s with the following properties:

- **M1.** If $\{p, q, r, s\}$ is a circuit, the absolute values $|p|,|q|,|r|,|s|$ are distinct.

- **M2.** If $\{a, b, c, d\}$ is any set of four unsigned points, there is a circuit $\{p, q, r, s\}$ with $|p| = a$, $|q| = b$, $|r| = c$, $|s| = d$.

- **M3.** If $C = \{p, q, r, s\}$ is a circuit, so is the negated set $\bar{C} = \{\bar{p}, \bar{q}, \bar{r}, \bar{s}\}$

- **M4.** If $C = \{p, q, r, s\}$ and $C' = \{\bar{p}, q', r', s'\}$ are any circuits with $C' \neq \bar{C}$, then there is at least one circuit C' contained in the set $\{q, r, s, q', r', s'\}$.

An oriented matroid is called acyclic if every circuit contains at least one negative point (or one positive point by axiom M3).

The connection between oriented matroids and CC-systems is quite simple. The circuits $\{p, q, r, s\}$ of an oriented matroid M are precisely the sets of signed points such that the signed triples of CC-system satisfy

$$sqp = srq = spr = pqr,$$

where $=$ denotes equality of boolean values.

The theory of oriented matroids is extensive and deals with more general systems than these. A major breakthrough in the field of matroids was the development of so called *universality theorems* that conclude certain realizability rules which apply for all point configurations (or CC-systems).

In [32] Richter-Gebert defines oriented matroids as combinatorial models of vector configurations in the real linear space R^n. An oriented matroid contains all relevant data about the relative positions of vectors in a configuration. There are two kinds of oriented matroids: *realizable* oriented matroids (that come from concrete vector configurations), and *nonrealizable* oriented matroids (which can be considered as combinatorial or topological generalizations of point configurations). One of the universality theorems proven by Richter-Gebert states:

There is a polynomial algorithm that takes as input a system Ω of polynomial equations and strict inequalities with integer coefficients and produces an oriented matroid $M(\Omega)$ such that the realizability problem for $M(\Omega)$ is equivalent to the solvability problem of Ω.

In a sense, we have taken the same approach in realizing CC-systems. Every arrangement in the tournament of the point can be presented as a

strict inequality, and area that satisfies set of inequalities for all arrangements in the tournament is the realizability area of the point in question. Thus the application of linear programming techniques in resolving the realizability question is quite appropriate in our case.

3 Realizability of pseudoline arrangements and point configurations

3.1 Previous research

If a CC-system can arise from actual points in the plane, it is called *realizable* (or *embeddable).* Not all CC-systems are realizable in plane, since they are defined by axioms that involve at most five points. As mentioned before, in section on pseudoline arrangements, the definition of realizability in CC-systems is equivalent to stretchability of pseudoline arrangements. Substantial research has been done on the question of deciding when an arrangement of pseudolines is stretchable to an isomorphic arrangement of straight lines; this is equivalent to deciding when a CC-system is realizable by points in the Cartesian plane [29]. Levi [30] observed that non-stretchable arrangements exist, if we allow three pseudolines to intersect at the same point; such arrangements correspond to nonrealizable oriented matroids of rank 3 in which some 3-element subsets are dependent. Ringel [34] showed 9 pseudolines in a nonstretchable *simple* arrangement; i.e., Ringel's example is equivalent to a nonrealizable CC-system on 9 points. Goodman and Pollack [20] proved that all CC-systems on 8 or fewer points are realizable; but they proved a few years later [21] that almost all CC-systems on n points are unrealizable, in the limit as $n \to \infty$. Bokowski and Sturmfels [3] constructed nonrealizable CC-systems on 10, 12, 14, ... points with the property that a realizable system is obtained when any point is deleted. Hence there can be no finite set of axioms analogous to Axioms 1 – 5 that characterize precisely the realizable systems.

As Knuth indicates in [29] and it was mentioned in our section on complexity, the problem of deciding whether or not a given set of triples is part of a realizable CC-system is NP-hard [35]. Knuth actually conjectures that this problem may not be even in NP, although Tarski's decision procedure [33] shows that realizability can be tested in finite time.

3.2 The Pappus and the Desargues configurations as examples of nonrealizable CC-systems

The most famous nonrealizable configuration on nine points corresponds to so-called theorem of Pappus, which is a legal CC-system.

Theorem of Pappus. If A, B and C are three distinct points on one line and if A', B' and C' are three different distinct points on a second line, then the intersections of lines AC' and CA', lines AB' and BA', and lines BC' and CB' are collinear.

The figure generated by Pappus' Theorem involves nine distinct points lying by threes on three distinct lines. The total of the nine points and nine lines can be studied independently as a finite geometry. However, it is impossible to realize this configuration as a valid CC-system on a Euclidean plane without bending the lines in the system.

$$|pqw||rpv||qry||prx||quz| + |rpv||qru||rpz||qpy||qwx|+$$

$$|qru||pqw||rpz||prx||qvy| + |qru||pqw||rpz||qpy||rvx|+$$

$$|pqw||rpv||pqx||rqz||ruy| + |rpv||qru||pqx||qpy||rwz|+$$

$$|rpv||qru||pqx||rqz||pwy| + |qru||pqw||qry||prx||pvz|+$$

$$|pqw||rpv||qry||rqz||pux| = 0,$$

The identity described above implies that counterclockwise triples of any nine points in the plane must satisfy the following axiom:

$$\neg((pqw \oplus rpv \oplus qry \oplus prx \oplus quz) \wedge (rpv \oplus qru \oplus rpz \oplus qpy \oplus qwx)$$

$$\wedge \ldots \wedge (pqw \oplus rpv \oplus qry \oplus rqz \oplus pux)),$$

For if each of the parenthesized clauses in the above axiom were true, each term on the left of the previous identity would be positive, and the sum could not be zero [29]. It is possible to construct a CC-system such that all counterclockwise triples $pqw, rpv, qry, \ldots, pux$ are true. Thus CC-systems can violate the above axiom, they are more general than realizable systems. In fact, we cannot expect five axioms of CC-systems to be strong enough to deduce the 9-point theorem of Pappus, which states that if eight of the triples of points $pux, pvx, pwy, qvy, qwy, quz, rwz, ruz, rvx$ are collinear, then the ninth triple is also collinear.

We can construct a CC-system on nine points that is unrealizable in the plane, by constructing a CC-system in which each of the triples occurring in the determinants of Pappus identity is a counterclockwise triple of points.

Figure 1 on the next page shows a symmetrical configuration of nine points that would correspond to the theorem of Pappus if the lines xv, yw, and zu were straightened so that the triangles now containing p, q, and r shrink to points, and if p, q, r move to those triple-intersection points. We have perturbed p, q, r slightly, and bent three of the lines, so that the triples pux, ruy, qvy, and six others obtained by cyclic rotation

$$p \to q \to r \to p\,, \qquad u \to v \to w \to u\,, \qquad x \to y \to z \to x$$

will all have counterclockwise orientation. (Some sort of perturbation and line-bending is obviously necessary if we are to have a diagram, because we know that no CC system containing the triples of Pappus configuration can be realizable.) The other counterclockwise triples needed, namely pqw, pqx, qpy, and their cyclic counterparts under the condition above, are clearly present on the picture.

The Desargues configuration was introduced as nonrealizable by Shor [35]. This configuration contains 10 points and 10 lines. Similar to Pappus' theorem for nine points, Desargues' theorem proves that if nine of the triples of points (lines) are collinear (concurrent), the last triple must also be collinear. So, Desargues' theorem is an extension of the Pappus configuration to 10 points. By bending the lines in the Pappus and the Desargues configurations, we can obtain nonrealizable uniform configurations as shown on figures 2 — 7.

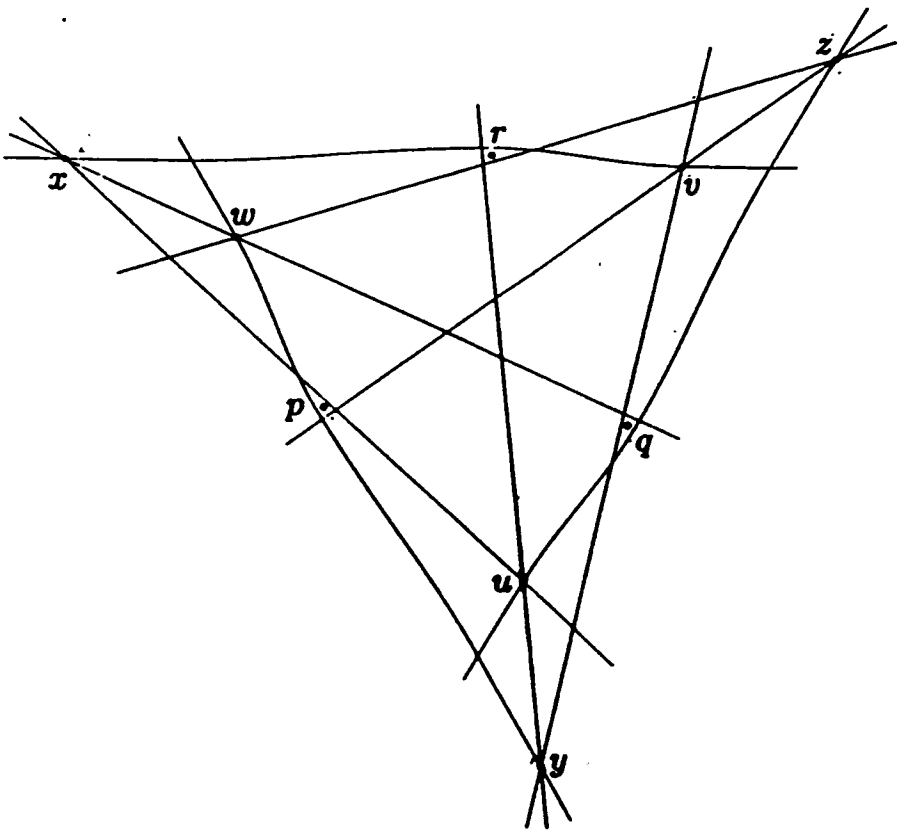
3.3 Our approach

In general, realizability problem can be approached from different angles. There have been various combinatorial approaches to this question from the point of view of pseudoline arrangements and oriented matroids, in particular, those by Goodman and Pollack.

The approach we have taken in our thesis was to attempt realizing the maximum number of CC-systems on 9 or more points by assigning Euclidean coordinates and performing counterclockwise tests. The problem of realizability still remains to be NP-hard from the complexity point of view per [35] but it is possible to test realizability of the system geometrically in finite time as proven in [33]. So our attempt was designing a realizability test for CC-systems, since heuristic solution to this question cannot be found in polynomial time.

We particularly concentrated on realizability solutions based on rules and conditions of CC-systems. By analyzing Knuth's convex hull algorithms and transitivity characteristics of tournaments, it became possible to suggest an

Figure 1.



A non-Euclidean CC-system (The Pappus Configuration)

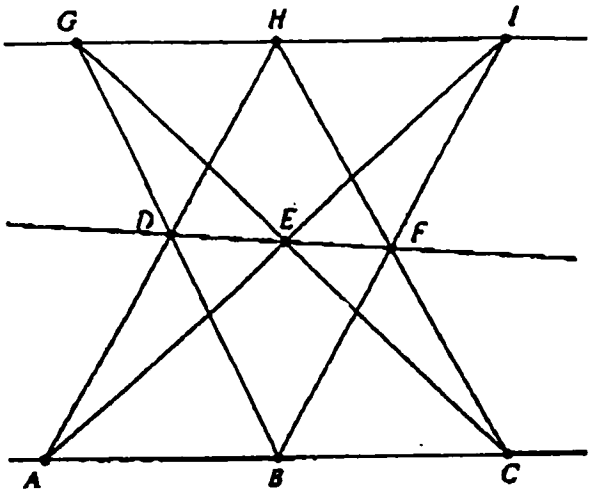


Figure 2. A Pappus configuration.

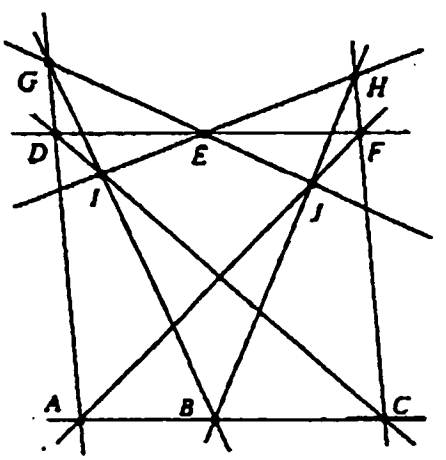


Figure 3. A Desargues configuration.

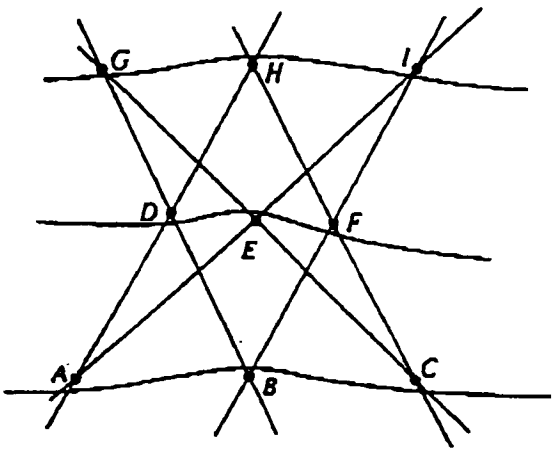


Figure 4. A nonrealizable Pappus configuration.

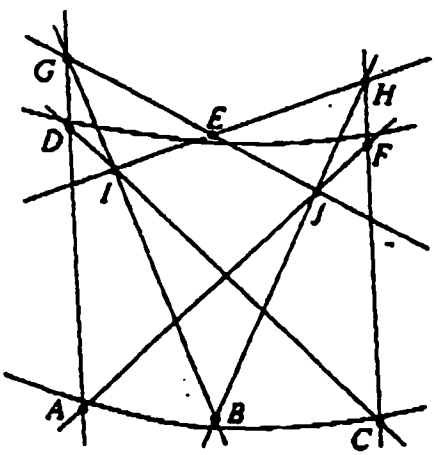


Figure 5. A nonrealizable Desargues configuration.

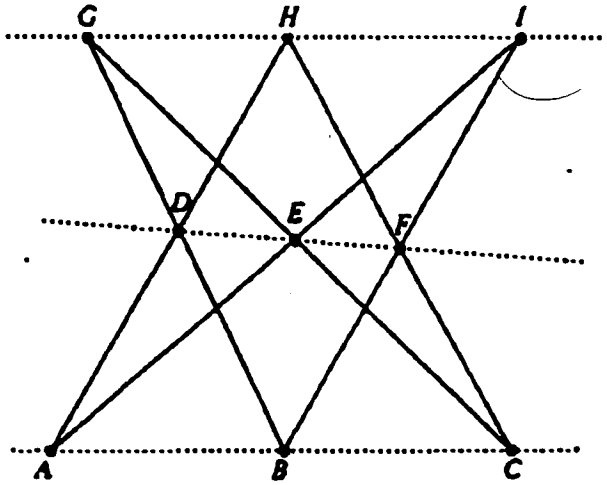


Figure 6. A Pappus configuration with two "imaginary" lines.

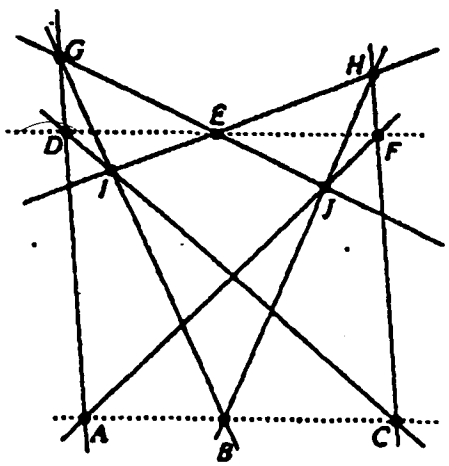


Figure 7. A Desargues configuration with two "imaginary" lines.

algorithm that would embed the given CC-system geometrically. The task of finding Euclidean coordinates for arbitrary CC-system while satisfying all axioms is of basic importance in our work. By finding optimal coordinates at which a given CC-system is embedded in the Euclidean plane and satisfies all axioms of Knuth, we can strongly state that the system is realizable and it does arise from a Euclidean plane.

Another approach taken is the conversion from a point system defined by Euclidean coordinates into CC-system. We have written a program which translates point system defined by Euclidean coordinates to a CC-system in a canonical format. An appropriate mechanism of axiom testing has been used as a feature in this program. It allows us to check whether the configuration generated from coordinates is a valid CC-system or not. Transformation from a Euclidean system into a CC-system and backwards allows us to develop an understanding of a CC-system as a point configuration based on counterclockwise triple predicate. Having this capability also enabled us to develop a graphical user interface for visualizing as well as creating, editing and storing realizable CC-systems from Euclidean coordinates.

Definition of convex hull algorithms, heuristics of arbitrary assignment of coordinates to points of the hull as well as the application of the linear programming techniques (similar to simplex) to halfplanes generated by these hulls are presented below in a section 4 on algorithms. In the same section, we presented the developed Java graphical user interface for CC-systems.

4 Algorithms and software system description

4.1 Convex hull algorithms

In [29] Knuth asserts that every CC-system on three or more points contains at least three extreme points. In general, this is quite obvious, that every point system would be bound by at least a triangle. Nevertheless, this fact was proven by the lemma introduced in section on convex hulls and it basically means that in any CC-system there is at least one transitive tournament associated with points of this system.

We implemented a simple algorithm (realize.c), which for a given CC-system represented by tournaments of its points, detects a 3-point cycle in a tournament of every point. Detection of just one 3-point cycle is strong enough, according to Axiom 4, to conclude that a given point is not in current convex hull. By continuously going through the point tournaments and removing the points which are extreme (with transitive tournaments),

```
for each v ∈ Adj[u]
    do if color[v] = WHITE
        then color[v] ← GRAY
            π[v] ← u
            ENQUEUE(Q, v)
    else if color[v] = GRAY
        Points in Q form a cycle!!
        break
DEQUEUE(Q)
color[u] ← BLACK
```

The above algorithm first colors all points in a given system *white*. Then it picks any source point s from tournament of a given point p and colors it *gray*, sets the predecessor point π to null, and adds point s as u to the front of queue Q (the queue is empty initially). This is easy to do from matrix representation of the tournament. Further, the algorithm searches for points adjacent to point u or head of the queue Q. As opposed to BFS described in [5], which is performed on an undirected graph, our algorithm searches for adjacent points v using directed edges, which are marked as 1's in the matrix of the tournament. All unrecognized (white) adjacent points are colored gray upon recognition and are *enqueued* in Q.

The main difference of this algorithm from the one presented in [5] is that it quits once it detects an adjacent gray point, which basically means that point is already in the queue Q and we have detected a cycle. Detection of the cycle means that the point p has a nontransitive tournament, and thus it is not a convex point of the system.

The breadth-first search algorithm usually runs with complexity $O(n)$ on the tournament of one point. Since there are n points and tournaments for a given CC-system the complexity of algorithm in determining one convex-hull would be $O(n^2)$.

4.2 Realizability solver

4.2.1 Algorithms and their application

Another algorithm has been developed based on linear programming techniques and Axiom 4 (interiority) for CC-systems. It is applied to all 3-point cycles in a tournament of a given point of the system. Thus, knowing the coordinates of points forming these 3-point cycles, the simplex method determines the *realizability space* for any given point in CC-system. In lin-

ear programming terms, this space would be the intersection of halfplanes formed by all 3-point cycles in a tournament. If the point in question is geometrically embedded in that area, all CC-system triple relations are satisfied. By applying this algorithm to each point of the system, we can search for a realization space of that point. If realization spaces for all points are found, then the entire system is realizable.

We start by defining a coordinate space to embed the given CC-system. In our testing we used a square area of 10,000 points (100×100). Depending on the complexity of the systems under study, the size of this plane (space) can be increased or decreased. Then we arbitrarily assign Euclidean coordinates to points of the first hull determined by one of the convex hull algorithms from the previous section. This can be done in any fashion, and initially we used circular way of coordinate assignment, i.e.

$$x_n = R - R \times cos(360 \times n/N) \text{ and } y_n = R + R \times sin(360 \times n/N)$$

where x_n, y_n are the Euclidean coordinates of a given point n, N is the total number of points in a given hull, and R is the radius of a random circle along which we are distributing the points of the initial hull. In this case, n points in the first hull would geometrically form an equal-sided n-gon. This way of initial hull distribution is convenient, but, nevertheless, cannot be constant. Depending on CC-systems, we might have to change the initial hull configuration in attempt to realize the whole system.

The coordinates of each given point inside the hull are then determined by scanning the whole area and by applying a set of linear programming (LP) conditions to halfplanes formed by nontransitive triples (3-point cycles) in the tournament of a given point. Now we use only those 3-point cycles, where coordinates of every point forming the cycle are already determined by the program. The coordinates found for each point are stored by the program and immediately used in determining the coordinates of remaining points.

For example, suppose we have a CC-system of 6 points p, q, r, s, t, u and after applying convex hull algorithm, we determine p, r, t, u as the first (outside) hull. Then assign coordinates to p, r, t, u in a circular fashion as described above. In order to determine the realizability area of, say, point q in the next hull, we apply our LP conditions to halfplanes formed by those nontransitive triples in the tournament of q, which involve p, r, t, u so that

if $q \in \Delta ptu$ **and** $q \in \Delta pru$

then $q \in ptu \wedge pru$

Our algorithm finds the geometric intersection of ptu and pru (as well as any number of halfplanes), while the coordinates of points involved and tangent factors of lines between them are already calculated. Intersection in this case is defined by the set of points bound by lines pt, tu, up, pr, ru, and if after the application of our linear conditions the intersection is found, then point q is geometrically realizable. Once the coordinates of q are determined, we can use them in finding the next point. Our algorithm works as follows:

Given $arx[10000]$, $ary[10000]$ **of points on 100x100 space**
$c = 10000, cc = 0;$
for every Δptu **in the tournament of** q
  determine tangent factors k_{pt}, k_{tu}, k_{up}
  determine line offsets b_{pt}, b_{tu}, b_{up}
  for every line l **in the triple** ptu
  $cc = c;$
  $c = 0;$
    for every point $i = 0 \rightarrow cc$ **in arrays** arx **and** ary
      if$((k_l < 0$ && $\Delta y_l < 0$ && $ary[i] > k_l * arx[i] + b_l)$ ||
        $(k_l < 0$ && $\Delta y_l > 0$ && $ary[i] < k_l * arx[i] + b_l)$ ||
        $(k_l > 0$ && $\Delta y_l > 0$ && $ary[i] > k_l * arx[i] + b_l)$ ||
        $(k_l > 0$ && $\Delta y_l < 0$ && $ary[i] < k_l * arx[i] + b_l)$ ||
        $(\Delta y_l == 0$ && $\Delta x_l > 0$ && $ary[i] > b_l)$ ||
        $(\Delta y_l == 0$ && $\Delta x_l < 0$ && $ary[i] < b_l)$ ||
        $(\Delta x_l == 0$ && $\Delta y_l > 0$ && $arx[i] < x_l)$ ||
        $(\Delta x_l == 0$ && $\Delta y_l < 0$ && $arx[i] > x_l))$ **then**
          $arx[c] = arx[i];$
          $ary[c] = ary[i];$
          $c++;$
$x_q = arx[c/3];$
$y_q = ary[c/3];$

The set of conditions defined for each line in triples works similar to simplex algorithm by establishing a set of inequalities for finding realization space (or feasible area) of the point. The reliability of this algorithm in correctly embedding the given CC-system is verified by absolute scanning of the area of 100×100 points, and considering all triple relationships. Testing results with our algorithm are presented in the next four subsections.

4.2.2 Basic testing of the realizability solver

In order to test how correct is our realizability solver in realizing CC-systems geometrically, we tried it on systems that are already known to be realizable. Using graphical user interface and *createcc* program, we generated various CC-systems from points with known coordinates in Euclidean plane.

The output of the realizability solver on a single 11-point CC-system with initial coordinates:

Point No.	X	Y
0	20	10
1	100	80
2	31	70
3	38	100
4	0	50
5	43	34
6	80	40
7	65	65
8	90	22
9	70	78
10	60	20

looks as follows:

System 1 Starting

There are 5 extreme points in convex 0: 0 8 1 3 4
Removing convex hull.....
There are 6 nonextreme points: 2 5 6 7 9 10

There are 5 extreme points in convex 1: 2 5 10 6 9
Removing convex hull.....
There are 1 nonextreme points: 7
Point 0: 0,50
Point 8: 34,2
Point 1: 90,20
Point 3: 90,79
Point 4: 34,97
Point 2: 54,76
Point 5: 20,48

Point 10: 16,29
Point 6: 52,9
Point 9: 83,46
Point 7: 71,44

If we build both configurations on a plane, they have the same set of consecutive convex hulls and satisfy the same set of triples (counterclockwise predicates).

4.2.3 Testing results on 8-point CC-systems

Testing on 8-point systems was of particular importance in proving the reliability of our solver. As mentioned in previous sections, Goodman and Pollack proved that all CC-systems on up to 8 points are realizable. Thus, if our solver is able to embed all nonisomorphic 8-point systems (total 3,315) under various hull configurations, then it works properly.

Presented below are the realizability testing results for 8-point CC-systems:

No. of hull points	3	4	5	6	7	8
Total No. of Systems	1,178	1,468	570	90	8	1
Realized Systems	1,178	1,468	570	90	8	1
Unrealized Systems	0	0	0	0	0	0
Hull Configurations	22	13	7	4	2	1

Eventually, all 8-point systems were embedded with definite Euclidean coordinates under various hull configurations tried. Most counterclockwise systems are bound by a 4-gon or a triangle. As it is shown in next 2 sections, this pattern remains in CC-systems on 9 and 10 points.

4.2.4 Testing results on 9-point CC-systems

We have tested the realizability solver on a file containing all nonisomorphic 9-point CC-systems (total 158,830). These systems are of particular interest in our study, because all systems on less than 9 points are known to be realizable, according to Goodman and Pollack. There is one theoretically nonrealizable CC-system known on 9 points, as defined by Knuth the Pappus configuration. Nonrealizable 9-pseudoline arrangement was also presented by Ringel [34] as mentioned in previous sections.

Our task was to attempt embedding all non-isomorphic 9-point point systems geometrically, and determine those apart from Pappus, which are not realizable. We classified all 9-point systems by the number of points in their convex hull. Our testing revealed that majority of systems is bound by 4-gons (70,482) and triangles (55,239). Eventually, all 9-point systems with five-, six-, seven-, eight- and, of course, nine-point hulls (total 154,417) were found embeddable by our realizability solver. The Pappus system was found unrealizable among those with 3-point convex hulls. Testing results are presented in the table below:

No. of hull points	3	4	5	6	7	8	9
Total No. of Systems	55,239	70,482	28,234	4,552	311	11	1
Realized Systems	52,882	68,478	28,234	4,552	311	11	1
Unrealized Systems	2,357	2,004	0	0	0	0	0
Hull Configurations	80	80	53	41	17	5	1

As it is obvious from table above, systems that have fewer points in their convex hulls require more hull configurations to be tried in an attempt to realize them. In the first configuration we distributed points along a circle forming an equal-sided n-gon. In all remaining configurations we manipulated with hull-point locations, so that all counterclockwise predicates (triples) would be satisfied with assigned and calculated coordinates.

Systems that have the first convex hull of 9 points are obviously realizable, since they form a cycle, and there are no more points in the system. Systems with a majority (6, 7, 8) of their points in the first convex hull are also relatively easy to decide, since we arbitrarily assign coordinates to the first hull, and the remaining (3, 2, 1) points inside the hull can be embedded by manipulating with outter hull configuration. Particular difficulty in using this solver arises in cases when there are only 3 or 4 points in the convex hull. In such systems, we can arbitrarily assign coordinates to only 3 or 4 points, and subsequently only $1 - 3$ triples can be used in determining the coordinates of first point inside the hull. Pappus configuration is an example of such system with 3-point convex hull.

The approach we have taken in designing the algorithm was to resolve the realizability question on maximum number of systems. Systems that were not realized by our solver are not necessarily nonrealizable. Particular attention should also be paid to NP-hardness of realizability problem. The approach taken by our algorithm concentrates on a practical embedding attempt using geometric coordinates, and NP-hardness still remains to be

an issue. Other methods of realizability solution might be applied in attempt to embed the remaining systems found unrealizable by our program.

4.2.5 Testing results on 10-point systems with complementary symmetry

Another realizability testing we have attempted is on all 10-point systems with complementary symmetry. These are the systems that look the same if we flip all triangle orientations. In [35] Shor proved that such symmetric pseudoline arrangements are stretchable (realizable) only to nonsymmetric line arrangements.

The pattern of this testing is similar to the previous one on nonisomorphic 9-point systems. Most 10-point systems (total 13,103) is bound by 4-gons (4,578) and triangles (2,962). Testing results are presented below:

No. of hull points	3	4	5	6	7	8	9	10
Total No. of Systems	2,962	4,578	3,227	1,676	538	106	15	1
Realized Systems	2,014	4,022	3,227	1,676	538	106	15	1
Unrealized Systems	948	556	0	0	0	0	0	0
Hull Configurations	25	25	19	16	12	9	4	1

Again, as we see the number of hull configurations tried in attempt to embed the system rises as the size of the hull gets smaller. The complexity of analyzing 10-point systems is also harder than that of 9-point systems.

4.3 Graphical user interface for geometric embedding

In addition to the combinatorial/geometric realizability solver, we have developed a graphical user interface for visualization of CC-systems and geometric embedding of points. The interface has been developed as a Java graphics application, and it provides the ability to display points on the screen based on coordinates generated by the realizability solver. It also allows to interactively add, move, and store points (with coordinates from screen) in a file. Dynamical features of this interface are achieved using *MouseAdapter* and *MouseMotionAdapter* classes in Java as well as *Mouse-* and *MouseMotionListener* interface functions including *mouseDown(), mouseDrag(), mousePressed(),* etc. The description of these classes is attached in Java code document. We also used CoreJava library package, provided along with "CoreJava - Advanced Features" book by C. S. Hortsman and G. Cornell, for file input/output from the interface.

The combinatorial realizability solver described in previous section calculates approximate coordinates for embedding arbitrary CC-system. Graphical user interface also serves as a second-hand testing tool for analyzing few CC-systems that cannot be realized by realizability solver or already known to be unrealizable theoretically (like the Pappus configuration). Visualization of such system allows the user to move the points and try different configurations of points in attempt to visually embed the point to Euclidean plane (screen).

Our interface also dynamically calculates the realizability area from halfplanes generated by nontransitive tournament triples and coordinates of points forming them, and displays this area, depending on current position of points. As opposed to realizability solver, this calculation is performed simply using the **contains** feature of the **Polygon** class in Java, and not by directly checking all linear inequalities. This reduces the complexity of our code significantly compared to implementation in C, since there is no separate function provided in C libraries for computing the intersection of halfplanes.

A menu added to our interface allows the user to perform point manipulation and triple display operations step-by-step. All coordinates for input of one CC-system are stored in a single .crd file. All triples associated with every point of a CC-system are stored in a separate .tri file for that point. Once we select to display points on our screen, we can then display all triples (halfplanes) with intersection marked in red for a particular point in question. This allows the user to actually visualize the approximate location where point can be embedded satisfying all counterclockwise predicates of the system.

In addition, this interface also allows to add points to existing CC-system and to edit existing CC-systems for obtaining new point configurations (with coordinates). We may also store coordinates in .crd file which is generated upon request from the menu. The file is named as n.crd, where n is total number of points in the CC-system. These coordinates can then be passed to our program *createcc* (in C) for transferring a new CC-system into canonical format. This feature was of particular importance in basic testing of our realizability solver.

4.4 Functional specification

4.4.1 Realizability solver functions (in C)

- **main()** function is the core of the realizability solver. It opens the source CC-file in canonical format and calls all necessary functions to geometrically realize a given system. Upon finding the convex hull, main function removes it and reiterates the search for a hull on remaining points. It also preserves the original numbering of points through its execution.
- **CCtoTN()** function extracts tournaments for every point of the given CC-system.
- **testn()** function performs an axiom testing on the system using the tournaments of points.
- **copycct()** function makes a copy of CC-system in canonical format.
- **outcct()** function stores a CC-system presented by tournaments of its points into a file.
- **cycle()** function detects all 3-cycles in the tournament of a given point of CC-system.
- **hull()** function calls *cycle()* function for every point of the system. It also stores points in the current convex hull separately.
- **convex_sort()** function sorts the points of the convex hull in counterclockwise order.
- **assign_hull()** function assigns arbitrary coordinates to points of the first (outter) hull. These coordinates are used later in embedding the whole system.
- **embed_point()** function is called for every point not in the first (outter) hull. This function executes simplex algorithm for determining the approximate realizability area for every given point.

4.4.2 CC-system generator

In addition to realizability solver we have developed a program that generates a CC-system from coordinates of points. In this program, we used

PtoCCt() function by Stanisław Radziszowski, which translates points presented by coordinates to CC-system in canonical format. This system is certainly known to be realizable, since it arises from coordinates in Euclidean plane.

- **main()** function opens coordinate file (.crd) to read, executes *read_points* function on input and writes the output to CC-system file (.cc) in canonical format.
- **PtoCCt(&T)** [by S.P.R.] function translates points into a CC-system in canonical format; T is the tournament representation of point system.
- **read_points()** function reads the coordinates from a file and uses *assert* condition for computing the number and validity of input points.

This CC-system generator is very useful in testing the correctness of our realizability solver. By generating CC-system from coordinate input and applying the realizability solver to output .cc file, we can check the matching (or closeness) of coordinates to initial ones from which CC-system is generated.

4.5 User's manual

4.5.1 Realizability solver

Realizability solver can be run using

jjh3710% realize ⟨*.cc*⟩

command. Here ⟨*.cc*⟩ is the name of the file containing various configurations of CC-systems (in canonical format) on the same number of points. Upon batch-mode execution this program performs the realizability check on each CC-system in the file. The program automatically separates systems which are not realizable geometrically under the given arbitrary configuration of initial hull. Later these systems can be retested by changing initial hull configuration.

Another version of this solver is *realizesingle* which records coordinates for an input of one CC-system into a coordinate file (*.crd). This file can be passed later to our graphical interface for displaying the CC-system.

4.5.2 Graphical user interface

GUI can be started using

jjh3710% java win

command at Unix prompt. This will bring up the Java application window of size 500x500 points with a menu.

CC-system from a certain coordinate (.crd) file is displayed by selecting

CC-systems − > Display Points from File − > filename.crd

combination from the menu. Selection

CC-systems − > Display Triples from File − > filename.tri

displays triples for a particular CC-system. In this case, the intersection of halfplanes for a particular point is dynamically calculated and displayed as a red-colored area. **CC-systems** menu option also provides suboptions that allow the user to edit a CC-system (add or move points) and save the system to a file.

4.5.3 CC-system generator

This program is run using

jjh3710% createcc ⟨*.crd*⟩ ⟨*.cc*⟩

Here ⟨*.crd*⟩ file contains coordinates of points in the system, and output ⟨*.cc*⟩ contains CC-system in canonical format generated by the program.

5 Conclusion

The realizability problem is complex in a sense that there are no definite axioms or theorems to characterize realizable point configurations. Since this problem is proven to be NP-hard, it's impossible to come up with an algorithmic solution in polynomial time. However, it's possible to test the realizability of the system using linear programming techniques and combinatorial counterclockwise tests.

In the course of this thesis, we have combined combinatorial and geometric principles for finding a solution to realizability problem. The algorithms developed allow us to attempt geometric embedding of counterclockwise systems by finding convex hulls, point distribution in counterclockwise order within each hull, and triple relationships. If embedding is performed successfully, it is sufficient to prove that the given system is realizable, or arises from Euclidean halfspace.

The solver we have developed tests the realizability of given counterclockwise systems in a batch mode. The main purpose of this solver is to attempt deciding on an arbitrary system by assigning random coordinates to convex hull and checking realizability spaces of each point using nontransitive triples in its tournament. Point-by-point embedding approach creates a confidence that the system embedded by the solver is in fact realizable. However, it does not guarantee that the system is nonrealizable if it's not found to be realizable by the solver.

From tests conducted with systems on eight, nine and ten points, it's obvious that with decreasing number of points in the hull of any system, the complexity of realizability solution increases. This is mainly because the initial hull assignment is done randomly, and unfortunately there is no certain way to determine an exact graphical configuration of convex hull. The only information that can be obtained by analyzing CC-system is the location of points in regards to each other, i.e., point distribution in counterclockwise order within the convex hull, triples involving certain points, etc. Still the information is not sufficient enough to establish certain locations of hull points and randomness in coordinate assignment is needed. Thus testing with all possible hull configurations requires considerable time and effort. This problem can be partially solved by increasing the embedding space, but such an approach would also increase the complexity and running time of the program.

A graphical user interface we developed is an effective tool to manually (graphically) decide on realizability of one particular system. It allows the user to visualize the system, attempt different hull configurations, and view

the realizability spaces dynamically. This interface can be used if the solver determines that the system is nonrealizable.

Future work on a realizability solver could concentrate on development of a random coordinate assignment mechanism that would consider all possible configurations while preserving the minimal running time of the program. In addition, linking the graphical user interface to realizability solver would allow to process CC-systems in a batch mode directly from the interface.

Many problems of computational geometry contain a rich collection of questions with topological flavor. In the framework of CC-systems, several other problems can be enumerated such as:

- Is every CCC-system embeddable in a CCCC-system?
- What are the properties of CCCC-systems realizable in Euclidean 4-space.

Here, a CCC-system is a relation on ordered quadruples (as opposed to triples in CC-system), satisfying five axioms obeyed by the incircle relation. A CCCC-system is defined as a hypertournament of rank 5 such that fixing any point yields a CCC-system. Attempting various realizability solutions on CCC- and CCCC-systems using rules and conditions of those systems would be a challenging effort for those interested.

References

[1] AGARWAL, P. K. Partitioning arrangements of lines: II. Applications. *Discrete Comput. Geom.*, 5:533-573, 1990.

[2] AVIS, D. and FUKUDA, K. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.,* 8:295-313, 1992.

[3] BOKOWSKI, J. and STURMFELS, B. Nonrealizability proofs in computational geometry. *Discrete and Computational Geometry 5* (1990), pp. 333-350.

[4] CHAN, T. M. Y. Output-sensitive results on convex hulls, extreme points, and related problems. In *Proc. 11th Annual ACM Sympos. Comput. Geom.,* pp. 10-19, 1995.

[5] CORMEN, T.H., LEISERSON, C. E., and RIVEST, R. L. Introduction to Algorithms. *The MIT Press electrical engineering and computer science series,* Cambridge, Massachusetts, 1990, pp. 469-477.

[6] DYER, M. E. The complexity of vertex enumeration methods. *Math. Oper. Res.*,8:381-402, 1983.

[7] EDELSBRUNNER, H. *Algorithms in Combinatorial Geometry,* volume 10 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, Heidelberg, West Germany, 1987.

[8] EDELSBRUNNER, H., GUIBAS, L. J., HERSHBERGER, J., SEIDEL, R., SHARIR, M. and WELZL, E. Implicitly representing arrangements of lines or segments. *Discrete Comput. Geom.,* 4:433-466, 1989.

[9] EDELSBRUNNER, H., GUIBAS, L. J., and SHARIR, M. The complexity of many cells in arrangements of planes and related problems. *Discrete Comput. Geom.*,5:197-216, 1990

[10] EDELSBRUNNER, H., and GUIBAS, L. J. Topologically sweeping an arrangement. *J. Comput. Syst. Sci.,* 38:165-194, 1989. Corrigendum in 42 (1991), 249-251.

[11] EDELSBRUNNER, H., GUIBAS, L. J., and SHARIR, M. The complexity and construction of many faces in arrangements of lines and of segments. *Discrete Comput. Geom.*,5:161-196, 1990

[12] EDELSBRUNNER, H. and MÜCKE, E. P. Simulation of simplicityL, a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.,* 9:66-104, 1990

[13] EDELSBRUNNER, H., O'ROURKE, J. and SEIDEL, R. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.,* 15:341-363, 1986

[14] EDELSBRUNNER, H., SEIDEL R. and SHARIR, M. On the zone theorem for hyperplane arrangements. *SIAM J. Comput.* , 22(2):418-429, 1993

[15] ERDÖS, P. and PURDY, G. Two combinatorial problems in the plane. *Discrete Comput. Geom.,*13(3-4):441-443, 1995.

[16] FELSNER, S. On the number of arrangements of pseudolines. In *Proc. 12 th Annual ACM Sympos. Comput. Geom.,* pp. 30-37, 1996.

[17] FELSNER, S. and WEIL H. Sweeps, arrangements and signotopes. In `http://www.inf.fu-berlin.de/ felsner/Paper/sas-dam.ps.gz`

[18] FOLKMAN, J. and LAWRENCE, J. Oriented Matroids. *J. Combin. Theory,* Ser. B, 25 (1987), 199-236

[19] FÜREDI, Z. and PALÁSTI, I. Arrangements of lines with a large number of triangles. *Proc. Amer. Math. Soc.,* 92(4):561-566, 1984.

[20] GOODMAN, J. E. and POLLACK, R. Proof of Grünbaum's Conjecture On The Stretchability Of Certain Arrangements Of Pseudolines, *Journal of Combinatorial Theory A29* (1980), pp. 385-390.

[21] GOODMAN, J. E. and POLLACK, R. Upper bounds for configurations and polytopes in R^d, *Discrete and Computational Geometry 1* (1986), pp. 219-227

[22] GOODMAN, J. E. and POLLACK, R. Allowable sequences and order types in discrete and computational geometry. In J. Pach, editor, *New Trends in Discrete and Computational Geometry,* volume 10 of *Algorithms and Combinatorics,* pp. 103-134, Springer-Verlag, 1993.

[23] GOODMAN, J. E., POLLACK, R., WENGER R., ZAMFIRESCU T., There is a universal topological plane. *In: Proc. of the Eighth Annual ACM Symposium on Computational Geometry* (1992)

[24] GRAHAM, R. L. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*,1:132-133, 1972.

[25] GRÜNBAUM, Branko. Convex Polytopes. *Wiley,* New York, NY, 1967. Revised edition, KLEE, V. and KLEINSCHMIDT, P., editors, *Graduate Texts in Mathematics,* Springer-Verlag, in preparation.

[26] GRÜNBAUM, Branko. Arrangements and Spreads. *Number 10 in Regional Conf. Ser. Math. Amer. Math. Soc.*,Providence, RI, 1972.

[27] KALHOFF, F. B. Oriented rank 3 matroids and projective planes. `http://www-lsii.mathematik.uni-dortmund.de/preprints/96-06.html`

[28] KATZ, M. J. and SHARIR, M. An expander-based approach to geometric optimization. In *Proc. 9th Annual ACM Sympos. Comput. Geom.,* pp.198-207.

[29] KNUTH, D. E. Axioms and Hulls, *volume 606 of Lecture Notes in Computer Science.* Springer-Verlag, Heidelberg, Germany 1992.

[30] LEVI, F. Die Teilung der projectiven Ebene durch Gerade oder Pseudogerade, *Berichte über die Verhandlungen der sächsischen Akademie der Wissenchaften,* Leipzig, Mathematisch-physische Klasse 78 (1926), 256-267.

[31] MNËV, N. E. The Universality Theorems on The Classification Problem of Configuration Varieties and Convex Polytopes Varieties. *in Topology and Geometry - Rohlin Seminar (O.Y.Viro, ed.),* Lecture Notes in Math., vol. 1346, Springer, Berlin, 1988, pp. 527-544.

[32] RICHTER-GEBERT, J. The Universality Theorems of Oriented Matroids and Polytopes. *Proceedings of the 1996 AMS-IMS-SIAM Joint Summer Research Conference,* July 14-18, 1996, Mt. Holyoke College, *Advances in Discrete and Computational Geometry,* American Mathematical Society, Providence, RI, 1999.

[33] RICHTER-GEBERT, J. The Combinatorial Realizability Criteria for Oriented Matroids. *Mitteilungen aus dem Mathematischen Seminar Giessen 194* (1989), 113 pp.

[34] RINGEL, J. Über Geraden in Allgemeiner Lage, *Elemente der Matematik 12* (1957), pp. 75-82.

[35] SHOR, P. W. Stretchability of pseudolines is NP-hard. In *Applied geometry and Discrete Mathematics: The Victor Klee Festschrift,* volume 4 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science,* pp. 531-554. AMS Press, 1991.

Class win

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Panel
                    |
                    +--java.applet.Applet
                          |
                          +--win
```

public class **win**
extends java.applet.Applet
implements java.awt.event.ActionListener

Public class which invokes all menu, input/output, and display functions.

See Also:
Serialized Form

Field Summary

static int	**apoints** Public counter that holds total number of points including those added to the system
static java.lang.String[]	**ccfile** Public string array that contains the list of all .crd files in current directory.
static int	**flag** Public flaq that is raised (set to 1) upon the mouse click on already existing point.
static java.lang.String	**globaltri** Public string that contains the name of .tri file
protected static int[]	**lastx** Protected array for x coordinates of points

protected static int[]	**lasty** Protected array for y coordinates of points
static java.lang.String	**newccfile** Public string that contains the name of .crd file
protected static int[]	**pa** Protected static array that contains first points in non-transitive tournament triples (read from .tri file)
protected static int[]	**pb** Protected static array that contains second points in non-transitive tournament triples (read from .tri file)
protected static int[]	**pc** Protected static array that contains third points in non-transitive tournament triples (read from .tri file)
static int	**pnt** Public field that catches the point (label) upon the mouse click and flag setting
static int	**points** Public counter that holds total number of points in input .crd file
static int	**triples** Public counter for all triples in .tri file
static java.lang.String[]	**trpfile** Public string array that contains the list of all .tri files in current directory.

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Constructor Summary

win()

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent e) Public method that calls various point manipulation functions upon receving an appropriate command from the menu
protected void	**addpoint**() Protected method that adds points to application interface upon mouse click
protected void	**clear**() Protected method that is called to clear application screen, or set the background to default white
static void	**CreateInterface**() Public method that creates application interface as well as its menu and filters for reading .crd and .tri files from the directory
void	**init**() Public method that initializes the screen for the application.
static void	**main**(java.lang.String[] args) Public method that calls CreateInterface method
protected void	**mvpoint**() Protected method that moves the point from location where mouse is pressed to location where mouse is released.
void	**readcc**(java.lang.String ccf) Public method that reads the coordinate file (.crd) line-by-line.
void	**readtriple**(java.lang.String trp) Public method that reads all triples (cycles) for particular point tournament from .tri file and displays them on the screen along with their intersection, or realizability space, (if any) colored in red
protected void	**writecc**() Protected method that writes coordinates of points on the interface into a .crd file.

Methods inherited from class java.applet.Applet

destroy, getAppletContext, getAppletInfo, getAudioClip, getAudioClip, getCodeBase, getDocumentBase, getImage, getImage, getLocale, getParameter, getParameterInfo, isActive, newAudioClip, play, play, resize, resize, setStub, showStatus, start, stop

Methods inherited from class java.awt.Panel

addNotify

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, addImpl, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, layout, list, list, locate, minimumSize, paint, paintComponents, paramString, preferredSize, print, printComponents, processContainerEvent, processEvent, remove, remove, removeAll, removeContainerListener, removeNotify, setFont, setLayout, update, validate, validateTree

Methods inherited from class java.awt.Component
action, add, addComponentListener, addFocusListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addPropertyChangeListener, addPropertyChangeListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getInputContext, getInputMethodRequests, getLocation, getLocation, getLocationOnScreen, getName, getParent, getPeer, getSize, getSize, getToolkit, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside, isDisplayable, isDoubleBuffered, isEnabled, isFocusTraversable, isLightweight, isOpaque, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, remove, removeComponentListener, removeFocusListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, reshape, setBackground, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setEnabled, setForeground, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, show, size, toString, transferFocus

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

lastx

```
protected static int[] lastx
```

Protected array for x coordinates of points

---

lasty

```
protected static int[] lasty
```

Protected array for y coordinates of points

---

points

```
public static int points
```

Public counter that holds total number of points in input .crd file

---

apoints

```
public static int apoints
```

Public counter that holds total number of points including those added to the system

---

pa

```
protected static int[] pa
```

Protected static array that contains first points in non-transitive tournament triples (read from .tri file)

---

pb

```
protected static int[] pb
```

Protected static array that contains second points in non-transitive tournament triples (read from .tri file)

---

pc

```
protected static int[] pc
```

Protected static array that contains third points in non-transitive tournament triples (read from .tri file)

---

triples

```
public static int triples
```

Public counter for all triples in .tri file

---

trpfile

```
public static java.lang.String[] trpfile
```

Public string array that contains the list of all .tri files in current directory. This list is displayed in the menu for triple files.

---

ccfile

```
public static java.lang.String[] ccfile
```

Public string array that contains the list of all .crd files in current directory. This list is displayed in the menu for CC-systems.

---

newccfile

```
public static java.lang.String newccfile
```

Public string that contains the name of .crd file

---

globaltri

```
public static java.lang.String globaltri
```

Public string that contains the name of .tri file

---

flag

```
public static int flag
```

Public flaq that is raised (set to 1) upon the mouse click on already existing point. This feature is used in point movement functions

---

pnt

```
public static int pnt
```

Public field that catches the point (label) upon the mouse click and flag setting

Constructor Detail

win

```
public win()
```

Method Detail

init

```
public void init()
```

Public method that initializes the screen for the application.

Overrides:

init in class java.applet.Applet

CreateInterface

```
public static void CreateInterface()
```

Public method that creates application interface as well as its menu and filters for reading .crd and .tri files from the directory

---

main

```
public static void main(java.lang.String[] args)
```

Public method that calls CreateInterface method

---

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent e)
```

Public method that calls various point manipulation functions upon receving an appropriate

command from the menu

Specified by:

actionPerformed in interface java.awt.event.ActionListener

---

clear

```
protected void clear()
```

Protected method that is called to clear application screen, or set the background to default white

---

readcc

```
public void readcc(java.lang.String ccf)
```

Public method that reads the coordinate file (.crd) line-by-line.

---

readtriple

```
public void readtriple(java.lang.String trp)
```

Public method that reads all triples (cycles) for particular point tournament from .tri file and displays them on the screen along with their intersection, or realizability space, (if any) colored in red

---

writecc

```
protected void writecc()
```

Protected method that writes coordinates of points on the interface into a .crd file.

---

addpoint

```
protected void addpoint()
```

Protected method that adds points to application interface upon mouse click

---

mvpoint

`protected void` **`mvpoint`**`()`

Protected method that moves the point from location where mouse is pressed to location where mouse is released. This method uses mousePressed, mouseReleased features of MouseListener

---

Class FileTest

```
java.lang.Object
  |
  +--FileTest
```

public class **FileTest**
extends java.lang.Object

Public class FileTest invoked to read data from .crd or .tri file or to write data into .crd file.

Field Summary

static int	**linecount** Public variable that contains the number of lines in a given .tri or .crd file

Constructor Summary

FileTest()

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

linecount

```
public static int linecount
```

Public variable that contains the number of lines in a given .tri or .crd file

Constructor Detail

FileTest

```
public FileTest()
```

Class CoordRead1

```
java.lang.Object
   |
   +--CoordRead1
```

public class **CoordRead1**
extends java.lang.Object

Field Summary

protected int	**coordx** Protected variable for x coordinate read from .crd file
protected int	**coordy** Protected variable for y coordinate read from .crd file

Constructor Summary

CoordRead1()
CoordRead1(int n, int s) Public method CoordRead1 called for assigning coordinates x and y.

Method Summary

void	**readData**(java.io.BufferedReader is) Public method that reads the data line-by-line from .crd file, where coordinates are delimited by comma.
void	**writeData**(java.io.PrintWriter os) Public method that writes the data line-by-line into .crd file, and delimits coordinates by comma.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

coordx

```
protected int coordx
```

Protected variable for x coordinate read from .crd file

---

coordy

```
protected int coordy
```

Protected variable for y coordinate read from .crd file

Constructor Detail

CoordRead1

```
public CoordRead1(int n,
                  int s)
```

Public method CoordRead1 called for assigning coordinates x and y.

CoordRead1

```
public CoordRead1()
```

Method Detail

readData

```
public void readData(java.io.BufferedReader is)
              throws java.io.IOException
```

Public method that reads the data line-by-line from .crd file, where coordinates are delimited by comma. This method uses Format method defined in CoreJava library for formatting the input/output data.

---

writeData

```
public void writeData(java.io.PrintWriter os)
              throws java.io.IOException
```

Public method that writes the data line-by-line into .crd file, and delimits coordinates by comma. This method uses Format method defined in CoreJava library for formatting the input/output data.

Class TripleRead

```
java.lang.Object
   |
   +--TripleRead
```

public class **TripleRead**
extends java.lang.Object

Public class invoked to read data from .tri file

Field Summary

protected int	**pointa** Protected variable for the first point in any triple
protected int	**pointb** Protected variable for the second point in any triple
protected int	**pointc** Protected variable for the third point in any triple

Constructor Summary

TripleRead()

Method Summary

void	**readTripleData**(java.io.BufferedReader is) Public method for reading the data from .tri file.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

pointa

```
protected int pointa
```

Protected variable for the first point in any triple

---

pointb

```
protected int pointb
```

Protected variable for the second point in any triple

---

pointc

```
protected int pointc
```

Protected variable for the third point in any triple

Constructor Detail

TripleRead

```
public TripleRead()
```

Method Detail

readTripleData

```
public void readTripleData(java.io.BufferedReader is)
                    throws java.io.IOException
```

Public method for reading the data from .tri file. This method uses Format method defined in CoreJava library for formatting the input or the output data.

---

jjh3710

Request: grad_lw1-901 from idaho

Options: flist='Makefile'

Title: Makefile

#####
idaho

Sun Jul 25 18:16:40 EDT 1999

```
********** Option Summary **********
 (See  "man net_ljx000" for details)

auto (default),   postscript,  pcl, hpgl2, hpgl2_p, raw, relay
manual, tray1, tray2, tray3, bin1, bin2,     mtype<media type>
legal, letter, exec, ledger/11x17       A3, A4, A5, A6, B5-ISO
com10, C5, DL, monarch           PostCardS, PostCard/PostCardD
B4-JIS, B5-JIS, B6-JIS               topaz, yb, nb, job, nojob
dpi#, fuser<fusing mode>              simplex, duplex, hduplex
2up, 2+,4up(pcl-hpux only), portrait, landscape, quality<mode>
srbb#, srb#, sre#,         tondensity<mode>, econo# (#=on/off)
For PS                                          ascii, ps1, ps2
For PCL .     text, ln66, stye#, italic, condensed, condensedi
c, 10, 12, lpi#  height#,  weight#, medium, bold, ebold, type#
```

```
CC = cc
COMPILE = $(CC) -c -lm
CFLAGS = -lm

hp:
        make clean; make all \
                 "CC = c89"

nonhp:
        make clean; make all

all:    realize listcc

realize:        convert.o lib.o realize.o simplexlib.o hull.o
        $(CC)  -o realize $(CFLAGS) convert.o lib.o realize.o simplexlib.o hull

lib.o:  lib.c macro.h
        $(COMPILE) lib.c

convert.o: convert.c macro.h
        $(COMPILE) convert.c

simplexlib.o: simplexlib.c macro.h
              $(COMPILE) simplexlib.c

realize.o: realize.c macro.h
        $(COMPILE) realize.c

clean:
        rm -f *.o
```

jjh3710

Request: grad_lw1-899 from idaho

Options: flist='simplexlib.c'

Title: simplexlib.c

#####
idaho

Sun Jul 25 18:14:53 EDT 1999

```
********** Option Summary **********
 (See  "man net_ljx000" for details)

auto (default),   postscript,  pcl, hpgl2, hpgl2_p, raw, relay
manual, tray1, tray2, tray3, bin1, bin2,     mtype<media type>
legal, letter, exec, ledger/11x17       A3, A4, A5, A6, B5-ISO
com10, C5, DL, monarch           PostCardS, PostCard/PostCardD
B4-JIS, B5-JIS, B6-JIS               topaz, yb, nb, job, nojob
dpi#, fuser<fusing mode>              simplex, duplex, hduplex
2up, 2+,4up(pcl-hpux only), portrait, landscape, quality<mode>
srbb#, srb#, sre#,         tondensity<mode>, econo# (#=on/off)
For PS                                        ascii, ps1, ps2
For PCL       text, ln66, stye#, italic, condensed, condensedi
c, 10, 12, lpi#  height#,  weight#, medium, bold, ebold, type#
```

```
/* simplexlib.c
 * Copyright 1999 Javid Huseynov
 *
 * This file contains all functions for geometric realization
 * of the system. It includes the implementation of linear
 * programming algorithm that calculates the realizability
 * space of any particular point.
 */

#include <math.h>
#include "macro.h"

#ifdef HP
#undef _WIN32
#define _INCLUDE_HPUX_SOURCE
#define _INCLUDE_XOPEN_SOURCE
#endif

/* function that arbitrarily assigns coordinates to
   point of the first convex hull */

assign_hull(m, hull, x, y)
int m;
int hull[SIZE];
int x[SIZE];
int y[SIZE];
{
   int i;

   for(i=0;i<m;i++) {
      x[hull[i]]=(int)(50-50*cos((360*i/m)*3.1415/180));
      y[hull[i]]=(int)(50-50*sin((360*i/m)*3.1415/180));
      printf("Point %d: %d,%d\n",hull[i],x[hull[i]],y[hull[i]]);
   }

}


/* function that embeds the point in question using the
 * coordinates of points in the first hull and all
 * counterclockwise relations
 */
embed_point(p,ptr,trc,x,y)
int p;
int ptr[SIZE][1000][3];
int trc[SIZE];
int x[SIZE];
int y[SIZE];
{
   int i,j, t;          /* indices */
   double k[3], b[3]; /* arrays for tangents and line offsets */
   int dx[3], dy[3];  /* coordinate difference arrays for triple */
   int arx[10000];    /* array of all x coordinates */
   int ary[10000];    /* array of all y coordinates */
   int cc=0,cc1=0;    /* point counters */


   /* fill up all coordinates on 100x100 space */
   for(i=0;i<100;i++)
      for(j=0;j<100;j++) {
```

```
      arx[cc]=i;
      ary[cc]=j;
      cc++;
   }


for(i=0;i<trc[p];i++) {
   /* check if any of the points in triples are assigned
    * any coordinates (if not then x=999 and y=999)
    */
   if(x[ptr[p][i][0]] != 999 && y[ptr[p][i][0]] !=999 &&
      x[ptr[p][i][1]] != 999 && y[ptr[p][i][1]] !=999 &&
      x[ptr[p][i][2]] != 999 && y[ptr[p][i][2]] !=999) {


      /* calculate the coordinate differences for every line */
      dx[0]=x[ptr[p][i][1]]-x[ptr[p][i][0]];
      dy[0]=y[ptr[p][i][1]]-y[ptr[p][i][0]];

      dx[1]=x[ptr[p][i][2]]-x[ptr[p][i][1]];
      dy[1]=y[ptr[p][i][2]]-y[ptr[p][i][1]];

      dx[2]=x[ptr[p][i][0]]-x[ptr[p][i][2]];
      dy[2]=y[ptr[p][i][0]]-y[ptr[p][i][2]];

      for(j=0;j<3;j++) {
        k[j]=(double)(dy[j])/(double)(dx[j]);
        b[j]=(double)(y[ptr[p][i][j]])-k[j]*(double)(x[ptr[p][i][j]]);
      }

      for(t=0;t<3;t++) {
         cc1=cc;
         cc=0;
         for(j=0;j<cc1;j++)
            /* check all counterclockwise conditions on triples */
            if((k[t]<0 && dy[t]<0 && ary[j]>=k[t]*arx[j]+b[t]) ||
               (k[t]<0 && dy[t]>0 && ary[j]<=k[t]*arx[j]+b[t]) ||
               (k[t]>0 && dy[t]>0 && ary[j]>=k[t]*arx[j]+b[t]) ||
               (k[t]>0 && dy[t]<0 && ary[j]<=k[t]*arx[j]+b[t]) ||
               (dy[t]==0 && dx[t]>0 && ary[j]>=y[ptr[p][i][t]]) ||
               (dy[t]==0 && dx[t]<0 && ary[j]<=y[ptr[p][i][t]]) ||
               (dx[t]==0 && dy[t]>0 && arx[j]<=x[ptr[p][i][t]]) ||
               (dx[t]==0 && dy[t]<0 && arx[j]>=x[ptr[p][i][t]])) {
                  /* add the x and y coordinates of realizability
                   * space to appropriate coordinate arrays */
               arx[cc]=arx[j];
               ary[cc]=ary[j];
               cc++;   /* increment the point counter */
               }

      }
   }
}
if(cc!=0) {.  /* if there is at least one point in the
               * realizability space */
   x[p]=arx[cc/3];
   y[p]=ary[cc/3];
   printf("Point %d: %d,%d\n",p,x[p],y[p]);
   return 1;
}
```

jjh3710

Request: grad_lw1-900 from idaho

Options: flist='hull.c'

Title: hull.c

#####
idaho

Sun Jul 25 18:15:22 EDT 1999

```
********** Option Summary **********
 (See  "man net_ljx000" for details)

auto (default),   postscript,  pcl, hpgl2, hpgl2_p, raw, relay
manual, tray1, tray2, tray3, bin1, bin2,     mtype<media type>
legal, letter, exec, ledger/11x17       A3, A4, A5, A6, B5-ISO
com10, C5, DL, monarch           PostCardS, PostCard/PostCardD
B4-JIS, B5-JIS, B6-JIS               topaz, yb, nb, job, nojob
dpi#, fuser<fusing mode>              simplex, duplex, hduplex
2up, 2+,4up(pcl-hpux only), portrait, landscape, quality<mode>
srbb#, srb#, sre#,         tondensity<mode>, econo# (#=on/off)
For PS  .                                      ascii, ps1, ps2
For PCL .     text, ln66, stye#, italic, condensed, condensedi
c, 10, 12, lpi#  height#,  weight#, medium, bold, ebold, type#
```

```
/* hull.c
 * Copyright 1999 Javid Huseynov
 *
 * This file contains convex hull functions
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "macro.h"

#ifdef HP
#undef _WIN32
#define _INCLUDE_HPUX_SOURCE
#define _INCLUDE_XOPEN_SOURCE
#endif

#define NDEBUG



/* function cycle loops through the given tournament
 * and determines all 3-point cycles */

int cycle(pp,n,point,flag,pts,ptr,trc,c)
struct tn *pp;
int n;
int point;
int flag;
int pts[SIZE];
int ptr[SIZE][1000][3];
int trc[SIZE];
int c;
{
   int i,j,k;

   if(c==0)
      trc[point]=0;

      for(i=0;i<n;i++)
         for(j=i+1;j<n;j++)
            if(pp->arc[i][j]==1)
               for(k=i;k<n;k++)
                  if ((pp->arc[j][k]==1) && (pp->arc[k][i]==1)) {
#ifdef NDEBUG
                     if(c==0) {
                        ptr[point][trc[point]][0]=pts[i];
                        ptr[point][trc[point]][1]=pts[j];
                        ptr[point][trc[point]][2]=pts[k];
                        trc[point]++;
                     }
                     fflush(stdout);
#endif
                     flag=1;
                  }
   return flag;
}

/* function hull calls cycle program for each point in the system */
```

```
int hull(P, ahull, nonhull, ccpoints, ptr, trc, c)
struct TN *P;
int ahull[SIZE];
int nonhull[SIZE];
int ccpoints[SIZE];
int ptr[SIZE][1000][3];
int trc[SIZE];
int c;
{
   int i, flag=0;
   int j=0;  /* convex point counter */
   int k=0;  /* non-convex point counter */

   for(i=0;i<P->n;i++)
     if(ccpoints[i]==NULL)
       ccpoints[i]=i;
       for(i=0;i<P->n;i++) {
         if(!cycle(&(P->tp[i]),P->n,ccpoints[i],flag,ccpoints,ptr,trc,c))
             ahull[j++]=ccpoints[i];
         else
             nonhull[k++]=ccpoints[i];
        }
        return j;
}


/* function that sorts points in a given convex hull
   in counterclockwise order */

convex_sort(T,m,list)
struct TN * T;
int *list;
{
   struct tn *pt;
   int j,done;

   pt = &T->tp[list[0]];

   for (done=2;done<m;done++)
      for (j=done-1;j;j--) {
         if (pt->arc[list[j]][list[j+1]]) break;
            else swap(list+j,list+j+1);
      }
}
```

jjh3710

Request: grad_lw1-898 from idaho

Options: flist='macro.h'

Title: macro.h

#####
idaho

Sun Jul 25 18:13:34 EDT 1999

```
********** Option Summary **********
 (See  "man net_ljx000" for details)

auto (default),   postscript,  pcl, hpgl2, hpgl2_p, raw, relay
manual, tray1, tray2, tray3, bin1, bin2,     mtype<media type>
legal, letter, exec, ledger/11x17       A3, A4, A5, A6, B5-ISO
com10, C5, DL, monarch           PostCardS, PostCard/PostCardD
B4-JIS, B5-JIS, B6-JIS               topaz, yb, nb, job, nojob
dpi#, fuser<fusing mode>              simplex, duplex, hduplex
2up, 2+,4up(pcl-hpux only), portrait, landscape, quality<mode>
srbb#, srb#, sre#,         tondensity<mode>, econo# (#=on/off)
For PS                                          ascii, ps1, ps2
For PCL       text, ln66, stye#, italic, condensed, condensedi
c, 10, 12, lpi#  height#,  weight#, medium, bold, ebold, type#
```

```
/* macro.h
 * Copyright 1998 Stanislaw Radziszowski
 * Updated 1999 Javid Huseynov
 */

#include <stdio.h>

#define min(x,y) ((x<y) ? (x) : (y))
#define max(x,y) ((x<y) ? (y) : (x))
#define SIZE 16                     /* points                */
#define PAIR 300                    /* C(points,2)           */
#define TRIP 50000                  /* C(points,3) + 6       */
#define OrdTrip 2100                /* 3*C(points,3) + 6     */
#define NUMSETS 131100              /* 2**points + 6         */
#define NUMSETS 131100              /* 2**points + 6         */
#define QSETS 2380                  /* C(n,4)                */
#define QUAD 2380                   /* C(n,4)                */
#define PSIZE 9                     /* max part to glue      */
#define PMAX 260                    /* max permutations      */

int convpoly[NUMSETS];              /* convex-cocave flag    */
int quadlist[QSETS];                /* covcave 4-gons        */
int centerlist[QSETS];              /* centers of 4-gons     */
int   bitpos[NUMSETS];              /* bit position          */
int     biti[SIZE+1];               /* the i-th bit          */
int chooses[0xFFFF][4];

struct CCp {                        /* n Knuth CC-pairs      */
        int n;
        int top[PAIR];
        int bot[PAIR];
        };

struct HP {
        int pair[PAIR];
        int trior[PAIR];
};

struct CCt {                        /* n oriented triangles */
        int n;
        int triple[TRIP];
        int dir[TRIP];
        };

struct tn {                         /* point p tournament    */
        int p;
        int arc[SIZE][SIZE];
        };

struct TN {                         /* n tournaments         */
        int n;
        struct tn tp[SIZE];
        };

struct triple {
        int a;
        int b;
        int c;
        };
```

```
typedef struct { int x, y, z; } triangle;

struct qconc4 {
        int center;
        int ext;
        };

struct qconv4 {
        int q;
        int diag1;
        int diag2;
        };

struct qconv5 {
        int q;
        int end;
        int cyc[5];
        };

struct set45 {
        int nc4;                    /* number of concave 4-sets  */
        int nv4;                    /* number of convex 4-sets   */
        int nv5;                    /* number of convex 5-sets   */
        struct qconc4 qc4[QSETS];   /* concave 4-sets            */
        struct qconv4 qv4[QSETS];   /* convex 4-sets             */
        struct qconv5 qv5[QSETS];   /* convex 5-sets             */
        };

struct pset45 {
        int nc4;                    /* number of concave 4-sets  */
        int nv4;                    /* number of convex 4-sets   */
        int nv5;                    /* number of convex 5-sets   */
        struct qconc4* qc4[QSETS]; /* concave 4-sets             */
        struct qconv4* qv4[QSETS]; /* convex 4-sets              */
        struct qconv5* qv5[QSETS]; /* convex 5-sets              */
        };

struct permedge {
        int *p0;                    /* perm0 array               */
        int *p1;                    /* perm1 array                        */
        int  pref;                  /* prefixes of pairs         */
        };

struct convex_str {
        int hull[SIZE];             /* points in current hull       */
        int nhull[SIZE];            /* points not in current hull   */
        int m;                      /* number of hull points        */
        int n;                      /* number of points not in hull */
};
```

jjh3710

Request: grad_lw1-228 from michigan

Options: flist='convex.c'

Title: convex.c

#####
michigan

Mon Jul 26 10:54:15 EDT 1999

```
********** Option Summary **********
 (See  "man net_ljx000" for details)

auto (default),   postscript,  pcl, hpgl2, hpgl2_p, raw, relay
manual, tray1, tray2, tray3, bin1, bin2,     mtype<media type>
legal, letter, exec, ledger/11x17       A3, A4, A5, A6, B5-ISO
com10, C5, DL, monarch           PostCardS, PostCard/PostCardD
B4-JIS, B5-JIS, B6-JIS               topaz, yb, nb, job, nojob
dpi#, fuser<fusing mode>              simplex, duplex, hduplex
2up, 2+,4up(pcl-hpux only), portrait, landscape, quality<mode>
srbb#, srb#, sre#,         tondensity<mode>, econo# (#=on/off)
For PS                                          ascii, ps1, ps2
For PCL       text, ln66, stye#, italic, condensed, condensedi
c, 10, 12, lpi#  height#,  weight#, medium, bold, ebold, type#
```

```
/* convex.c
 * Copyright 1998 Javid Huseynov
 * This file contains the convex hull algorithm implemented
 * with breadth-first search method
 */

#include <stdio.h>
#include <stdlib.h>
#include "macro.h"

extern incct(FILE*, struct CCt*);
extern CCtoTN(struct CCt*, struct TN*);
extern showtn(struct TN*);

#define NDEBUG

struct vertex {
        int color;
};

int front=1, rear=0;

int increment (int value, int n) {
        return (value+1)%(n+1);
}

void Enqueue (int value, int *BFSQueue, int n) {
        rear = increment(rear,n);
        BFSQueue[rear] = value;
}

int Head(int *BFSQueue) {
        return BFSQueue[front];
}

/* preconditions: queue is not empty */
void Dequeue(int n) {
        front = increment(front,n);
}

int is_empty(int n) {
        return increment(rear,n) == front;
}
/*****************************************************************/

void BFStour(struct tn *point_p, int n) {
        int i, u, s, count=0;
        struct vertex* V;
        int *BFSQueue = (int *)malloc(sizeof(int)*(n+1));
        printf("\nTour for point %d",point_p->p);
        V = (struct vertex *)malloc(sizeof(struct vertex)*n);

        for(i=0;i<n;i++) {
                if (i!=point_p->p) {
                        V[i].color = 0;
                }
        }
        while(count!=n-1) {
        for(i=0;i<n;i++) {
                if((V[i].color==0) && (i!=point_p->p)) {
```

```
                        s=i; break;
                }
        }
        V[s].color = 1;
        Enqueue(s,BFSQueue,n);
        /* search for a cycle */
        while (!is_empty(n)) {
                u=Head(BFSQueue);
                for(i=0;i<n;i++) {
                        if ((i!=point_p->p) && point_p->arc[u][i]) {
                                if (V[i].color==0) {
                                        V[i].color=1;
                                        Enqueue(i,BFSQueue,n);
                                }
                                else if (V[i].color==1){
                                        printf("\ncycle!!! %d",i);
                                        free(BFSQueue);
                                        return;
                                }
                        }
                }
                Dequeue(n);
                printf("\n%d",u);
                V[u].color = 2;
                count++;
        }
        }
        free(BFSQueue);
        return;
}

void BFS(struct TN *P) {
        int i;
        for(i=0;i<P->n;i++) {
                BFStour(&(P->tp[i]),P->n);
        }
}

int main(int argc, char ** argv) {
        FILE *inptr;
        struct CCt T;
        struct TN P;

        if (argc<2) exit(1);
        if ((inptr = fopen(argv[1],"r"))==NULL) {
                printf("convexBFS: cannot open '%s'\n",argv[1]);
                exit(1);
        }
        tabinit();
        while (incct(inptr,&T)) {
                CCtoTN(&T,&P);
                BFS(&P); /* perform a BFS on the tournaments */
        }
        fclose(inptr);
}
```

jjh3710

Request: grad_lw1-897 from idaho

Options: flist='realize.c'

Title: realize.c

#####
idaho

Sun Jul 25 17:47:41 EDT 1999

```
********** Option Summary **********
 (See  "man net_ljx000" for details)

auto (default),   postscript,  pcl, hpgl2, hpgl2_p, raw, relay
manual, tray1, tray2, tray3, bin1, bin2,     mtype<media type>
legal, letter, exec, ledger/11x17       A3, A4, A5, A6, B5-ISO
com10, C5, DL, monarch           PostCardS, PostCard/PostCardD
B4-JIS, B5-JIS, B6-JIS               topaz, yb, nb, job, nojob
dpi#, fuser<fusing mode>              simplex, duplex, hduplex
2up, 2+,4up(pcl-hpux only), portrait, landscape, quality<mode>
srbb#, srb#, sre#,         tondensity<mode>, econo# (#=on/off)
For PS                                              ascii, ps1, ps2
For PCL       text, ln66, stye#, italic, condensed, condensedi
c, 10, 12, lpi#  height#,  weight#, medium, bold, ebold, type#
```

```
/* realize.c
 * Copyright 1999 Javid Huseynov
 *
 * This file contains the main function that calls
 * convex hull algorithms, arbitrary hull assignment
 * functions, geometric realizability functions for
 * every system in the input .cc file. The main
 * function also performs all necessary format
 * conversions in-between various function calls.
 */


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "macro.h"

#ifdef HP
#undef _WIN32
#define _INCLUDE_HPUX_SOURCE
#define _INCLUDE_XOPEN_SOURCE
#endif

extern incct(FILE*, struct CCt*);
extern testtn(struct TN*);
extern CCtoTN(struct CCt*, struct TN*);
extern showtn(struct TN*);

#define NDEBUG

int main(int argc, char ** argv) {
   FILE *inptr, *outptr;
   struct CCt T, temp;
   struct TN P;
   int ahull[SIZE];      /* points in current convex hull */
   int nonhull[SIZE];    /* points not in current convex hull */
   int ccpoints[SIZE];   /* all points in the system */
   int m, mask, k, i, j, t=0;
   int c=0;              /* the number of convex hulls */
   struct convex_str convex[10];
   int crdx[SIZE];       /* x coordinates of points */
   int crdy[SIZE];       /* y coordinates of points */
   int ptr[SIZE][1000][3]; /* all 3-point in tournaments of all
                            * points in the system */
   int trc[SIZE];        /* number of 3-cycles for each point */
   int nembpc=0;         /* nonembeddable point flag */
   int nembcc=0;         /* output system counter */
   int syscount=0;       /* input system counter */


   if (argc<3)
      exit(1);

   /* open the input .cc file for reading */
   if ((inptr = fopen(argv[1],"r"))==NULL) {
      printf("convex: cannot open '%s'\n",argv[1]);
      exit(1);
   }
```

```
   /* open the output .cc file for writing
    * all systems unrealized by this program */
   if ((outptr = fopen(argv[2],"w"))==NULL) {
      printf("convex: cannot open '%s'\n",argv[2]);
      exit(1);
   }

   /* fill up the coordinate arrays with number 999,
    * which is out of range for our realizability space.
    * This means that there are no coordinates assigned
    * by the solver yet.
    */
   for (i=0;i<SIZE;i++) {
      crdx[i]=999;
      crdy[i]=999;
   }
   tabinit();

   while (incct(inptr,&T)){ /* while there are systems on the input */
      printf("---------------------\nSystem %d Starting\n",
             syscount+1);
      copycct(&T,&temp); /* make a copy of the system read */
      do {
         CCtoTN(&temp,&P); /* convert from cc format to tournament */
#ifdef DEBUG
         showtn(&P);
#endif
         /* perform an axiom testing */
         if (!testtn(&P)) {
            printf("Axiom testing has failed\n");
            continue;
         }
         /* determine the convex hull */
         m=hull(&P,ahull,nonhull,ccpoints,ptr,trc,c);
         convex[c].m=m;
         convex[c].n=P.n-m;

         /* sort points in the convex hull in CC order */
         convex_sort(&P,m,ahull);
         printf("\nThere are %d extreme points in convex %d:  ",m,c);
         for (i=0;i<m;i++) {
            convex[c].hull[i]=ahull[i];
            printf("%d ",convex[c].hull[i]);
         }
         printf("\nRemoving convex hull.....\n");

         k=0; mask=0;
         for(i=0;i<m;i++)
            mask|=1<<ahull[i];
         for(i=0;i<chooses[temp.n][3];i++) {
            if(!(temp.triple[i]&mask))
               temp.dir[k++]=temp.dir[i];
            if(k==chooses[temp.n-m][3])
               break;
         }
         temp.n-=m;
         printf("There are %d nonextreme points: ",temp.n);

         /* copy non-hull points into a new sub-CC system
          * for the next iteration of the algorithm */
```

```
        for (i=0;i<temp.n;i++) {
           convex[c].nhull[i]=nonhull[i];
           printf("%d ",convex[c].nhull[i]);
        }
        printf("\n");
        for(i=0;i<temp.n;i++)
           ccpoints[i]=nonhull[i];
           c++;  /* increment convex counter */
        } while (temp.n > 3);
        /* exit when the number of points is <= 3 */

        convex[c].m=temp.n;
        if(convex[c].m>0) {
           /* sort the hull points in cc order */
           convex_sort(&P,temp.n,nonhull);
           for (i=0;i<temp.n;i++)
              convex[c].hull[i]=nonhull[i];
        }


        /* arbitrarily assign coordinates to  points
         * of the first hull */
        assign_hull(convex[0].m,convex[0].hull,crdx,crdy);


        /* embed points one by one from consecutive convex hulls */
        for(i=1;i<=c;i++)
           for(j=0;j<convex[i].m;j++)
              if(!embed_point(convex[i].hull[j],ptr,trc,crdx,crdy))
                 nembpc=1;


        /* if detected at least one non-embeddable point, the
         * system cannot be realized by the solver */
        if(nembpc==1) {
           nembcc++;
           printf("TOTAL %d SYSTEMS ARE NON-EMBEDDABLE\n",nembcc);
           outcct(outptr,&T); /* write this system to the output */
        }

        /* Clean up after each system is analyzed */
        nembpc=0;
        for (j=0;j<=c;j++)
           for (i=0;i<SIZE;i++) {
              convex[j].hull[i]=NULL;
              convex[j].nhull[i]=NULL;
           }
        for (i=0;i<SIZE;i++) {
           ahull[i]=NULL;
           nonhull[i]=NULL;
           ccpoints[i]=NULL;
           crdx[i]=999;
           crdy[i]=999;
           trc[i]=NULL;
        }
        fflush(stdout);
        c=0;    /* reset convex counter to 0 */
        syscount++; /* increment the input system counter */

}
```

```
    fclose(inptr);
    fclose(outptr);
}
```

```
********** Option Summary **********
 (See  "man net_ljx000" for details)

auto (default),   postscript,  pcl, hpgl2, hpgl2_p, raw, relay
manual, tray1, tray2, tray3, bin1, bin2,     mtype<media type>
legal, letter, exec, ledger/11x17       A3, A4, A5, A6, B5-ISO
com10, C5, DL, monarch           PostCardS, PostCard/PostCardD
B4-JIS, B5-JIS, B6-JIS               topaz, yb, nb, job, nojob
dpi#, fuser<fusing mode>              simplex, duplex, hduplex
2up, 2+.4up(pcl-hpux only), portrait, landscape, quality<mode>
srbb#, srb#, sre#,         tondensity<mode>, econo# (#=on/off)
For PS                                          ascii, ps1, ps2
For PCL       text, ln66, stye#, italic, condensed, condensedi
c, 10, 12, lpi#  height#,  weight#, medium, bold, ebold, type#
```

```
/** TripleRead.java
  * Copyright @1998-1999 Javid Huseynov
  */

import java.io.*;
import java.util.*;
import corejava.*;

/** Public class invoked to read data from .tri file
  */

public class TripleRead
{
   public TripleRead() {}

   /** Public method for reading the data from .tri file. This method
     * uses Format method defined in CoreJava library for
     * formatting the input or the output data.
     */
   public void readTripleData(BufferedReader is) throws IOException
   {  String s = is.readLine();
      StringTokenizer t = new StringTokenizer(s, ",");
      pointa = Format.atoi(t.nextToken());
      pointb = Format.atoi(t.nextToken());
      pointc = Format.atoi(t.nextToken());

   }
   /** Protected variable for the first point in any triple
     */
   protected int pointa;

   /** Protected variable for the second point in any triple
     */
   protected int pointb;

   /** Protected variable for the third point in any triple
     */
   protected int pointc;
}
```

jjh3710

Request: grad_lw1-903 from idaho

Options: flist='win.java'

Title: win.java

#####
idaho

Sun Jul 25 18:35:58 EDT 1999

```
********** Option Summary **********
 (See  "man net_ljx000" for details)

auto (default),   postscript,  pcl, hpgl2, hpgl2_p, raw, relay
manual, tray1, tray2, tray3, bin1, bin2,     mtype<media type>
legal, letter, exec, ledger/11x17       A3, A4, A5, A6, B5-ISO
com10, C5, DL, monarch           PostCardS, PostCard/PostCardD
B4-JIS, B5-JIS, B6-JIS               topaz, yb, nb, job, nojob
dpi#, fuser<fusing mode>              simplex, duplex, hduplex
2up, 2+,4up(pcl-hpux only), portrait, landscape, quality<mode>
srbb#, srb#, sre#,         tondensity<mode>, econo# (#=on/off)
For PS  .                                        ascii, ps1, ps2
For PCL       text, ln66, stye#, italic, condensed, condensedi
c, 10, 12, lpi#  height#,  weight#, medium, bold, ebold, type#
```

```
/** win.java
  * Copyright @1998-1999 Javid Huseynov
  */

import java.awt.*;
import java.awt.Polygon.*;
import java.awt.event.*;
import java.awt.Menu;
import java.awt.MenuItem;
import java.awt.image.*;
import java.applet.*;
import java.io.*;
import java.util.*;
import corejava.*;
import java.net.*;


/** Public class which invokes all menu, input/output,
  * and display functions.
  */
public class win extends Applet implements ActionListener {

   /**
    * Public method that initializes the screen for the
    * application.
    */

   public void init() {}

   /**
    * Public method that creates application interface as well
    * as its menu and filters for reading .crd and .tri files
    * from the directory
    */

   public static void CreateInterface() {
      String directory=null;
      FilenameFilter trifilter=null;
      FilenameFilter ptfilter=null;
      int i;

      Frame f = new Frame();
      /** Create a new instance of win() applet class
        */
      win a = new win();
      f.add(a, "Center");
      a.init();
      f.setSize(500,500);
      f.show();
      f.addWindowListener (new WindowAdapter() {
         public void windowClosing (WindowEvent e) { System.exit(0);}
      });

      MenuBar menubar = new MenuBar();
      f.setMenuBar(menubar);

      Menu file = new Menu("File");
      Menu ccsystems = new Menu("CC Systems");

      Menu subpoints = new Menu("Display Points from File ");
```

```
Menu subtriple = new Menu("Display Triples from File  ");

menubar.add(file);
menubar.add(ccsystems);

// Record all files in current directory to dir list
directory = System.getProperty("user.dir");
File dir = new File(directory);

// filter the dir list to select only .tri files
trifilter = new FilenameFilter() {
  public boolean accept(File dir, String name) {
          if(name.endsWith(".tri"))
                  return true;
          else return false;
  }
};


ptfilter = new FilenameFilter() {
  public boolean accept(File dir, String name) {
          if(name.endsWith(".crd"))
                  return true;
          else return false;
  }
};


// copy dir list into array trpfile[]
trpfile = dir.list(trifilter);

ccfile = dir.list(ptfilter);

// add action listener for selecting certain .tri file from menu
// and set action command to selected file
for(i=0;i<trpfile.length;i++)
  subtriple.add(trpfile[i]);

subtriple.addActionListener(a);
for(i=0;i<trpfile.length;i++)
  subtriple.setActionCommand(trpfile[i]);


// add action listener for selection certain .crd file
// to read from in menu
for(i=0;i<ccfile.length;i++)
  subpoints.add(ccfile[i]);
subpoints.addActionListener(a);
for(i=0;i<ccfile.length;i++)
  subpoints.setActionCommand(ccfile[i]);


// add subpoints and subtriple submenus to CCsystems menu
ccsystems.add(subpoints);
ccsystems.add(subtriple);

MenuItem clear =  new MenuItem("Clear", new MenuShortcut(KeyEvent.VK_C))
clear.addActionListener(a);
clear.setActionCommand("clear");
file.add(clear);
```

```
    MenuItem quit = new MenuItem("Quit", new MenuShortcut(KeyEvent.VK_X));
    quit.addActionListener(a);
    quit.setActionCommand("quit");
    file.add(quit);

    MenuItem writecc = new MenuItem("Write Points", new MenuShortcut(KeyEvent.
    writecc.addActionListener(a);
    writecc.setActionCommand("writecc");
    ccsystems.add(writecc);

    MenuItem addpoint = new MenuItem("Add Points", new MenuShortcut(KeyEvent.V
    addpoint.addActionListener(a);
    addpoint.setActionCommand("addpoint");
    ccsystems.add(addpoint);

    MenuItem mvpoint = new MenuItem("Move Points", new MenuShortcut(KeyEvent.V
    mvpoint.addActionListener(a);
    mvpoint.setActionCommand("mvpoint");
    ccsystems.add(mvpoint);

}

/** Public method that calls CreateInterface method
  */
public static void main(String[] args) {
    CreateInterface();
}


/** Public method that calls various point manipulation functions
 *  upon receving an appropriate command from the menu
 */
public void actionPerformed(ActionEvent e) {
    String cmd = e.getActionCommand();
    int i;
    for(i=0;i<trpfile.length;i++) {
        if(cmd.equals(trpfile[i])) {
            readtriple(trpfile[i]);
            globaltri = trpfile[i];
        }
    }
    for(i=0;i<ccfile.length;i++)
        if(cmd.equals(ccfile[i])) {
            readcc(ccfile[i]);
            newccfile = ccfile[i];
        }

    if (cmd.equals("clear")) {
        clear();
        apoints = 0;
    }
    else if (cmd.equals("quit")) System.exit(0);
    else if (cmd.equals("writecc")) writecc();
    else if (cmd.equals("addpoint")) addpoint();
    else if (cmd.equals("mvpoint")) mvpoint();

}

/** Protected method that is called to clear application screen,
```

```
 *  or set the background to default white
 */

protected void clear() {
     Graphics g = this.getGraphics();
     g.setColor(this.getBackground());
     g.fillRect(0,0,this.getSize().width, this.getSize().height);

}


/**
 * Public method that reads the coordinate file (.crd) line-by-line.
 */

public void readcc(String ccf) {
     /* read the Coordinate file */
     int i;

     try
     {  BufferedReader is = new BufferedReader(new
              FileReader(ccf));
        BufferedReader it = new BufferedReader(new
              FileReader(ccf));
        FileTest.countLine(it);
        CoordRead1[] in = FileTest.readData(is);

        is.close();
        it.close();
     }

     catch(IOException e)
     {  System.out.print("Error: " + e);
        System.exit(1);
     }
     catch (SecurityException e)
     {
        System.out.print("Security Error: " + e);
     }
     /* end read */
     /* display the points read */
     int j;
     Graphics g = getGraphics();
     g.setColor(Color.blue);
     apoints=points;
     for(j=0;j<apoints;j++) {
             g.drawRect(lastx[j], lasty[j], 3, 3);
             g.fillRect(lastx[j], lasty[j], 3, 3);
             String s= "" + j + "(" + lastx[j] + "," + lasty[j] + ")";
             g.drawString(s,lastx[j]-1,lasty[j]-1);
     }
}


/** Public method that reads all triples (cycles) for particular point
 *  tournament from .tri file and displays them on the screen along
 *  with their intersection, or realizability space, (if any) colored
 *  in red
 */
```

```
public void readtriple(String trp) {
     int i,j;

      try
     {  BufferedReader is = new BufferedReader(new
              FileReader(trp));
        BufferedReader it = new BufferedReader(new
              FileReader(trp));
        FileTest.countTriple(it);
        TripleRead[] in = FileTest.readTripleData(is);
        is.close();
        it.close();
     }

     catch(IOException e)
     {  System.out.print("Error: " + e);
        System.exit(1);
     }
     catch (SecurityException e)
     {
        System.out.print("Security Error: " + e);
     }
     /* end read */
     /* display the triples read */


     //Polygon p[] = new Polygon()[100];
     Graphics g = getGraphics();
     g.setColor(Color.black);

     int x,y,count, ncount;
     int[] xarray=new int[100000];
     int[] yarray=new int[100000];
     count=0; ncount=0;
     Polygon p1=new Polygon();
     p1.addPoint(lastx[pa[0]],lasty[pa[0]]);
     p1.addPoint(lastx[pb[0]],lasty[pb[0]]);
     p1.addPoint(lastx[pc[0]],lasty[pc[0]]);
     g.drawPolygon(p1);

     for(x=0;x<500;x++)
        for(y=0;y<500;y++) {
             if(p1.contains(x,y)) {
                     xarray[count]=x;
                     yarray[count]=y;
                     count++;
             }
        }
     for(j=1;j<triples;j++) {
        Polygon p2= new Polygon();
        p2.addPoint(lastx[pa[j]],lasty[pa[j]]);
        p2.addPoint(lastx[pb[j]],lasty[pb[j]]);
        p2.addPoint(lastx[pc[j]],lasty[pc[j]]);
        g.drawPolygon(p2);
        ncount=0;
        for(i=0;i<count;i++) {
             if(p2.contains(xarray[i],yarray[i])) {
                     xarray[ncount]=xarray[i];
                     yarray[ncount]=yarray[i];
                     ncount++;
```

```
                }
            }
            count=ncount;
        }

        g.setColor(Color.red);
        if(count==0) {
                String msg="Point in question is not embeddable";
                g.drawString(msg,10,450);
        }
        for(i=0;i<count;i++)
                g.fillRect(xarray[i],yarray[i],1,1);


}

/** Protected method that writes coordinates of points on the interface
 *  into a .crd file.
 */
protected void writecc() {
        int i;
        CoordRead1[] coordinate = new CoordRead1[apoints];
        for(i=0;i<apoints;i++)
                coordinate[i] = new CoordRead1(lastx[i],lasty[i]);
        System.out.println("Array size: " + coordinate.length);

        if(newccfile==null)
                newccfile=apoints + ".crd";
        try
        {  PrintWriter os = new PrintWriter(new
                FileWriter(newccfile));
           FileTest.writeData(coordinate, os);
           os.close();
        }
        catch(IOException e)
        {  System.out.print("Error: " + e);
           System.exit(1);
        }

        catch (SecurityException e)
        {
           System.out.print("Security Error: " + e);
        }
}

/** Protected method that adds points to application interface upon
 *  mouse click
 */

protected void addpoint() {
   addMouseListener (new MouseAdapter() {

   public void mouseClicked(MouseEvent e) {

     Graphics g = getGraphics();
     // set the new point color to red
     g.setColor(Color.red);

     // get the coordinate locations
     lastx[apoints]=e.getX();
```

```
      lasty[apoints]=e.getY();

      g.drawRect(lastx[apoints], lasty[apoints], 3, 3);
      g.fillRect(lastx[apoints], lasty[apoints], 3, 3);
      String s= "" + apoints;
      g.drawString(s,lastx[apoints]-1,lasty[apoints]-1);
      System.out.println(lastx[apoints] + " " + lasty[apoints] + " " + apoints
      apoints++;

      }
    });
 }

/** Protected method that moves the point from location where mouse
 *  is pressed to location where mouse is released. This method
 *  uses mousePressed, mouseReleased features of MouseListener
 */

 protected void mvpoint() {
   addMouseListener (new MouseAdapter() {
     public void mousePressed(MouseEvent ee) {
       int x = ee.getX();
       int y = ee.getY();
       int i;
       flag=0;
       for (i=0;i<apoints;i++) {
          if((x-lastx[i])>=-5 && (x-łastx[i])<=5 &&
             (y-lasty[i])>=-5 && (y-lasty[i])<=5) {
              System.out.println("You clicked on point " + i);
              pnt=i;
              flag=1;
              x=-100;
              y=-100;
              break;
       }


    }
}
public void mouseReleased(MouseEvent eee) {
    Graphics g = getGraphics();
    g.setColor(Color.red);
    int x = eee.getX();
    int y = eee.getY();
    int i;
    if (flag==1) {
    lastx[pnt]=x;
    lasty[pnt]=y;
    clear();
    g.setColor(Color.red);
    for(i=0;i<apoints;i++){
       g.drawRect(lastx[i], lasty[i], 3, 3);
       g.fillRect(lastx[i], lasty[i], 3, 3);
       String s= "" + i;
       g.drawString(s,lastx[i]-1,lasty[i]-1);
    }
    g.setColor(Color.blue);
    if(globaltri!=null)
       readtriple(globaltri);
```

```
    }
  });
}

/** Protected array for x coordinates of points
 */
protected static int[] lastx=new int[100];
/** Protected array for y coordinates of points
 */
protected static int[] lasty=new int[100];

/** Public counter that holds total number of points
 *  in input .crd file
 */
public static int points;

/** Public counter that holds total number of points
 *  including those added to the system
 */
public static int apoints;

/** Protected static array that contains first points in
 *  non-transitive tournament triples (read from .tri file)
 */
protected static int[] pa=new int[100];
/** Protected static array that contains second points in
 *  non-transitive tournament triples (read from .tri file)
 */
protected static int[] pb=new int[100];
/** Protected static array that contains third points in
 *  non-transitive tournament triples (read from .tri file)
 */
protected static int[] pc=new int[100];
/** Public counter for all triples in .tri file
 */
public static int triples;
/** Public string array that contains the list of all .tri
 *  files in current directory. This list is displayed in
 *  the menu for triple files.
 */
public static String[] trpfile = new String[100];
/**  Public string array that contains the list of all .crd
 *  files in current directory. This list is displayed in
 *  the menu for CC-systems.
 */
public static String[] ccfile = new String[10];

/** Public string that contains the name of .crd file
 */
public static String newccfile;
/** Public string that contains the name of .tri file
 */
public static String globaltri;
/** Public flaq that is raised (set to 1) upon the mouse
 *  click on already existing point. This feature is used
 *  in point movement functions
 */
public static int flag;
```

```
/** Public field that catches the point (label) upon the mouse
 *  click and flag setting
 */
public static int pnt;
```

jjh3710

Request: grad_lw1-904 from idaho

Options: flist='CoordRead1.java'

Title: CoordRead1.java

#####
idaho

Sun Jul 25 18:36:45 EDT 1999

```
********** Option Summary **********
 (See  "man net_ljx000" for details)

auto (default),   postscript,  pcl, hpgl2, hpgl2_p, raw, relay
manual, tray1, tray2, tray3, bin1, bin2,     mtype<media type>
legal, letter, exec, ledger/11x17       A3, A4, A5, A6, B5-ISO
com10, C5, DL, monarch           PostCardS, PostCard/PostCardD
B4-JIS, B5-JIS, B6-JIS               topaz, yb, nb, job, nojob
dpi#, fuser<fusing mode>              simplex, duplex, hduplex
2up, 2+,4up(pcl-hpux only), portrait, landscape, quality<mode>
srbb#, srb#, sre#,         tondensity<mode>, econo# (#=on/off)
For PS                                          ascii, ps1, ps2
For PCL      text, ln66, stye#, italic, condensed, condensedi
c, 10, 12, lpi#  height#,  weight#, medium, bold, ebold, type#
```

```
/** CoordRead1.java
  * Copyright @1998-1999 Javid Huseynov
  */

import java.io.*;
import java.util.*;
import corejava.*;

public class CoordRead1
{
   /** Public method CoordRead1 called for assigning
     * coordinates x and y.
     */
   public CoordRead1(int n, int s)
   {  x = n;
      y = s;
   }


   public CoordRead1() {}


   /** Public method that reads the data line-by-line
     * from .crd file, where coordinates are delimited
     * by comma.
     * This method uses Format method defined in CoreJava
     * library for formatting the input/output data.
     */
   public void readData(BufferedReader is) throws IOException
   {  String s = is.readLine();
      StringTokenizer t = new StringTokenizer(s, ",");
      coordx = Format.atoi(t.nextToken());
      coordy = Format.atoi(t.nextToken());
      if (coordy<0)
        coordy = coordy*(-1);
   }

   /** Public method that writes the data line-by-line
     * into .crd file, and delimits coordinates by comma.
     * This method uses Format method defined in CoreJava
     * library for formatting the input/output data.
     */

   public void writeData(PrintWriter os) throws IOException
   {  Format.print(os, "%d,", x);
      Format.print(os, "-%d\n", y);
   }

   /** Private variable for x coordinate written into .crd file
     */
   private int x;
   /** Private variable for y coordinate written into .crd file
     */
   private int y;
   /** Protected variable for x coordinate read from .crd file
     */
   protected int coordx;
   /** Protected variable for y coordinate read from .crd file
     */
   protected int coordy;
```

jjh3710

Request: grad_lw1-905 from idaho

Options: flist='FileTest.java'

Title: FileTest.java

#####
idaho

Sun Jul 25 18:37:08 EDT 1999

```
********** Option Summary **********
 (See  "man net_ljx000" for details)

auto (default),   postscript,  pcl, hpgl2, hpgl2_p, raw, relay
manual, tray1, tray2, tray3, bin1, bin2,      mtype<media type>
legal, letter, exec, ledger/11x17       A3, A4, A5, A6, B5-ISO
com10, C5, DL, monarch           PostCardS, PostCard/PostCardD
B4-JIS, B5-JIS, B6-JIS               topaz, yb, nb, job, nojob
dpi#, fuser<fusing mode>              simplex, duplex, hduplex
2up, 2+,4up(pcl-hpux only), portrait, landscape, quality<mode>
srbb#, srb#, sre#,         tondensity<mode>, econo# (#=on/off)
For PS                                          ascii, ps1, ps2
For PCL .     text, ln66, stye#, italic, condensed, condensedi
c, 10, 12, lpi#  height#,  weight#, medium, bold, ebold, type#
```

```
/** FileTest.java
  * Copyright @1998-1999 Javid Huseynov
  */

import java.io.*;
import java.util.*;
import corejava.*;

/** Public class FileTest invoked to read data
  * from .crd or .tri file or to write data into
  * .crd file.
  */

public class FileTest
{
   // method called to count the number of lines
   // in .crd file
   static void countLine(BufferedReader it)
        throws IOException
   {
      String dum;
      while ((dum=it.readLine())!=null) linecount++;

   }

   // method called to read the data from .crd file
   static CoordRead1[] readData(BufferedReader is)
      throws IOException
   {
      win.points=linecount;
      CoordRead1[] e = new CoordRead1[linecount];
      int i;
      for (i = 0; i < linecount; i++)
      {  e[i] = new CoordRead1();
         e[i].readData(is);
         win.lastx[i]=e[i].coordx;
         win.lasty[i]=e[i].coordy;
      }
      linecount=0;
      return e;

   }

   // method called to count the number of lines in
   // in .tri file
   static void countTriple(BufferedReader it)
        throws IOException
   {
      String dum;
      while ((dum=it.readLine())!=null) linecount++;

   }

   // method invoked to read data from .tri file
   static TripleRead[] readTripleData(BufferedReader is)
      throws IOException
   {
      //int n = Format.atoi(is.readLine());
      win.triples=linecount;
      TripleRead[] ee = new TripleRead[linecount];
```

```
   for (i=0;i<linecount;i++)
   {
     ee[i] = new TripleRead();
     ee[i].readTripleData(is);
     win.pa[i]=ee[i].pointa;
     win.pb[i]=ee[i].pointb;
     win.pc[i]=ee[i].pointc;
   }
   linecount=0;
   return ee;
}

// method invoked to write data into .crd file
static void writeData(CoordRead1[] e, PrintWriter os)
   throws IOException
{
   int i;
   for (i = 0; i < e.length; i++)
      e[i].writeData(os);
}
/** Public variable that contains the number of lines
  * in a given .tri or .crd file
  */
 public static int linecount;
```

```
      for (i=0;i<linecount;i++)
      {
        ee[i] = new TripleRead();
        ee[i].readTripleData(is);
        win.pa[i]=ee[i].pointa;
        win.pb[i]=ee[i].pointb;
        win.pc[i]=ee[i].pointc;
      }
      linecount=0;
      return ee;
   }

   // method invoked to write data into .crd file
   static void writeData(CoordRead1[] e, PrintWriter os)
      throws IOException
   {
      int i;
      for (i = 0; i < e.length; i++)
         e[i].writeData(os);
   }
   /** Public variable that contains the number of lines
     * in a given .tri or .crd file
     */
    public static int linecount;
}
```