A DYNAMIC SETUP FOR ELEMENTARY GEOMETRY

JÜRGEN RICHTER-GEBERT AND ULRICH H. KORTENKAMP

Abstract. In this article we survey the theoretical background that is required to build a consistent and continuous setup of *dynamic elementary geometry*. Unlike in static elementary geometry in dynamic elementary geometry the elements of a construction are allowed to move around as long as the geometric constraints intended by the construction are not violated. A typical problem in such a scenario is to resolve ambiguous situations that arise from geometric operations like intersecting a circle and a line. After introducing a formal framework for dealing with dynamic geometric constructions, we will demonstrate that a suitable resolution of these ambiguities requires the consideration of complex projective spaces. We will discuss several aspects where one can benefit from such a rather general approach. Finally, we will sketch some proofs that show that several fundamental algorithmic problems arising in such a context are NP-hard or even harder.

1. Introduction

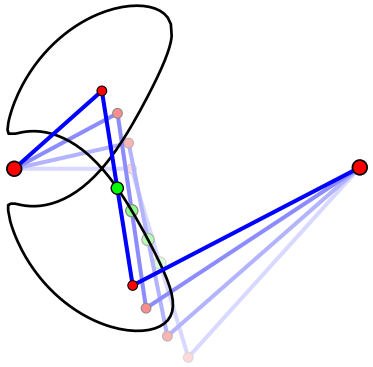
Computational Geometry very often focuses on static problems, like computing the convex hull or Voronoi complex of a given set of points. Fundamentally new questions arise when the objects under consideration are no longer static, but may move around with respect to certain geometric constraints. This scenario is not unusual, for instance every mechanism can be considered as a dynamic geometric entity.

In this article we focus on the new areas of problems that arise from genuinely dynamic effects. Constructions from elementary geometry play a crucial role in this context, since they form first natural instances of non-trivial examples where it is reasonable to study the dynamic behavior. In particular, we study computational aspects that arise if one wants to implement a *Dynamic Geometry System* (DGS) where constructions can be graphically performed by a sequence of mouse clicks. A DGS is usually equipped with a so called "drag mode" that allows – after the construction is completed – to pick a free element with the mouse, drag it, and watch the movement of the entire construction following the motion according to the geometric constraints of the construction. In the best case a DGS could be considered as a generic visualization tool for elementary geometric configurations. The research that led to the results presented here was motivated by the desire (and the actual work) of implementing a concrete software package for doing dynamic geometry on a computer [11, 12].

We start with an informal description of what we consider to be *dynamic geometry*. Imagine any construction of elementary geometry (say a ruler and compass construction of the midpoint of two points A and B). It consists of certain *free* elements (the points A and B) and certain *dependent* elements whose position is determined by the position of the free elements. Each specific picture of such a construction is a snapshot taken from the continuum of all possible drawings for all

possible locations of the free elements. By moving the free elements we can *walk* continuously from one instance of the construction to another one. During such a walk a continuous motion of the free elements should be reflected by in a continuous movement of the dependent elements.

One of the most fundamental problems for the "drag mode" arises when one considers *one* point of an intersection of a line and a circle and allows the line to be moved around. Since the point of intersection is not unique, a computer program that visualizes the movement has to decide for every "discrete snapshot" which of the two intersection points is meant. If this decision is not made correctly a "path-jumping" of the point may occur (while the line performs just a tiny movement the position of the intersection may suddenly jump from one possible place to the other).



Geometric locus under the motion of a "three bar linkage". The use of complex path tracing generates reliably complete real branches of algebraic curves. The orientation heuristics that are usually used by other software can only generate partial loci.

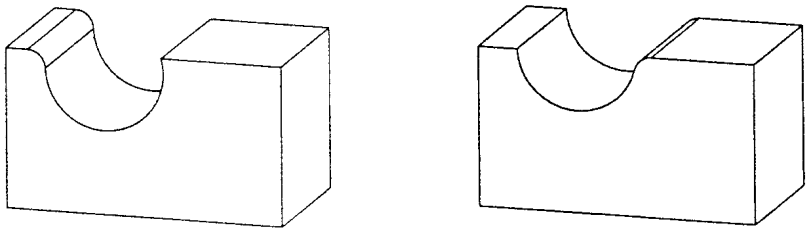
A careful analysis of the situation shows that for a satisfactory resolution of the problem one has to embed the configuration in an ambient complex projective space. One even has to take monodromy effects and underlying Riemann surfaces into account. We will later on explain how such a complexified setup can be used to obtain continuity of the dependent elements. For this we first specify how a desired continuous behavior of a dynamic geometry system should look like. After this we show how embedding the entire configuration in the ambient complex space suddenly allows to avoid degenerate or singular situations. By choosing "complex detours" for the input parameters we manage to reliably follow the different branches of ambiguous operations. The purpose of the article is to survey these results and not to give a rigorous treatment. For a more elaborate treatment see [6, 13].

After this survey of the theoretical background, we will investigate the algorithmic complexity of "making the right decisions". It will turn out that even in very weak versions this problem is NP-hard. In some stronger versions it is PSPACE hard or even undecidable. In particular one can prove that ...

- ... it is in general PSPACE-hard to decide whether two instances of the same construction can be continuously deformed into each other by moving the base elements along a real path.
- ... it is NP-hard to calculate the position of the dependent elements after a specific move of a free element.

Detailed proves of these results can be found in [13].

Although the results of this article arose from the study of configuration spaces of elementary geometric constructions they are naturally related to many other setups in the area of geometry. Among those are the study of configuration spaces of mechanical linkages [2, 5], realization spaces of oriented matroids [8, 1, 9, 16] and polytopes [10], and the piano movers problem (with possibly many pianos) [4, 15]. Our complexity results are partially generalizations and strengthenings of known complexity results in these areas.



Jumping elements are also a typical problem in parametric CAD. In this example (taken from Hoffmann [3]) a hole was drilled on the boundary of a block, and one of the edges was beveled. After moving the hole the bevel jumps from one edge to the other.

Besides the narrow context of dynamic geometry our results are relevant for all areas where geometric objects are moved around under certain geometric constraints, like robotics, parametric CAD [3], virtual reality, or computational kinematics. Our results imply that many problems of these areas are computationally difficult. A typical problem of this kind arises in parametric CAD and is know as the *persistent naming problem* [3]. It asks for reasonable generic algorithms that allow to maintain the so called "design intend" under the continuous change of the controlling parameters.

2. Geometric straight line programs

We restrict ourselves to the following particularly simple scenario, which arises in the context of interactive geometry software: *a dynamic setup for elementary geometry.* Nonetheless, we want to emphasize the the underlying methods apply also to much more general contexts.

Large parts of elementary geometry are based on the theory of *ruler and compass constructions.* Such constructions are usually done by first drawing a set of "free points" in the plane and then proceeding by adding new objects with operations like: "join of two points", "intersection", "circle given by midpoint and perimeter point". We formalize constructions that use these operations by the concept of *geometric straight line programs.*

We assume that the objects are given by suitable parameters (coordinates). A geometric straight line program (GSP) is a sequence of program statements, where each statement describes the position of a new elementary object. The operations we allow are:

`L=Join(P1,P2)`	Line `L` is the join of points `P1` and `P2`
`P=Meet(L1,L2)`	Point `P` is the meet of lines `L1` and `L2`
`C=Circle(M,P)`	Circle `P` is the a circle with center `M` through `P`
`P=FreePoint`	Point `P` is at random position (x,y)
`P=IntersectionCL(C,L)`	Point `P` is the intersection of line `L` and circle `C`
`P=IntersectionCC(C1,C2)`	Point `P` is the intersection of the circles `C1` and `C2`
`L=AngularBisector(L1,L2)`	Line `L` is the angular bisector of the lines `L1` and `L2`

The first three of these operations produce a unique element. The last three operations have an intrinsic ambiguity: A line and a circle, or two circles, can have more than one intersection; two lines have two angular bisectors. It can even happen that a line and a circle do not intersect at all. So, in general a GSP does not describe a *unique* geometric situation. One may even "get stuck" during the execution of a GSP if an intersection does not exist. However, for a given geometric configuration of points, lines and circles it is easy to check whether it is compatible the definition of a given GSP. Such a configuration is then called an *instance* of the GSP. It is clear that for every GSP and for each choice of concrete values for the coordinates of the free points, there are at most finitely many possible compatible instances, since this number is bounded by 2^n where n is the number of ambiguous choices.

A rigorous definition of *GSPs* and *instances* can be found in [6, 13]. The crucial point for such a definition is that each statement of a GSP represents a *relation* between input variables and the output variables rather than an *assignment operation*. This allows for the necessary ambiguities to give a proper definition of operations like the intersection of circle and a line. Let us consider this operation a little closer. Let $\mathcal{P}, \mathcal{L}, \mathcal{C}$ denote the spaces of points, lines and circles, respectively[1]. The operation $p = \texttt{IntersectionCL}(c, l)$ is represented by a ternary relation $R_{\texttt{IntCL}} \in (\mathcal{C} \times \mathcal{L}) \times \mathcal{P}$. A triple (c, l, p) of a circle c a line l and a point p is in $R_{\texttt{IntCL}}$ if and only if p is an intersection of c and l. For a circle c and a line l there may in general be two points p_1 and p_2 with $(c, l, p_1) \in R_{\texttt{IntCL}}$ and $(c, l, p_2) \in R_{\texttt{IntCL}}$. On a semantic level we may call c and l the *input* of the operation and p_1 and p_2 the *output*. To each statement of the GSP we may associate a corresponding *type* (point, line or circle) of the output element. If we restrict ourselves to objects with real coordinates a line and a circle may have no intersection at all. In other words there may be $c \in \mathcal{C}$ and $l \in \mathcal{L}$ with $\{p \in \mathcal{P} \mid (c, l, p) \in R_{\texttt{IntCL}}\} = \emptyset$. We have to distinguish three major kinds of relations:

- *deterministic operations,* where the position of the output element is uniquely determined by the input elements (like *join*, and *meet*),
- *non-deterministic operations,* where there is a finite number of possible positions for the output elements (like *intersection-circle-line*, *intersection-circle-circle*, and *angular bisector*),
- *the "free"-operation* is a special operation to introduce free points that underly no further restriction. Considered as a relation the operation is simply a unary operation that admits all points $R_{\texttt{Free}} = \mathcal{P}$.

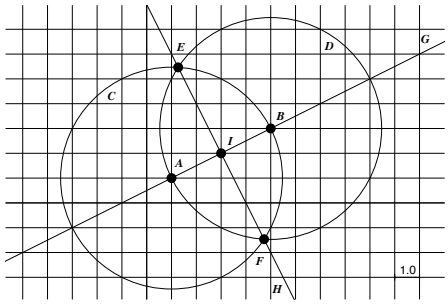
In the rigorous definition of GSPs furthermore care has to be taken to exclude degenerate situations like taking the join of two coincident points. Such situations are called *non-admissible* However, we neglect these technicalities here. An *instance* of a GSP is an assignment of actual values to the variables of a GSP such that all

[1]We will later on specify the exact mathematical content of these spaces

relations are satisfied. The type of each variable has to match the type of the corresponding program statement.

Example: The following GSP encodes a construction of the midpoint of two points A and B.

```
1:  A=FreePoint;
2:  B=FreePoint;
3:  C=Circle(A,B);
4:  D=Circle(B,A);
5:  E=IntersectionCC(C,D);
6:  F=IntersectionCC(C,D);
7:  G=Join(A,B);
8:  H=Join(E,F);
9:  I=Meet(G,H);
```



The picture shows one possible instance for $A = (1, 1)$ and $B = (5, 3)$. In this GSP points E and F have the same definition, namely being the intersection of the two circles. Only for the "right" choices you will obtain an actual instance of the GSP in which the final point is indeed the midpoint of A and B.

3. The problem of continuity

We now study a GSP in a dynamic setup. We want to model the situation that one (or many) free elements are moved from one position to another. We specify how a desirable behavior of the dependent elements under such a motion should look like. (To have a more vivid image of this problem assume you constructed the above picture with an interactive geometry program, how should the dependent elements behave, when you move point A or point B?)

For a more formal treatment consider the coordinates of the free points parametrized by a single parameter $\lambda \in [0, 1]$. For each position of the free elements we want to single out a *reasonable* choice of the dependent elements. For a given GSP $\mathbf{P}$ we consider only those selections of dependent elements that lead to admissible instances of $\mathbf{P}$, i.e. all elements satisfy all relations specified by the program and no "degenerate" situation arises in the technical sense that we have not closer specified.

It is clear that the only freedom for the choice of dependent elements comes from the ambiguities of non-deterministic operations. A desirable behavior would be the following:

> *"While the free points move continuously all dependent objects move continuously as well." In other words: "The coordinates of all elements are continuous functions in λ."*

More formally let $\mathbf{P}$ be a specific GSP with n lines. Assume that the spaces $\mathcal{P}, \mathcal{L}$ and $\mathcal{C}$ are equipped with a suitable topology. An instance $(v_1, v_2 \ldots, v_n)$ of $\mathbf{P}$ can be considered as an element of $\mathcal{T}_1 \times \mathcal{T}_2 \times \cdots \times \mathcal{T}_n$, where for all i we have $\mathcal{T}_i \in \{\mathcal{P}, \mathcal{L}, \mathcal{C}\}$ according to the output type of line i in $\mathbf{P}$. W.l.o.g. we may assume that the first k operations of are the `Free`-operations. A continuous movement of the free elements is given by continuous functions $v_i : [0, 1] \to \mathcal{T}_i$ for $i = 1, \ldots, k$. Our above requirement of continuity translates to the problem of generating continuous functions $v_i : [0, 1] \to \mathcal{T}_i$ for $i = k+1, \ldots, n$ such that for each $t \in [0, 1]$ the sequence $(v_1(t), v_2(t) \ldots, v_n(t))$ is an admissible instance of $\mathbf{P}$.

At first sight it is not clear whether this requirement is satisfiable at all (compare [7]). In fact, all geometry systems and programs for parametric CAD that are currently available suffer from non-continuous behavior of dependent elements; while you move a free point it may happen that parts of the construction jump from one place to another. In particular, we must find a way to deal with the problem of vanishing intersections.

4. Complex projective geometry

To fulfill the continuity requirements we first need to fix a topology. We first consider the space $\mathcal{P}$ of all points. A first rough approach would be to identify $\mathcal{P}$ with the euclidean plane. However, this would not be suitable for our requirements. We will have to view the Euclidean plane as a subset of the projective plane. This gives us "points at infinity" and the desired topology is induced by the topology of the manifold structure that underlies the projective plane. In particular a *point can move to infinity and can "continuously" come back from the opposite side of the (embedded euclidean) plane.* On an algebraic level such a point may be represented by homogeneous coordinates (x, y, z) where vectors that differ only by a scalar multiple are identified. For finite points (i.e. those with $z \neq 0$) the original position in the euclidean plane can always be recovered as $(x/z, y/z)$.

The next and more important enlargement of the setup comes from embedding the whole situation in complex space. For this we simply assume that the coordinates of the objects may take also values in the field of complex numbers and study *complex two-dimensional projective geometry.* Since every complex number can be described by two real numbers this space has real dimension four. Nevertheless, the Euclidean plane can still be found as a substructure of this space.

Compared to the real setup complex calculations have a great advantage: Intersections never (!) vanish. Even if a line and a circle do not intersect in the real Euclidean plane, it is still reasonable to consider intersection points with complex coordinates, since the coordinates are just solutions of suitable quadratic equations.

Roughly speaking, we identify the space $\mathcal{P}$ with $\mathbb{CP}^2$ the complex projective plane equipped with the natural topology. In a similar way we identify $\mathcal{L}$ (the space of lines) with the corresponding dual space of $\mathbb{CP}^2$. We may think of a line given by the parameters (a, b, c) as the solution set of the equation $ax+by+cz = 0$. Again the parameters may also become complex numbers and parameter vectors that only differ by scalar multiples have to be identified.

The space $\mathcal{C}$ of all circles consists of all complex 4-tuples (a, b, c, d) (with scalar multiples identified). The circle represented by the parameters (a, b, c, d) is the solution set of the homogeneous circle equation $ax^2 + ay^2 + bxz + cyz + dz^2 = 0$.

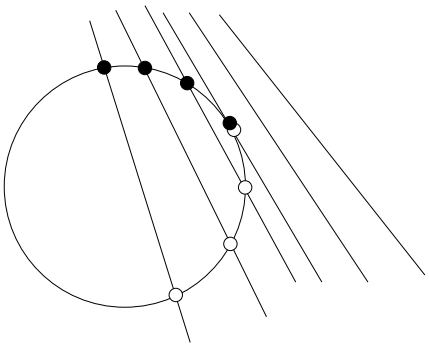
5. Moving in complex spaces

Assuming that the free points perform a continuous motion there is an easy strategy of dealing with multiple intersections: *Consider both intersections as individual objects and trace their paths through complex space.* As long as the intersection points do not coincide one can easily tell them apart and (in a continuous model of computation) trace them as individual objects.

However, at first sight there seems to be a major obstacle for making this strategy a reasonable algorithm. Consider the simple case of intersecting a line and a circle. Assume that the circle is at a fixed position and that the line is moved from a

position at which the two objects intersect to a position where they do not intersect (in real space). Mark the two intersections in the start situation by a black and a white dot, and try to trace them during the movement. Even if we consider the vanishing intersections as still existent in complex space there will be a tangent situation where the black and the white point do coincide. At first sight there seems to be no reasonable choice of how to associate the colors "black" and "white" to the complex intersection points that arise *after* the tangent situation. In fact, there is no such preferable choice, since in the tangent situation the situation is completely symmetric.

However, there is an easy way to break this symmetry by choosing a suitable path for the movement of the line. To see this, let us consider the situation in terms of coordinates. Let the circle be the unit circle $\{(x, y, z) \mid x^2 + y^2 - z^2 = 0\}$ and let the line be a horizontal line $\{(x, y, z) \mid 0x + y - 2\lambda z = 0\}$ parametrized by λ. As λ moves from 0 to 1 the line moves from the x-axis to a horizontal position where it cuts the y-axis at $y = 2$. The intersections of the two objects have homogeneous coordinates $(\pm\sqrt{1 - 4\lambda^2}, 2\lambda, 1)$ (for the purpose of an easy "de-homogenization" we set $z = 1$, then the x and y entries represent then usual euclidean coordinates). We see that for $\lambda = 0$ we have two real solutions $(0, \pm 1, 1)$, for $\lambda = 1$ we have two complex solutions $(2, \pm\sqrt{-15}, 1)$, and for $\lambda = 1$ the two solutions coincide.



In the Euclidean plane there can be one, two or none intersections of a line and a circle. In complex space these intersections never vanish. If we can avoid "singular situations" it is always possible to trace the two intersections individually.

So, how can we avoid the situations where the intersection points coincide? For this we again take advantage of the complex setup. Let us call an instance of a GSP *non-singular* if no "double intersections" occur. If we move from one non-singular instance of a GSP to another non-singular instance of the GSP there is always a path through complex space that avoids all singular situations. This is a consequence of the fact that a non-constant analytic function can have no accumulation points as its set of roots. This means that we can always take a "complex detour" that allows us to individually trace all dependent objects of a construction. For such paths we obtain perfect continuity.

To obtain such a detour in our example we may take a reparameterization of the parameter λ according to $\lambda(t) = (\cos(t) + 1)/2 + i \cdot \sin(t)/2$ while t moves along the real segment $[0, \pi]$ the parameter $\lambda(t)$ moves from 0 to 1 along a complex semi-circle. If we parameterize our geometric configuration with this parameter, the start- and end-positions coincide with the start- and end-positions we previously had. However in between the line becomes complex and so do the two points of intersection. The important point is that along this path the two points never

coincide. Thus for this specific path one can clearly distinguish which of the points in the final position has to be "black" and which has to be "white".

In fact, for an analytic path that avoids all singularities the coordinates of the dependent objects can be expressed by analytic functions in the input parameter.

6. From discrete samples to continuous movement

How can we imply these insights to a the implementation of a dynamic geometry program? The crucial point here is that in a dynamic geometry program no explicit paths of the free elements are determined by the user. If the user picks a free point with the mouse and moves it from one position to another one gets the impression that the path of the point makes a continuous movement. However the only information the computer gets consists of a discrete set of mouse events that indicate the position of the point at a sequence of sample points. Between these sample points the program has the "freedom" to take an arbitrary path. So, as long as the sample points correspond to non-singular situation it is always possible to connect them by piecewise continuous (even analytic) paths that avoid all singular situations. (One may view the entire situation as if the mouse pointer that dragges the point escapes "off screen" to complex space between each of the sample points of a dragging action.)

Clearly, this theoretical approach still does not give a concrete algorithm how to perform this dragging and tracing of elements numerically. In fact, in Section 9 we will see that this problem is indeed intrinsically hard.

7. Real benefits

The approach above may sound far too complicated to resolve the original problem of having a dynamic setup for Euclidean geometry. However, it can be proved that as soon as we want to have continuous behavior of the dependent elements, there is only one way to make the decisions and that this choice coincides with our solution. Although the setup uses complex numbers we have several benefits in the real case. Here is a list of keywords of what becomes possible under this setup:

- All derived elements behave *analytically*:
  After an analytic path for the free elements has been chosen, the coordinates of the dependent elements can be expressed as analytic functions in λ as well. This is the case, since each of the primitive operations is expressible by an analytic function (with possibly several branches). The composition of analytic functions is again analytic.
- The solution is *unique*:
  After the path is chosen there is no more ambiguity in the system. Each element follows the unique path that we get by analytic continuation.
- There are *no jumping elements*:
  After a path is chosen at least in theory no jumps of ambiguous elements occur.
- The behavior is *globally consistent*:
  Assume that you already have a construction that has a certain dynamic behavior. It is not possible to enlarge this construction in such a way that elements that constructed later perform a jump. The system makes anticipatorily the right decisions.

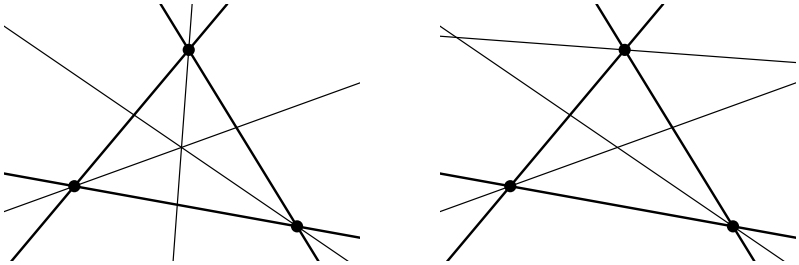
- Geometric theorems are true *once-and-forever*:
  If a certain geometric property (that can be expressed by a polynomial equation) holds in an arbitrary small neighborhood of the parameter space, then it holds in the entire parameter space. This is a consequence of the fact that an analytic function either vanishes always or only very "sporadically".
- *Randomized proving* works:
  The last fact can be used to test geometric relations on a randomized basis. The system performs a random walk within the parameter space and test whether for each choice of the parameters a conjectured relation is satisfied. By testing a large number of samples the probability of making a wrong statement can be kept arbitrary small.
- *Self exploring Loci*:
  The fact that performing a full cycle around a singular situation may cause a monodromy effect on the depending elements can be used to get animations and loci that explore the entire configuration space by generic methods.
- We get generic tools for *computational kinematics*:
  Besides the narrow field of dynamic geometry the same methods apply to areas such as computational kinematics, parametric CAD and virtual reality.

8. Randomized proving and continuity

To keep its own data structures clean, our program needs consistent information about incidences and equalities that occur in configurations. Such incidences may either be trivial consequences of the construction or arise from geometric theorems like the altitudes of a triangle meeting in a point. We actually get this information by a *randomized theorem checking* technique. Enough random instances of a configuration are generated and for each of them the conjectured incidence is checked. This is done until the program either accepts the theorem with a certain high probability or it rejects it, if a counterexample was found.

To be really reliable, the randomized theorem checking engine needs enough (!) random (!) examples. Again there arises a theoretical problem which originates from ambiguities in geometric constructions. Consider the theorem stating that *the angular bisectors of the sides of a triangle meet in a point.* Due to the intrinsic ambiguity of the angular bisector operation this sentence stated as such is not true. Consider the drawing in the picture below. It shows two valid instances of the construction: Take three points — form the three joins of any pair of them — draw the three angular bisectors of any pair of lines. In one of the drawings the chosen angular bisectors meet in the other they do not.

For the theorem checking the program does a "random walk" that stays always in the desired component of the configuration space. Staying in the correct component during this random walk again depends on consistent and continuous behavior of dependent elements.



Depending on the choices the angular bisectors of a triangle can intersect or not. Arandomizued proving algorithm has to generate only samples that lie in the "correct" (i.e. intendet) component of the configuration space. This can be achieved by performing random walks (using our continuity setup) rather than random jumps.

9. Complexity issues

Let us now briefly sketch the issues of algorithmic complexity that arise in this context (details, proofs and further results can be found in [13, 14]). Two fundamental questions can be formulated that capture the main algorithmic questions of dynamic geometry:

- **Reachability problem:** Given two instances of GSP. Is it possible to move the free points such that a first instance is smoothly deformed into a specific second one?
- **Tracing problem:** How can a dynamic geometry program decide after a move what instance to draw for the new position of the free elements?

In fact it makes an essential difference whether one allows in the reachability problem only real coordinates of the elements or also complex paths. For the real version it can be proved that (after suitable formalization) the reachability problem is in general PSPACE-hard. It is still NP-hard if one restricts oneself to constructions that only use *join*, *meet*, and *angular bisector* operations. The Tracing problem turns out to be (at least) NP-hard. We briefly sketch how the above results can be achieved. We first focus on the following result:

Theorem 9.1. *Let* $\mathbf{P}$ *be a geometric straight line program that uses at most three* angular bisector *operations and except of this only* join *and* meet *operations. Furthermore let* I_1 *and* I_2 *be two instances of* $\mathbf{P}$ *that differ only in the choice of one angular bisector. It is NP-hard to decide whether* I_1 *can be moved continuously into* I_2 *by a real continuous motion of the free points of* $\mathbf{P}$*.*

In order to sketch a proof of this theorem we describe how a reduction of the well known 3-SAT problem to the real reachability problem can be achieved. Our reduction proceeds in several steps. The first step consists of transforming 3-SAT to an algebraic setup. For this let us first formally state the 3-SAT decision problem:
3-SAT: *Let* $B = (b_1, \ldots, b_n)$ *be boolean variables, and let the literals over* B *be* $\widetilde{B} = (b_1, \ldots, b_n, \neg b_1, \ldots, \neg b_n)$*. Furthermore let* $C_1, \ldots, C_k$ *be clauses formed by disjunction of three literals from* $\widetilde{B}$*. Decide whether there is a truth assignment for* B *that satisfies all clauses* $C_1, \ldots, C_k$ *simultaneously.*

We may w.l.o.g. assume that each variable occurs at most once in each clause. We give a (polynomial time) procedure that transfers each instance of 3-SAT into

a corresponding problem concerning the roots of a multivariate polynomial. Let $b_1, \ldots, b_n$ be the boolean variables and let $C_1, \ldots, C_k$ be the clauses of a concrete 3-SAT S. To each b_i we assign a formal variable x_i. For a literal $l_i \in \{b_i, \neg b_i\}$ we set

$$f(x_i) := \begin{cases} x_i & \text{if } l_i = b_i, \\ 1 - x_i & \text{if } l_i = \neg b_i, \end{cases}$$

Assume that for each $j = 1, \ldots, k$ the clause C_j is of the form $l_r^j \vee l_s^j \vee l_t^j$ where the literal l_i^j is either b_i or $\neg b_i$. We set

$$F_j := f(l_r^j) \cdot f(l_s^j) \cdot f(l_t^j)$$

Finally we set

$$F_S = \sum_{j=1}^{k} F_j.$$

By this translation for instance the 3-SAT formula $(b_1 \vee \neg b_3 \vee b_5) \wedge (\neg b_2 \vee b_4 \vee \neg b_5)$ is translated to $(x_1 \cdot (1 - x_3) \cdot x_5) + ((1 - x_2) \cdot x_4 \cdot (1 - x_5))$. The satisfying truth assignments for S and the roots of $F(S)$ in $[0, 1]^n$ are related by the following lemma (here $[0, 1]$ denotes the closed interval between 0 and 1).

Lemma 9.2. *S has a satisfying truth assignment if and only if there are $(x_1, \ldots, x_n) \in [0, 1]^n$ with $F_S(x_1, \ldots, x_n) = 0$.*

Proof. If S has a concrete satisfying truth assignment $(b_1, \ldots, b_n) \in \{\textsc{True}, \textsc{False}\}^n$ we set

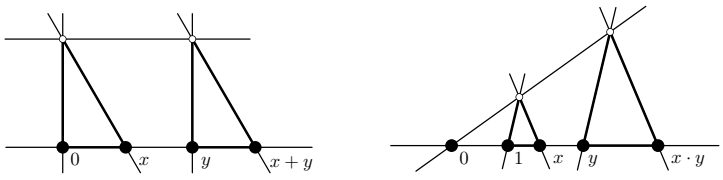
$$x_i := \begin{cases} 0 & \text{if } b_i = \textsc{True}, \\ 1 & \text{if } b_i = \textsc{False}, \end{cases}$$

Since every clause contains at least one true literal we the get that all $f_1, \ldots, f_k$ are zero. This yields that F_S is zero as well. Conversely, assume that there are values $(x_1, \ldots, x_n) \in [0, 1]^n$ such that $F_S(x_1, \ldots, x_n) = 0$. If the x_i are chosen in the interval $[0, 1]$ all f_j are non-negative. Thus if $\sum_{j=1}^{k} f_j = 0$ implies that all f_j are zero. However each f_i can only be zero if at least one of its factors is zero. By setting

$$b_i := \begin{cases} \textsc{True} & \text{if } x_i = 0, \\ \textsc{False} & \text{if } x_i \neq 0, \end{cases}$$

We get a satisfying truth assignment for S. □

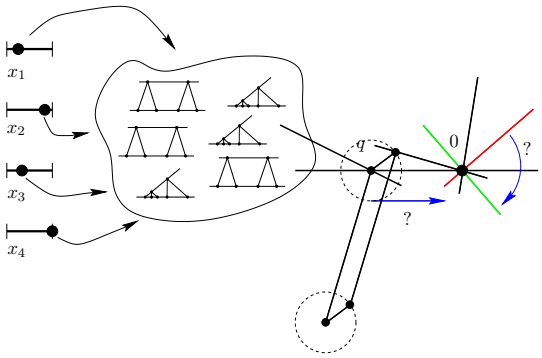
In the next step of our reduction we simulate the algebraic computation of the polynomial F_S by an elementary geometric construction. For this we take a line on which we fix positions of the points 0 and 1 to define a scale of measurement. Each point on the line corresponds then to a certain value. Multiplication and addition of values on the line can be performed by the classical *von Staudt constructions* (see picture below). (The parallelisms that occur in these construction can – after fixing a line at infinity – entirely expressed by joins and meets.)



Von Staudt constructions for addition and multiplication.

The calculation of the polynomial F_S can be decomposed into elementary arithmetic operations. The entire construction of the geometric counterpart to F_S contains n free points $x_1, \ldots, x_n$ as input variables and one dependent point q as output variable. If we restrict the position of the input points to the interval $[0, 1]$ on the computation line, we see that the point q can only be moved to the origin if the original 3-SAT problem S was satisfiable. Restricting the input points to the interval can be done by a small geometric gadget that uses Thales Theorem.

Finally, we construct a semi-free point that can move only on a small circle around the output point. We can detect whether this point can circle around the origin by an angular bisector construction. For this we join this point to the origin and form the three times iterated angular bisector with our calculation line. Schematically the construction is shown in the following picture.



Construction of a "geometric combination lock".

The whole construction forms a kind of geometric combination lock. Opening the locker corresponds to interchanging

We associate the following reachability problem to this construction. Is it possible from an arbitrary position of the input points to move in a real path such that the final angular bisector is rotated by an angle of 90° and all free points reached their initial position again. The whole construction forms a kind of geometric combination lock. Opening the lock corresponds to achieve the desired position of the angular bisector, but for this one has to know the correct positions of the code dials (the input variables). Opening the lock proceeds by first moving the dials to the right position, changing the the angular bisectors position and finally moving the dials back to the original position.

It is not difficult to prove that the corresponding reachability question is equivalent to finding a satisfying truth assignment for our original 3-SAT problem S. The argument for this goes as follows:

- The only way that the final angular bisector can make this 90° turn is that the line through the origin and through the point that is restricted to the circle around q makes a full turn.
- This is only possible if p and the origin get so close such that the circle around p contains the origin.
- This is only possible if the input points x_i can be moved to a position that corresponds to a satisfying truth assignment of S.

Thus changing the position of the final angular bisector requires that we know a satisfying truth assignment for S. This finishes our sketch of the proof of Theorem 7.1.

The other main theorem one can proof is the following.

Theorem 9.3. *Given a geometric straight line Program* $\mathbf{P}$ *that contains exactly one free point* p*. Furthermore given two instances* A *and* B *such that* p *is at position* a *in* A *and* p *is at position* b *in* B*. Let* $p(t) : [0, 1] \to [a, b]$ *be a concrete (straight) movement of* p *with* $p(0) = a$ *and* $p(1) = b$*. It is NP-hard to decide whether a continuous evaluation of* $\mathbf{P}$ *under this movement that starts at instance* A *ends up at the instance* B*.*

We very briefly sketch the idea behind the proof of this theorem. We combine our previous construction with a kind of "automatic safe cracker" that while moving the point p explores systematically all positions of the input variables, and for every position tries to change the location of the angular bisector. If the original 3-SAT instance S had a satisfying truth assignment, then the final angular bisector will have changed its position when p reached its end situation $p(1)$. Hence from the final instance of the configuration we can read of whether S had a satisfying truth assignment.

10. Remarks

More information about the software and the underlying mathematics can be found on the Cinderella Website at `http://www.cinderella.de`.

References

[1] H. Günzel, *The universal partition theorem for oriented matroids,* Discrete Comput. Geom., **19**, (1998), 521–551.

[2] J. Hopcroft, D. Joseph & S. Whitesides, *Movement problems for 2-dimensional Linkages,* SIAM J. Comput., **13**, (1984), 610–629.

[3] Ch. M. Hoffmann, *Solid Modeling,* in: J.E. Goodman & J. O'Rourke. (eds.): *Handbook of Discrete and Computational Geometry*, Lecture Notes in Mathematics **1346**, CRC Press, Boca Raton, New York, 1997, 863–880.

[4] J. Hopcroft, J.T. Schwarz & M. Sharir, *On the Complexity of Motion Planning for multiple Independent Objects; PSPACE-Hardness of the "Warehouseman's Problem",* Intern. J. Robotics Research **3**, (1984), 76–87.

[5] D. Jordan & M. Steiner, *Configuration Spaces of Mechanical Linkages,* Discrete Comput. Geom., **22**, (1999), 297–315.

[6] U. Kortenkamp, *Foundations of dynamic geometry*, PhD-thesis, ETH Zürich, 1999, `http://www.inf.fu-berlin.de/~kortenka/Papers/diss.pdf`.

[7] J.-M. Laborde, *Exploring non-euclidean geometry in a dynamic geometry environment like Cabri-géomètre*, Geometry Turned On (James King and Doris Schattschneider, eds.), MAA Notes, no. 41, The Mathematical Association of America, 1997, 185–191.

[8] N.E. Mnëv, *The universality theorems on the classification problem of configuration varieties and convex polytopes varieties,* in: Viro, O.Ya. (ed.): *Topology and Geometry — Rohlin Seminar*, Lecture Notes in Mathematics **1346**, Springer, Heidelberg 1988, 527–544.

[9] J. Richter-Gebert, *The Universality theorems for oriented matroids and polytopes,* Contemporary Mathematics **223**, (1999), 269–292.

[10] J. Richter-Gebert, *Realization Spaces of Polytopes,* Lecture Notes in Mathematics **1643**, Springer, Heidelberg 1996.

[11] J. Richter-Gebert & U. Kortenkamp, *Cinderella - The interactive geometry software*, Springer 1999; see also `http://www.cinderella.de`.

[12] J. Richter-Gebert & U. Kortenkamp, *Cinderella - Die interaktive Geometriesoftware*, HEUREKA Klett, 2000.

[13] J. Richter-Gebert & U. Kortenkamp, *Complexity issues in dynamic geometry*, submitted, manuskript available on request.
[14] J. Richter-Gebert & U. Kortenkamp, *Decision complexity in dynamic geometry*, to appear, manuskript available on request.
[15] J. Reif, *Complexity of the movers' problem and generalizations,* Proc. 20th IEEE conf. on Foundations of Comp. Sci., Long beach, Calif.: IEEE Computer Society, 1979, 421–427.
[16] P. Shor, *Stretchability of pseudolines is NP–hard,* in: *Applied Geometry and Discrete Mathematics – The Victor Klee Festschrift* (P. Gritzmann, B. Sturmfels, eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Amer. Math. Soc., Providence, RI, **4**, (1991), 531–554.

Jürgen Richter-Gebert
ETH Zürich
Inst. for Theoretical Computer Science
ETH Zentrum
CH-8092 Zürich
Switzerland
e-mail: richter@inf.ethz.ch

Ulrich H. Kortenkamp
FU Berlin
Institut für Informatik
Takustraße 9
D-14195 Berlin
Germany
e-mail: kortenkamp@inf.fu-berlin.de