Computer-Aided Design 56 (2014) 22–33

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

An algebraic taxonomy for locus computation in dynamic geometry[☆]

Miguel Á. Abánades[a,*], Francisco Botana[b], Antonio Montes[c], Tomás Recio[d]

[a] CES Felipe II, Universidad Complutense de Madrid, Spain
[b] Depto. de Matemática Aplicada I, Universidad de Vigo, Spain
[c] Universitat Politècnica de Catalunya, Spain
[d] Depto. de Matemáticas, Estadística y Computación, Universidad de Cantabria, Spain

HIGHLIGHTS

- A taxonomy for locus computation in dynamic geometry is proposed.
- An algorithm for automatic locus computation using the Gröbner Cover is described.
- A prototype of web application implementing the main algorithm is provided.

ARTICLE INFO

Article history:
Received 13 February 2014
Accepted 14 June 2014

Keywords:
Dynamic geometry
Locus computation
Parametric polynomial systems
GröbnerCover algorithm

ABSTRACT

The automatic determination of geometric loci is an important issue in Dynamic Geometry. In Dynamic Geometry systems, it is often the case that locus determination is purely graphical, producing an output that is not robust enough and not reusable by the given software. Parts of the true locus may be missing, and extraneous objects can be appended to it as side products of the locus determination process. In this paper, we propose a new method for the computation, in dynamic geometry, of a locus defined by algebraic conditions. It provides an analytic, exact description of the sought locus, making possible a subsequent precise manipulation of this object by the system. Moreover, a complete taxonomy, cataloging the potentially different kinds of geometric objects arising from the locus computation procedure, is introduced, allowing to easily discriminate these objects as either extraneous or as pertaining to the sought locus. Our technique takes profit of the recently developed GröbnerCover algorithm. The taxonomy introduced can be generalized to higher dimensions, but we focus on 2-dimensional loci for classical reasons. The proposed method is illustrated through a web-based application prototype, showing that it has reached enough maturity as to be considered a practical option to be included in the next generation of dynamic geometry environments.

1. Introduction

In general, a geometric locus is a set of points satisfying some condition. For instance, the set of points A at a given distance d to a specific point C is the circle centered at C of radius d. For another simple example of a different kind, let c be a given circle with center C, let Q be an arbitrary point on the circle and consider the locus of midpoints P of the segments CQ, as Q glides along the circle c.

In Dynamic Geometry (DG), the term locus generally refers to loci of this second kind: i.e. to the trajectory determined by the different positions of a point (the *tracer*, as point P above), corresponding to the different instances of the construction determined by the different positions of a second point (the *mover*, such as point Q above) along the path where it is constrained. This is the case for the first standard DG systems developed in the late 1980s (such as Cabri [1] and The Geometer's Sketchpad [2]), but it is also true for the more recent ones, such as GeoGebra [3] or Java Geometry Expert [4].

Note that even simple DG constructions can involve two-dimensional loci. Consider, for instance, two circles, each one with a point moving on it. While the locus of their midpoint is a circular region, no current DG environment would return such set, since the corresponding *locus* command cannot manage two independent mover points. Thus, our discussion is restricted to loci in constructions with exactly one degree of freedom. This approach includes standard DG loci, and also constructions currently not

☆ This paper has been recommended for acceptance by Ralph Martin.
* Corresponding author.
E-mail addresses: abanades@ajz.ucm.es, mabanades70@gmail.com (M.Á. Abánades).

http://dx.doi.org/10.1016/j.cad.2014.06.008
0010-4485/© 2014 Elsevier Ltd. All rights reserved.

covered by the locus function in interactive environments, such as a circle computed through its standard definition as the locus set of points at a given distance to a given point.

There is a wide consensus among DG developers to consider locus computation as one of the five basic properties in the DG paradigm (together with dynamic transformation, measurement, free dragging and animation; see, for instance [5]).

In Section 2 we review the different approaches followed by DG environments to address the computation of loci. We discuss the traditional numeric method, as well as some improvements aimed at providing a more detailed knowledge of loci, including those coming from the field of symbolic computation. Limitations and failures of these methods are emphasized, with a view towards providing a benchmark to test the performance of our method, which is illustrated by the examples in Section 5.

Our approach considers a given locus as a certain subset of the projection set of an associated algebraic variety (see Section 3). Many methods have been developed to obtain the Zariski closure of such projections (Gröbner bases, characteristic sets, discriminant varieties, border polynomials, …). Our proposal takes advantage of the specific features found in the recently developed GröbnerCover algorithm (see Section 4) to

- compute the projection set, yielding a constructible set (and not just the algebraic set given by the Zariski closure), and
- automatically discriminate the relevant components, within the constructible set, containing the given locus.

The last property is achieved by developing an elaborated taxonomy for different pieces of the aforementioned projection set, and by algorithmically assigning to each one the corresponding label (see Section 3).

Following this taxonomy, we establish a protocol that yields a faithful symbolic description of a given locus in terms of constructible sets, collecting pieces of the projection set featuring 'good' labels. In Section 4, a software tool implementing our proposal is described. Finally, several examples illustrating the method are discussed in detail in Section 5.

The provided examples show that our method overcomes limitations found in previous proposals, and also that it allows the computation of generalized loci in the sense of [6], see Section 5.4.

2. Locus computation in dynamic geometry: approaches and limitations

Sutherland's Sketchpad [7], one of the first graphic interfaces, developed half a century ago, already included some key concepts in the paradigm of dynamic geometry. Most remarkably, it introduced the use of a *light pen* to select and dynamically interact with geometric objects displayed on a screen, in a way almost identical to mouse dragging (or finger dragging on touchscreens).

In particular, for locus computation, the approach followed by Sketchpad is basically the same as the one present in current standard DG systems, namely, it consists of building a set of sample locus points (a *time exposure* in Sutherland's words). Below, we briefly describe this 'traditional' method, as well as some attempts towards its improvement.

2.1. The traditional method: loci by sampling

The standard approach followed by DG systems to obtain loci is based on sampling the path of the mover. Each sample point determines a position for the tracer, and hence a point in the locus. This set of locus points can then be shown as a collection of pixels on the screen, suggesting the sought locus.

On this list of locus points, most DG systems apply some simple heuristics to join contiguous points, in order to return the locus as a continuous, (usually) one-dimensional object, on the screen. A first difficulty arises here, because the applied heuristics can return aberrant loci, since small modifications in a construction can sometimes produce significant changes of position in dependent objects (see [8] for details).

A second problem, regardless of whether the locus is returned as a sequence of points or as a continuous curve, is the fact that the locus is simply a graphical representation, preventing the system from working any further with such output. For instance, since the equation of a curve (as a locus) is not available if this locus is obtained by the traditional method, computing its tangent at a point becomes many times very imprecise, if not impossible altogether.[1]

Another difficulty emerging from this numerical method is found when trying to obtain the intersection of a locus with another element in the construction. Although various solutions have been introduced in different systems, these are essentially approximate, and they often add serious inaccuracies to the construction.

2.2. Improvements to the traditional method

The search for more sophisticated ways to automatically obtain loci has led different DG systems to consider different approaches. We summarize here the most relevant.

2.2.1. Locus recognition by minimizing distance to algebraic curves

The first DG system to include a command to provide algebraic information for a locus was Cabri. Since its release in 2003, *Cabri Geometry* II *plus*, the current version of Cabri, incorporates a tool for computing *approximate* algebraic equations for loci.

Although proper documentation of this feature is not provided by Cabrilog, the company behind Cabri, a schematic description of the algorithm used in the back-end can be found in [9]. It is based on the random selection of one hundred locus points and the computation of the best approaching polynomial curve (up to degree six) to this collection of points. Let us point out that the limiting factors of this approach come from sampling and fitting points to sufficiently high accuracy. Moreover, the number of monomials whose coefficients must be found grows as the square of the degree.

This numerical procedure does not result, in our opinion, in a satisfactory solution. In fact, simple locus constructions can easily give rise to algebraic curves of degree higher than 6 (see, for instance, [10]), that would go undetected for Cabri. Moreover, no comment is attached to the locus output concerning the (in)exactness of the algebraic information provided, hence inducing a non expert user to take it as an accurate one (cf. [11], where Cabri is shown to return a cubic as equation for the curve of Watt).

Likewise, in [12,13], the authors consider also the rendering of some (many) sample points of a locus set constructed by ruler and compass as the initial data of an algorithm to determine the degree and parameters of an algebraic curve 'resembling' the locus. In a second step, a collection of such curves, obtained varying the position of basic construction points, is analyzed in order to get more general knowledge about the involved locus.

Although impressively precise in certain situations, the algorithm is prone to inaccuracies for curves of high degrees [12, p. 63]. Besides these problems, the authors report other drawbacks in the method, that make it unsuited for efficient implementation. In summary, we consider this a promising, but still an open approach to automated locus determination.

2.2.2. Randomized theorem proving techniques in cinderella

In [14,15], the authors (and developers of Cinderella [16]) review how their software uses automatic theorem proving to add

[1] See comment by the creator of The Geometer's Sketchpad about the construction of tangents to a locus set as the limit of secants in http://mathforum.org/kb/message.jspa?messageID=1095049.

extra information to certain elements in a geometric construction. In particular, Cinderella uses automated deduction techniques based on randomized methods to improve the knowledge about some loci.

Roughly speaking, randomized theorem proving consists on checking a property for a sufficient number of examples. Moreover, randomized theorem proving could provide a valid certificate answer if tested on a sufficient number of examples related to the degrees of the involved polynomials. For instance, given a construction including three points A, B and C, the system will take as a fact that the points are aligned if the line AB contains the point C for a large set of instances obtained by randomly modifying the position of points A and B. From then on, the system will take the elements $line(A, B)$ and $line(B, C)$ to be identical.

In particular, for any locus in a diagram (i.e. a finite set of sample locus points), the line defined by the first two sample points, is constructed. The system then checks whether the rest of the sample points belong to this line, not only for that particular instance of the diagram, but also for any instance in a large set of random modifications of the diagram. In that case, the locus element is replaced by that line (together with its equation). If the locus is not identified as a line, a similar process is followed using the circle defined by the first three locus points. If not identified as circle either, the conic defined by the first five interpolation points is taken as candidate.

This replacement, when successful, not only facilitates the rendering process of the locus, but also allows the system to use it for further constructions, such as intersections with other objects.

Although approximate in nature, the method provides an effective way to improve locus generation for many constructions. However, the current Cinderella implementation can deal only with lines and conics, since there are no other locus objects defined by equations in this system. This makes this approach a limited answer to the general question of locus implementation in DG.

Related to randomized theorem proving, but more sophisticated, is numerical algebraic geometry, which also exploits the idea of drawing conclusions about algebraic sets by numerically testing whether sample points satisfy algebraic conditions. Moreover, it uses sampling over the complex numbers, not just real numbers, to strengthen its performance (see [17]). Although no DGS has yet incorporated numerical algebraic geometry, it should provide a strong numerical alternative for locus computation to the approaches mentioned in this note.

2.2.3. *Locus discovery with algebraic elimination techniques*

In many instances, a dynamic geometry construction concerning a locus computation can be viewed as a set of polynomial equations, corresponding to the analytic expression of the geometric objects involved in the description of the mover and tracer points. Then, roughly speaking, computing a locus can be understood as obtaining an equivalent set of polynomials, but only in the variables corresponding to the tracer, i.e. as eliminating the remaining variables.

For this task, constructive elimination tools, such as Gröbner bases [18,19] and Wu's method [20,21], are crucial. Although some authors have used Wu's method for algebraic loci computation (e.g. [22–24], albeit with no GUI, and [25]) the use of Wu's method for a true automatic generation of loci within a DG system remains unexplored. On the other hand, Gröbner bases have been widely used for automatic theorem proving [26–28]. In particular, in [29], a method based on Gröbner bases for automatic discovery is described. Moreover, linking Cabri, the most popular DG system at the time, and the Gröbner basis method for automatic discovery, in an intelligent program for learning Euclidean geometry, is explicitly proposed. Specializing this approach, an algorithm for automatic discovery of loci based on Gröbner bases was introduced in [30].

The elimination (using Gröbner bases) of some variables in the polynomial ideal obtained as translation of the construction, leaves us with a set of polynomials in the tracer-point coordinates only. The zero set of these polynomials is, in general, a superset of the sought locus set.

A problem with this algebraic approach is that the obtained algebraic set may contain extra components, sometimes due to the fact that the method returns only Zariski closed sets,[2] some other times due to degenerate instances of the construction.

For instance, let us consider the limaçon of Pascal, a conchoid that can be constructed as a DG locus as follows. Let O be a fixed point on a circle c, let l be a line passing through O and P (a general point on c). Let Q be a point on l such that $distance(P, Q) = k$, where k is a constant. The limaçon of Pascal is the locus set traced by Q as P moves along c, as shown on Fig. 1 (left).

We make the following assignment of coordinates: $P(x_1, x_2)$, $Q(x, y)$. For example, if we consider that O is the point $(0, 2)$, $k = 1$ and $(0, 0)$ the center of c we get the ideal $I = (x_1^2 + x_2^2 - 4, (x_1 - x)^2 + (x_2 - y)^2 - 1, x(x_2 - 2) - x_1(y - 2))$ whose polynomials correspond, respectively, to the following geometric constraints: P is in the circle of center $(0, 0)$ and radius 2, $distance(P, Q) = 1$ and $Q \in Line(P, O)$. Eliminating variables x_1 and x_2, we obtain the following product of two polynomials $(x^4 + 2x^2y^2 + y^4 - 9x^2 - 9y^2 + 4y + 12)(x^2 + y^2 - 4y + 3)$. While the first factor provides the implicit equation for the actual limaçon, the second factor corresponds to a spurious circle associated to the degenerate case for which $P = O$, when the line l ceases to exist (see Fig. 1, right).

Example 1 in Section 5 provides an example of locus for which this procedure would return an algebraic set with extra points, due to the Zariski closedness of the result.

Despite its limitations, this algebraic approach was a significant improvement, not only over the traditional method, but also over all other approaches mentioned above. The provided analytical knowledge about general algebraic loci, albeit sometimes incorrect, is a prerequisite for integrating loci as standard objects in DG environments. Thus, the approach attracted the attention of developers, being this approach behind the LocusEquation command in the current version of GeoGebra.[3] Furthermore, it has also been implemented by the DG system JSXGraph using remote computations on a server [31], an idea previously developed in [32].

3. A taxonomy of loci as projections

As described in Section 2, many dynamic geometry constructions in the plane can be viewed as polynomial systems on the variables corresponding to the symbolic coordinates of the objects in the construction. While in standard dynamic geometry loci always involve a mover point, our approach subsumes these loci into a more general setting. In this way, simple loci as the circle defined through a point and a radius, or loci where there is not a mover bound to a linear object (see Section 5.4), can be efficiently found.

We start by distinguishing the variables corresponding to the coordinates of the tracer $T(x, y)$ from the rest of variables, say $x_1, \ldots, x_n$, corresponding to the remaining points and objects in the construction; in particular the coordinates of the mover if they are explicitly specified. Note that the consideration of a mover point comes from the constructive strategy followed in most DG environments when considering loci. However, in a constraint-based geometric system, no mover point is involved when searching for a locus, since more than one point can be generally used

[2] That is, the complete solution set of a system of polynomial equations. No missing points are allowed (see Section 5.1).

[3] See http://wiki.geogebra.org/en/LocusEquation_Command.

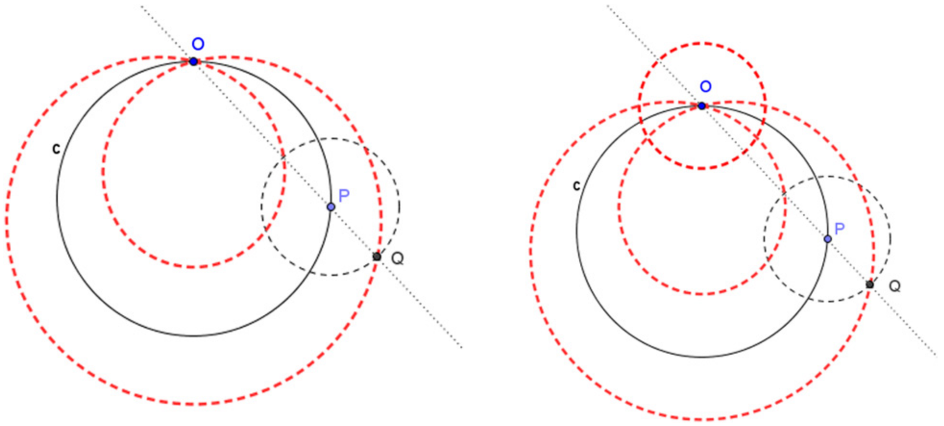


Fig. 1. Limaçon of Pascal as the locus set traced by Q as P runs along the circle c (left) and Limaçon with extra circle (right).

to *drag* the construction. Thus, although for the sake of clarity we talk about mover points when describing the examples, the reader should be aware that no algebraic preeminence is given to any point other than the locus point.

The translation of the geometrical constraints defining the construction results in a system F of polynomial equations in the *variables* $x_1, \ldots, x_n$ with coefficients given by polynomials in the *parameters* x, y.

Our approach consist of detecting for which values of the parameters (x, y) (tracer) there exist solutions of the system F. The set of equations is defined over a computable field K, that we always take to be $\mathbb{Q}$, whereas the values of the variables (and parameters) must be considered over an algebraically closed extension $\overline{K}$ of K, that we take to be $\mathbb{C}$. Our approach involves the comparison of dimensions of different algebraic varieties. Since the dimensions of complex and real varieties are in general different, we have opted to work in the complex framework, as customary in the field (see, for instance, [15,21]).

Let us point out that the algebraic study of loci that we are developing in dimension 2 for classical reasons, can be generalized in theory to higher dimensions. In fact Gröbner covers are not constrained to work only in the 2D case. Here, we focus on 2-dimensional loci since 3D DG environments are not yet quite developed. Furthermore, there are specific issues concerning the application of the theory of parametric polynomial systems to 3D DG. Thus, we delay such a study to a future communication.

Before giving our definition of locus, let us state some basic concepts about locally closed sets and constructible sets.

A locally closed set L is a difference of algebraic varieties $L = \mathbf{V}(E) \setminus \mathbf{V}(N)$. As explained in [33], for a locally closed set, a canonical P-representation expressed in terms of prime ideals can be obtained:

$$\textsc{Prep}(L) = \{\{\mathfrak{p}_i, \mathfrak{p}_{ij} : 1 \le j \le r_i, r\} : 1 \le i \le r\}$$

so that

$$L = \bigcup_{i=1}^{r} \left(\mathbf{V}(\mathfrak{p}_i) \setminus \left(\bigcup_{j=1}^{r_i} \mathbf{V}(\mathfrak{p}_{ij}) \right) \right).$$

As illustrative examples one can consider the following simple locally closed sets:

$S_1 = \mathbf{V}(x) \setminus \mathbf{V}(y(y-1))$

$\textsc{Prep}(S_1) = \mathbf{V}(x) \setminus (\mathbf{V}(x, y) \cup \mathbf{V}(x, y-1))$

$p_1 = \langle x \rangle, \qquad p_{11} = \langle x, y \rangle, \qquad p_{12} = \langle x, y-1 \rangle$

$S_2 = \mathbf{V}(xy)\, \mathbf{V}(x+y-1)$

$\textsc{Prep}(S_2) = (\mathbf{V}(x)\, \mathbf{V}(x, y-1)) \cup (\mathbf{V}(y) \setminus \mathbf{V}(x-1, y))$

$p_1 = \langle x \rangle, \qquad p_{11} = \langle x, y-1 \rangle, \qquad p_2 = \langle y \rangle,$

$p_{21} = \langle x-1, y \rangle.$

Each element $\mathbf{V}(\mathfrak{p}_i) \setminus \left(\bigcup_{j=1}^{r_i} \mathbf{V}(\mathfrak{p}_{ij}) \right)$ is called a *component* of L and is represented by $\{\mathfrak{p}_i, \{\mathfrak{p}_{ij} : 1 \le j \le r_i\}\}$, that, by abuse of terminology, is also denoted *component* whenever there is no ambiguity. In the canonical representation, the irreducible varieties are expressed in terms of prime ideals on account of the well known one-to-one correspondence between irreducible varieties and prime ideals. Given a component as above, the variety $\mathbf{V}(\mathfrak{p}_i)$ (or its representative $\mathfrak{p}_i$) is called the *top* of the component. Similarly, the varieties $\mathbf{V}(\mathfrak{p}_{ij})$ (or their representatives $\mathfrak{p}_{ij}$) are called the *holes*. In particular, the dimension of each hole variety is smaller than the dimension of its corresponding top variety.

A constructible set is a union of locally closed sets. In general, a union of locally closed sets is not locally closed, but we can also give a canonical description in terms of disjoint embedded locally closed subsets and represent them canonically in P-representations. Furthermore, we consider another representation for constructible sets, the C-representation, defined as follows.

Proposition 3.1 (C*-Representation of Constructible Sets*)**.** *Let $S \subset \overline{K}^m$ be a constructible set. There exist uniquely determined radical ideals $((\mathfrak{a}^{(\ell)}, \mathfrak{b}^{(\ell)}) : 1 \le \ell \le s)$, such that*

- $\mathfrak{a}^{(1)} \subset \mathfrak{b}^{(1)} \subset \mathfrak{a}^{(2)} \subset \mathfrak{b}^{(2)} \subset \cdots \subset \mathfrak{a}^{(s)} \subset \mathfrak{b}^{(s)}$
- $S^{(\ell)} = \mathbf{V}(\mathfrak{a}^{(\ell)}) \setminus \mathbf{V}(\mathfrak{b}^{(\ell)})$,
- $\overline{S^{(\ell)}} = \mathbf{V}(\mathfrak{a}^{(\ell)})$ *where $\overline{S^{(\ell)}}$ is the Zariski closure of $S^{(\ell)}$*
- $\overline{S^{(\ell)}} \setminus S^{(\ell)} = \mathbf{V}(\mathfrak{b}^{(\ell)})$
- $S = \bigcup_\ell S^{(\ell)}$ *is a disjoint union of embedded locally closed sets,*
- $\dim(\mathfrak{a}^{(1)}) > \dim(\mathfrak{b}^{(1)}) > \cdots > \dim(\mathfrak{a}^{(s)}) > \dim(\mathfrak{b}^{(s)})$.

$S^{(\ell)}$ is called the ℓth level *of the constructible set S, and the set of pairs $((\mathfrak{a}^{(\ell)}, \mathfrak{b}^{(\ell)}) : 1 \le \ell \le s)$ is called the C*-representation *of S.*

The canonical C-representation of a constructible set expresses the set as a hierarchical and disjoint union of locally closed subsets given in C-representation.

Proof. Let $\overline{S}$ be the closure of S. $\mathfrak{a}^{(1)}$ can be described canonically by $\overline{S} = \mathbf{V}(\mathfrak{a}^{(1)})$. If S is locally closed, then the complement of S w.r.t. $\overline{S}$ will be closed, and in that case $\mathfrak{b}$ can be canonically defined by $\mathbf{V}(\mathfrak{b}) = \overline{S} \setminus S \subset \overline{S}$ and so $S = \mathbf{V}(a^{(1)}) \setminus \mathbf{V}(\mathfrak{b})$. If $\overline{S} \setminus S$ is not closed, then $\mathfrak{b}^{(1)}$ can be defined by the closure $\mathbf{V}(\mathfrak{b}^1) = \overline{\overline{S} \setminus S}$ so that $\mathbf{V}(\mathfrak{b}^{(1)}) \subset \overline{S} = \mathbf{V}(\mathfrak{a}^{(1)})$, and denote $S^{(1)} = \mathbf{V}(\mathfrak{a}^{(1)}) \setminus \mathbf{V}(\mathfrak{b}^{(1)})$. We have

$$S^{(1)} = \mathbf{V}(\mathfrak{a}^{(1)}) \setminus \mathbf{V}(\mathfrak{b}^{(1)}) = \overline{S} \setminus \overline{\left(\overline{S} \setminus S\right)} \subseteq \overline{S} \setminus \left(\overline{S} \setminus S\right) = S \quad \text{and}$$

$$S \setminus S^{(1)} = S \setminus \left(\overline{S} \setminus \mathbf{V}(\mathfrak{b}^{(1)})\right) \subseteq S \setminus \left(S \setminus \mathbf{V}(\mathfrak{b}^{(1)})\right) = \mathbf{V}(\mathfrak{b}^{(1)}).$$

Thus $S^{(1)} \subseteq S$ and its complement $S \setminus S_1$ w.r.t. S is again constructible and included in $\mathbf{V}(\mathfrak{b}^{(1)})$. Particularly $\mathbf{V}(\mathfrak{a}^{(2)}) = \overline{S \setminus S_1}$ and $\mathbf{V}(\mathfrak{a}^{(2)}) \subseteq \mathbf{V}(\mathfrak{b}^{(1)})$. The process can be continued until $S^{(s+1)}$ becomes empty.

To prove the strict inclusions

$$\mathfrak{a}^{(1)} \subset \mathfrak{b}^{(1)} \subset \mathfrak{a}^{(2)} \subset \mathfrak{b}^{(2)} \subset \cdots \subset \mathfrak{a}^{(s)} \subset \mathfrak{b}^{(s)},$$

we have to consider the prime decomposition of the radical ideals

$$\mathfrak{a}^{(1)}, \mathfrak{b}^{(1)}, \mathfrak{a}^{(2)}, \mathfrak{b}^{(2)}, \ldots, \mathfrak{a}^{(s)}, \mathfrak{b}^{(s)},$$

and observe that, by construction, no prime ideal in the decomposition of one of those ideals can be equal to a prime ideal in the decomposition of the next radical ideal in the chain, as we have always consider closures and complements. From this result, as the dimension of an irreducible variety containing another irreducible variety is strictly higher than the latter, the result of the descending dimensions of the chain follows. □

Note 3.2. Because of the strict decreasing dimension of the hierarchical description, a constructible set of dimension 1 is not only constructible, but is also locally closed.

We can proceed now with the definition of a locus. As stated above, a locus in DG is translated into a set of parametric polynomial equations $F \subseteq \mathbb{Q}[\mathbf{u}, \mathbf{x}]$ where $\mathbf{u} = (x, y)$ are the parameters (representing the tracer) and $\mathbf{x} = (x_1, \ldots, x_n)$ the variables. Consider its solutions:

$$\mathbf{V}(F) = \{(\mathbf{u}, \mathbf{x}) \in \mathbb{C}^{2+n} : \forall f \in F, f(\mathbf{u}, \mathbf{x}) = 0\}.$$

Denote by π_1 and π_2 the projections onto the parameter and variable space, respectively:

$$\begin{array}{llll} \pi_1 : & \mathbb{C}^{2+n} & \longrightarrow & \mathbb{C}^2 \\ & (\mathbf{u}, \mathbf{x}) & \mapsto & \mathbf{u} \end{array} \qquad \begin{array}{llll} \pi_2 : & \mathbb{C}^{2+n} & \longrightarrow & \mathbb{C}^n \\ & (\mathbf{u}, \mathbf{x}) & \mapsto & \mathbf{x}. \end{array}$$

We can now introduce a generic formal definition of a locus in algebraic terms.

Definition 3.3. The generic locus L associated to the parametric polynomial system $F(\mathbf{u}, \mathbf{x})$, is the set $L = \pi_1(\mathbf{V}(F)) \subset \mathbb{C}^2$.

Roughly speaking, the locus is the set of points (x, y) satisfying the polynomials in F. Looking at F as a parametric polynomial system, we will discuss this system attending to the number, finite or infinite, of solutions of $x_1, \ldots, x_n$ in terms of parameters x, y. As a first step in the classification process at the base of our taxonomy, we split the complex locus $L = \pi_1(\mathbf{V}(F))$ into two disjoint subsets, regarding the dimension of the solution set for the variables corresponding to a specific value of the parameters: the *normal locus* and the *non-normal locus*. This distinction comes from the fact that a point in a DG locus is usually produced by a finite set of values of the variables.

Definition 3.4 (*Normal and Non-Normal Locus*). *Normal points* are those points $\mathbf{u} \in \mathbb{C}^2$ of the locus for which $\dim(\pi_2(\mathbf{V}(F) \cap \pi_1^{-1}(\mathbf{u}))) = 0$. The points $\mathbf{u}$ of the locus for which $\dim(\pi_2(\mathbf{V}(F) \cap \pi_1^{-1}(\mathbf{u}))) > 0$ are called *non-normal*. The set of all normal points is called the *normal locus* and the set of all non-normal points is called the *non-normal locus.*

Proposition 3.5. *The normal and non-normal loci are constructible sets.*

Proof. By Chevalley's theorem [34, IV.13.1.3 and IV.13.1.5], we know that the set $\{\mathbf{u} \in \mathbb{C}^2 : \dim(\mathbf{V}(F) \cap \pi_1^{-1}(\mathbf{u})) < d\}$ is open (in the Zariski topology) for any $d \in \mathbb{N}$. In particular, for $d = 1$, we obtain that the normal locus is constructible.

For an arbitrary dimension d we have that

$$\{\mathbf{u} \in \mathbb{C}^2 : \dim(\mathbf{V}(F) \cap \pi_1^{-1}(\mathbf{u})) = d\}$$

is equal to

$$\{\mathbf{u} \in \mathbb{C}^2 : \dim(\mathbf{V}(F) \cap \pi_1^{-1}(\mathbf{u})) < d + 1\}$$

minus

$$\{\mathbf{u} \in \mathbb{C}^2 : \dim(\mathbf{V}(F) \cap \pi_1^{-1}(\mathbf{u})) < d\},$$

which is the difference of two open sets, and hence constructible. This implies that the non-normal locus is a union of constructible sets and hence constructible. □

Proposition 3.5 allows us to further subdivide the locus set by considering the components associated to the canonical representations of the normal and non-normal locus as constructible sets. Informally speaking, a part of the locus is distinguished if it violates the one-to-one correspondence between the locus points and the corresponding set of variable values.

Definition 3.6 (*Normal and Special Components*). A component C_s of the normal locus is *special* if $\dim(C_s) > 0$ and $\dim(\pi_2(\mathbf{V}(F) \cap \pi_1^{-1}(C_s))) = 0$. The remaining components of the normal locus are *normal.*

Definition 3.7 (*Degenerate and Accumulation Components*). The components C_d of the non-normal locus of dimension greater than 0 are considered *degenerate components*, whereas the zero-dimensional components are *accumulation points* of the locus.

The geometric relevance of this algebraic classification of the different parts of a locus is open to interpretation by the user. Dynamic Geometry systems could present the collection of different parts (with the corresponding typology) of the computed locus, so the user would decide which pieces to discard or to keep as pertinent in a particular context.

Based on our experience (see Section 5), we will discard the degenerate components as geometrically irrelevant, as they usually correspond to degenerate instances of a construction, such as two coincident vertices in a triangle. However, we consider the accumulation points as forming part of the (geometric) locus, since they represent special points that are determined by infinitely many values of the variables. Examples of both phenomena can be found in the battery of examples included in the web prototype described in Section 4.2 ([35], Locus 7 and Locus 12 respectively).

Finally, let us point out that the relevance of our proposal for a taxonomy is that, in the many instances we have worked with so far, we have never had to split one component (in the sense of Definitions 3.6 or 3.7) in order to keep a part of that component as relevant and to throw away the other part as inadequate for the locus computation.

4. Algorithm and web implementation

In this section we address the following problem: how to effectively and efficiently compute the different components of a locus, according to Definitions 3.6 and 3.7 above. Here we propose the use of the recently developed GröbnerCover algorithm to automatically detect the different components of a locus in a DG system. This algorithm, inscribed in the theory of parametric polynomial systems solving, has as input a finite set of parametric polynomials, and outputs a finite partition of the parameter space into locally closed subsets together with polynomial data, from which the reduced Gröbner basis for a given parameter point can be directly determined.

What follows is a summary of the main properties of this algorithm, whose details can be found in [33].

Let $\mathcal{I} \subset \mathbb{Q}[\mathbf{u}][\mathbf{x}]$ be a polynomial ideal for the parameters $\mathbf{u} = u_1, \ldots, u_m$ and the variables $\mathbf{x} = x_1, \ldots, x_n$ and consider $\mathbf{V}(\mathcal{I})$, the *solution set* of the system given by $\mathcal{I}$:

$$\mathbf{V}(\mathcal{I}) = \{(\mathbf{u}, \mathbf{x}) \in \mathbb{C}^{m+n} : \forall f \in \mathcal{I}, f(\mathbf{u}, \mathbf{x}) = 0\}.$$

Given the ideal $\mathcal{I} \subset \mathbb{Q}[\mathbf{u}][\mathbf{x}]$ (and a monomial order in the variables), its Gröbner cover (GC) is a set of pairs $\{(S_i, B_i) : 1 \leq i \leq s\}$ of (*segment, basis*), that classifies the parameter space $\mathbb{C}^m$ by the kind of solutions in the variables:

Table 1
Locus algorithm.

```
Input: $G = \{(S_i, B_i, \mathrm{lpp}_i) : i \le i \le s\}$ the Gröbner cover of an ideal
    where $S_i = \cup_j C_{ij}$ and $C_{ij} = \{(\mathfrak{p}_{ij}, \{\mathfrak{p}_{ijk} : 1 \le k \le r_{ij}\}) : 1 \le j \le r_i\}$.
Output: $L = \mathbf{Locus}(G)$, the components of the $P$-representation of the locus
    $L = \{\{\mathfrak{q}_i, \{\mathfrak{q}_{ij} : 1 \le j \le s_i\}, \mathrm{type}_i\} : 1 \le i \le s\}$
begin
  $C_1$ = Select the segments of $G$ with $\dim(\mathrm{lpp}_i) = 0$ # normal-segments
  $C_1$ = Specialize the basis on every component of $C_1$ and mark the
    component Normal if the basis continues to depend on the $\mathbf{u}$'s and
    Special if not
  $C_2$ = Select all the components of the segments of $G$ with $\dim(\mathrm{lpp}_i) > 0$
    # non-normal segments
  $L_1 = \mathbf{LCUnion}(C_1)$;
    marking the components of $L_1$ as Normal or Special inheriting
    the character of the full
    $L_2 = \mathbf{LCUnion}(C_2)$;
  Mark the components of $L_2$ of $\dim(C) = 0$ and $\dim(C) > 0$
    # respectively as Accumulation and Degenerate components
  $L = L_1 \cup L_2$
end
```

1. The segments $S_i \subset \mathbb{C}^m$ are disjoint.
2. The segments S_i are locally closed subsets of the parameter space $\mathbb{C}^m$, expressed in the canonical P-representation, namely

$$S_i = \bigcup_j \left(\mathbf{V}(\mathfrak{p}_{ij}) \setminus \left(\bigcup_k \mathbf{V}(\mathfrak{p}_{ijk}) \right) \right),$$

   and $(\mathfrak{p}_{ij}, (\mathfrak{p}_{ijk} : 1 \le k \le s_{ij}))$ is called the jth component of the ith GC-segment.
3. Associated to each segment S_i there is a basis $B_i \subset \mathbb{Q}[\mathbf{u}][\mathbf{x}]$ that specializes to the reduced Gröbner basis of $\mathcal{I}$ for every point $\mathbf{u} \in S_i$ of the segment.
4. The kind of solution in the variables is given by the set of *leading power products* (lpp's) of the bases B_i, that are fixed for each GC segment S_i (and is also explicitly given by the algorithm). Thus, for all points in the segment, the ideal $\mathcal{I}$ has the same number of solutions.
5. Moreover, if the ideal $\mathcal{I}$ is homogeneous, then the lpp's sets are different on each segment. (The lpp's of the homogenized ideal are also explicitly given by the algorithm for each segment S_i as they characterize the segments.)

4.1. *The* Locus *algorithm*

Based on the output of the GröbnerCover algorithm applied to the system F associated to a DG locus, the Locus algorithm in Table 1 computes and classifies the locus components.

Definitions 3.4, 3.6 and 3.7 allow us to assign to each segment of the Gröbner cover a first locus taxonomy, regarding simply the set of leading power products of the bases (lpp). We obtain segments of three types:

Type 1 Segments with basis $\{1\}$ do not belong to the locus. In particular, the generic segment, which is the unique open segment in $\mathbb{C}^2$ (having thus dimension 2) is expected to have basis $\{1\}$, so the locus components are expected to have dimension less or equal to 1.
Type 2 Segments with a finite number of solutions correspond to the normal locus.
Type 3 Segments with an infinite number of solutions correspond to the non-normal locus.

Inside the normal locus segments of type 2, specializing the basis over each component allows us to refine the locus taxonomy. If the specialized basis does not depend on the parameters $\mathbf{u}$, then the component is labeled 'Special'. Otherwise it is labeled 'Normal'.

The non-normal locus segments of type 3 need not be previously classified.

To obtain the components of the constructible locus sets, the Locus algorithm has to collect now separately the components of both kinds of locus: the components of the normal segments of type 2 and of the non-normal segments of type 3. For this purpose, it uses the LCUnion algorithm (see [33]) which is designed to compute the canonical P-representation of the addition of locally closed components given in P-representation. LCUnion takes the components to be added and outputs the canonical P-representation of the first level of the resulting constructible set, and it also returns the components that have not been used because they belong to higher levels of the constructible set. To build the whole constructible set one has to iterate LCUnion with the remaining components. In fact, by Note 3.2, the additions to be done are locally closed, and so it suffices to use LCUnion only once.

For the normal locus, since the top varieties of the union are also tops of some component of the components of type 2 that are added, the label 'Normal' or 'Special' is inherited from the tops in LCUnion.

For the non-normal locus, it suffices to add the components of type 3 using LCUnion, and then label the resulting components as 'Accumulation' if the resulting component of the constructible set has a finite number of points, and 'Degenerate' if it contains infinitely many points.

The normal and the non-normal loci are disjoint, since a point in the parameter space cannot be normal and non-normal, and the GröbnerCover algorithm forms these subsets by adding segments that are disjoint. But the components inside the normal locus can have non-empty intersection and in that case the intersection points will belong to both components.

However, 'Accumulation' and 'Degenerate' components of the non-normal locus are disjoint, since 'Accumulation' points must be isolated points not adherent to any higher dimensional component, for in that case it would be incorporated to the higher dimensional component when considering the union.

4.2. *Web implementation*

The Locus algorithm detailed in the previous section provides four different kinds of components for a locus set, namely, normal, special, degenerate and accumulation components. Although all of them are algebraically meaningful, only the normal and accumulation components of a locus have been considered *true geometric parts* of a dynamic geometry locus.

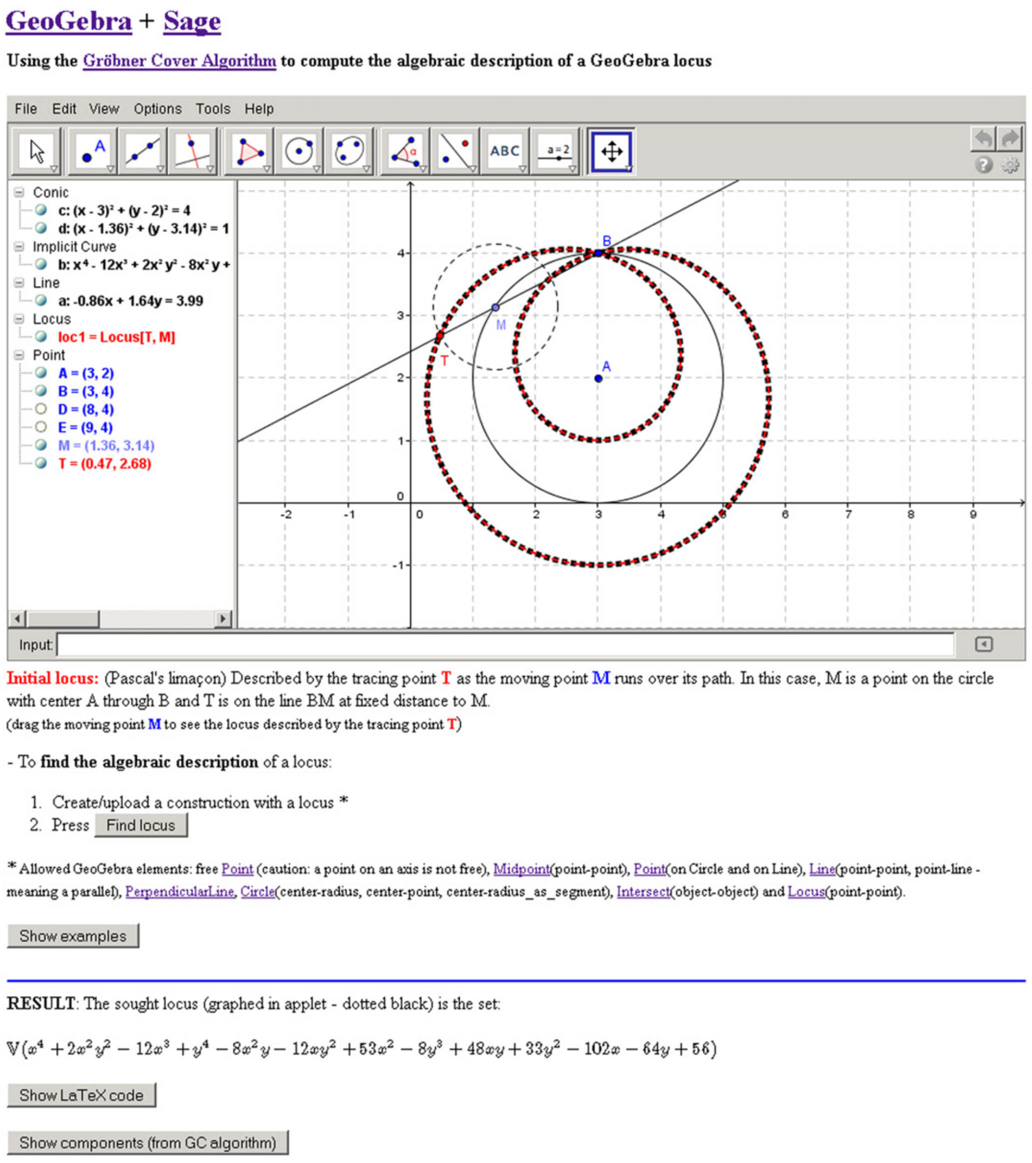
Following this criterion, a prototype web application that provides the accurate algebraic and graphic description of a geometric locus in a DG system has been developed. This prototype, freely accessible in [35] (where the code is moreover available), includes a battery of 12 representative examples.

The system consists of a drawing canvas, where the computed locus is displayed together with the initial elements. It is based on the free DG system GeoGebra[4] and the open source CAS Sage.[5]

More concretely, to obtain the algebraic description of a given locus, the algebraic knowledge obtained from a construction introduced through a GeoGebra applet is automatically encoded and sent to a Sage server, where it is remotely processed by Singular [36], a system bundled inside the Sage distribution.

Despite the technicalities of the remote interconnection of GeoGebra and Sage, the web application is presented as a simple web page with a GeoGebra applet, where to construct/upload a locus. Given a locus construction (specified using a predetermined set of GeoGebra commands), the prototype provides the algebraic description of the locus set by just pressing one button. The process goes roughly as follows.

[4] http://www.geogebra.org.
[5] http://www.sagemath.org.



GeoGebra + Sage

Using the Gröbner Cover Algorithm to compute the algebraic description of a GeoGebra locus

Conic
c: (x - 3)² + (y - 2)² = 4
d: (x - 1.36)² + (y - 3.14)² = 1
Implicit Curve
b: x⁴ - 12x³ + 2x² y² - 8x² y +
Line
a: -0.86x + 1.64y = 3.99
Locus
loc1 = Locus[T, M]
Point
A = (3, 2)
B = (3, 4)
D = (8, 4)
E = (9, 4)
M = (1.36, 3.14)
T = (0.47, 2.68)

Initial locus: (Pascal's limaçon) Described by the tracing point **T** as the moving point **M** runs over its path. In this case, M is a point on the circle with center A through B and T is on the line BM at fixed distance to M.
(drag the moving point **M** to see the locus described by the tracing point **T**)

- To **find the algebraic description** of a locus:

1. Create/upload a construction with a locus *
2. Press Find locus

* Allowed GeoGebra elements: free Point (caution: a point on an axis is not free), Midpoint(point-point), Point(on Circle and on Line), Line(point-point, point-line - meaning a parallel), PerpendicularLine, Circle(center-radius, center-point, center-radius_as_segment), Intersect(object-object) and Locus(point-point).

Show examples

RESULT: The sought locus (graphed in applet - dotted black) is the set:

$\mathbb{V}(x^4+2x^2y^2-12x^3+y^4-8x^2y-12xy^2+53x^2-8y^3+48xy+33y^2-102x-64y+56)$

Show LaTeX code

Show components (from GC algorithm)

Fig. 2. A screen capture of the prototype web page.

Despite the technicalities of the remote interconnection of GeoGebra and Sage, the web application is presented as a simple web page with a GeoGebra applet, where to construct/upload a locus. Currently, there is a reduced list of admissible GeoGebra commands involving points (Point, Midpoint), lines (Line, PerpendicularLine) and circles (Circle), together with intersecting objects (Intersect) and the standard Locus command (see the prototype web page, where links to the exact meaning of used commands are given). For a locus construction, the prototype provides the algebraic description of the locus set by just pressing one button. The process goes roughly as follows.

First, the XML description of the GeoGebra construction is sent to a Sage cell server [37]. On the server, the construction follows an algebraization process as specified by a special library [38]. The obtained parametric polynomial system is then fed into an implementation in Singular of the GröbnerCover algorithm. The results are finally returned to the user in text form as well as graphically in the applet.

A screen capture of the web page with an accurate description of the limaçon of Pascal discussed in Section 2.2 is shown in Fig. 2. It provides its graph (thick dotted) together with its description as an algebraic set. Note that no extra special component is included in the description, unlike the description provided by standard algebraic methods, as discussed in Section 2.2.

Although only the normal and accumulation components of a locus, as provided by the algorithm, are used by the system when describing the locus, all four sets of components are provided when pressing the *Show components (from GC algorithm)* button. The following is the textual information provided by the prototype, showing the different components for the limaçon of Pascal, where the extra circle mentioned in Section 2.2 is identified as special.

```
Components from GC algorithm:
 Normal components: [[[x^4+2*x^2*y^2+y^4-12*x^3
       -8*x^2*y-12*x*y^2
       -8*y^3+53*x^2+48*x*y
       +33*y^2-102*x-64*y+ 56]]]
 Accumulation components: []
 Special components: [[[x^2 + y^2 - 6*x - 8*y + 24]]]
 Degenerate components: []
```

Note that the goal is not to provide a system for a complete general use, but to show a proof of concept of the feasibility of using sophisticated algorithms like the GröbnerCover to supplement the symbolic capabilities of existing dynamic geometry systems, as well as to show the advantage of connecting different systems by using web services.

5. Examples

5.1. Example 1: Sketchpad classic construction

We consider the original locus example by Sutherland in [7, p. 102][6] that can be described as follows:

Let $A(x_a, y_a)$, $B(x_b, y_b)$ and $C(x_c, y_c)$ be three fixed points, and a and b two lines passing respectively through A and B. Let c be the circle with center C and radius r. Consider a point G on the circle c as the mover point. The line CG intersects a in a point I and b in a point H. We take as tracer the intersection point J of lines AH and BI (see Fig. 3).

We revisit this example through a simple Gröbner based elimination approach, as well as through our proposed method.

Assigning symbolic coordinates to $H(x_1, y_1)$, $I(x_2, y_2)$, $G(x_3, y_3)$, $J(x, y)$, and considering that the equations of lines a and b can be written as $m(X - x_a) - (Y - y_a) = 0$ and $n(X - x_b) - (Y - y_b) = 0$ respectively, it is easy to establish the polynomials for the construction:

$$\begin{aligned} F = {} & (x_3 - x_c)^2 + (y_3 - y_c)^2 - r^2, \\ & m(x_2 - x_a) - (y_2 - y_a), n(x_1 - x_b) - (y_1 - y_b), \\ & \begin{vmatrix} x_1 & y_1 & 1 \\ x_3 & y_3 & 1 \\ x_c & y_c & 1 \end{vmatrix}, \begin{vmatrix} x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_c & y_c & 1 \end{vmatrix}, \begin{vmatrix} x & y & 1 \\ x_a & y_a & 1 \\ x_1 & y_1 & 1 \end{vmatrix}, \begin{vmatrix} x & y & 1 \\ x_b & y_b & 1 \\ x_2 & y_2 & 1 \end{vmatrix}. \end{aligned} \tag{1}$$

In order to minimize the number of parameters, we fix points $A(0, 0)$, $B(3, 0)$, and the radius $r = 5$.

To determine the locus, we use basic elimination first. Computing the Gröbner basis of the ideal $F(x_a = 0, y_a = 0, x_b = 3, y_b = 0, r = 5)$ of formula (1) to eliminate the variables $x_1, y_1, x_2, y_2, x_3, y_3$ (with the graded reverse lexicographical order grevlex$(x_1, y_1, x_2, y_2, x_3, y_3)$, grevlex$(x_c, y_c, n, m, x, y)$) we obtain

$$\begin{aligned} & mny_cx^2 + ((m + n)xc - yc - 3n)y^2 + (mn(3 - 2x_c))xy \\ & - 3mny_cx + 3mnx_cy \end{aligned} \tag{2}$$

that gives a parametric locus depending on the parameters (x_c, y_c, m, n).

Let us now compute the locus using our algorithm. We must manually fix point C and the parameters m, n to have a concrete locus problem, as the algorithm does not efficiently deals with free parameters in its current version. We choose $C(1, 3)$, $m = 1$ and $n = -1/2$.

The specialized system is now:

$$\begin{aligned} F_0 = {} & (x_3 - 1)^2 + (y_3 - 3)^2 - 25, \\ & x_2 - y_2, x_1 - 3 + 2y_1, \\ & x_1y_3 - 3x_1 + 3x_3 - x_3y_1 + y_1 - y_3, \\ & x_2y_3 - 3x_2 + 3x_3 - x_3y_2 + y_2 - y_3, \\ & -xy_1 + x_1y, \\ & -xy_2 + 3y_2 - 3y + x_2y. \end{aligned} \tag{3}$$

In Singular we call:

```
> LIB "grobcov.lib"7;
> ring R=(0,x,y),(x1,y1,x2,y2,x3,y3),dp;
> ideal F0= ---;
> locusdg(grobcov(F0));
```

where in F0= ---, the lines are to be substituted by Eqs. (3) of the ideal. We obtain:

```
[1]:
   [1]:
      _[1]=(3x^2+xy-9x+2y^2+3y)
   [2]:
      [1]:
         _[1]=(y^2+8y+65)
         _[2]=(7x-y-60)
      [2]:
         _[1]=(2y+5)
         _[2]=(2x-1)
      [3]:
         _[1]=(4y+7)
         _[2]=(2x-7)
   [3]:
      Normal,1
```

which, as expected, gives the conic of formula (2) specialized for the concrete values of the parameters, from which two real points $(1/2, -5/2)$, $(7/2, -7/4)$ and two complex points $(8 + i, -4 + 7i)$, $(8 - i, -4 - 7i)$, are excluded.

Once the locus equation is known, it is trivial to check that the conic is tangent to lines AC and BC, a statement mentioned by Sutherland in [7].

This locus is Example 10 in our prototype [35]. By clicking the *Find locus* button, we obtain the picture of Fig. 3, where the description of the locus as a conic with two missing (real) points is provided.

It is instructive to analyze where the missing points come from. Dragging the mover point G to make line CG parallel to line a, makes line BI change, approaching a limit position parallel to CG and a. Thus BI and a do not intersect, *I goes to infinity*, and the system has no solution. This happens for the missing point $(1/2, -5/2)$. Things are analogous for the missing point $(7/2, -7/4)$.

5.2. Example 2: offset of a circle

Although in this paper we focus on locus computation, our approach can be efficiently used for computing other derived elements in a geometric construction, as it will be reported in a future note. Consider, for instance, the 1-offset of a circle g centered at the origin with radius 1. The offset is described by the system consisting of the equations of the base circle, the family f of circles enveloping the offset, and the expression $\frac{\partial f}{\partial a}\frac{\partial g}{\partial b} - \frac{\partial f}{\partial b}\frac{\partial g}{\partial a}$:

$$\begin{aligned} F = {} & a^2 + b^2 - 1, \\ & (x - a)^2 + (y - b)^2 - 1, \\ & 4(y - b)a - 4(x - a)b. \end{aligned} \tag{4}$$

Applying the GröbnerCover algorithm to the ideal

$$\begin{aligned} \mathfrak{I} = {} & \langle a^2 + b^2 - 1, (x - a)^2 + (y - b)^2 - 1, \\ & 4(y - b)a - 4(x - a)b\rangle, \end{aligned}$$

we obtain the following three segments:

Nr.	Segment	Basis	lpp
1	$\mathbb{C}^2 \setminus (\mathbf{V}(x^2 + y^2 - 4) \cup \mathbf{V}(y, x))$	$\{1\}$	$\{1\}$
2	$\mathbf{V}(x^2 + y^2 - 4)$	$\{2b-y, 2a-x\}$	$\{b, a\}$
3	$\mathbf{V}(y, x)$	$\{a^2 + b^2 - 1\}$	$\{a^2\}$

The Locus algorithm produces two disjoint components with different character:

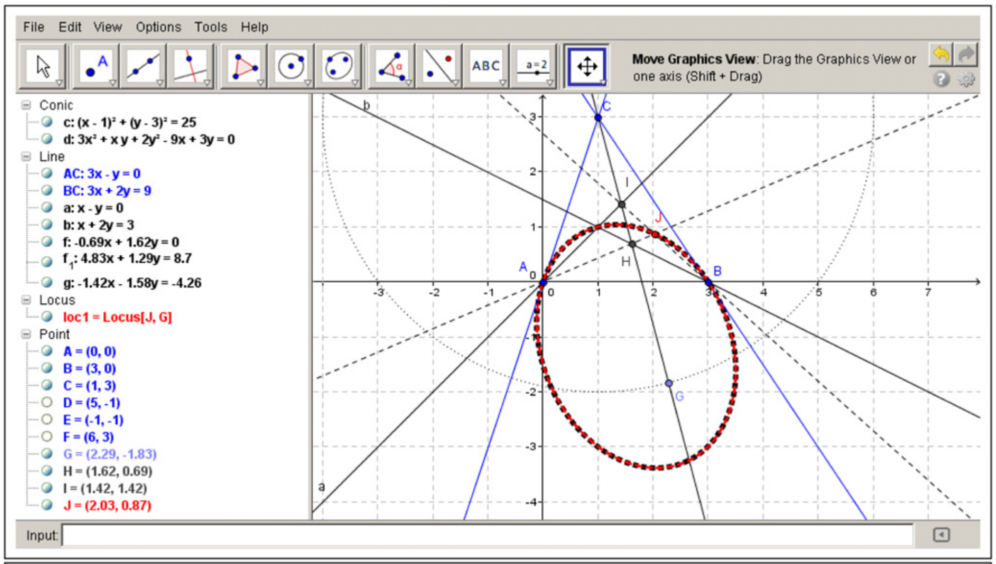
$\mathbf{V}(x^2 + y^2 - 4)$ Normal

$\mathbf{V}(y, x) \setminus \mathbf{V}(1)$ Accumulation.

The normal component is the circle of radius 2, as expected. For the accumulation point $(0, 0)$ we have $\pi_2(\pi_1^{-1}(0, 0)) = \mathbf{V}(a^2 + b^2 - 1)$ and the whole basic circle is part of the solution (a 1-dimensional set of points).

6 Available at http://www.cl.cam.ac.uk/techreports/.

7 Library available at http://www-ma2.upc.edu/montes/.


File
Edit
View
Options
Tools
Help
ABC
a=2
Move Graphics View: Drag the Graphics View or one axis (Shift + Drag)
Input:

Conic
c: (x - 1)² + (y - 3)² = 25
d: 3x² + x y + 2y² - 9x + 3y = 0
Line
AC: 3x - y = 0
BC: 3x + 2y = 9
a: x - y = 0
b: x + 2y = 3
f: -0.69x + 1.62y = 0
f_1: 4.83x + 1.29y = 8.7
g: -1.42x - 1.58y = -4.26
Locus
loc1 = Locus[J, G]
Point
A = (0, 0)
B = (3, 0)
C = (1, 3)
D = (5, -1)
E = (-1, -1)
F = (6, 3)
G = (2.29, -1.83)
H = (1.62, 0.69)
I = (1.42, 1.42)
J = (2.03, 0.87)

RESULT: The sought locus (graphed in applet - dotted black) is the set:

$$\mathbb{V}(3x^2+xy+2y^2-9x+3y)\setminus[\mathbb{V}(y^2+8y+65,7x-y-60)\cup\mathbb{V}(2y+5,2x-1)\cup\mathbb{V}(4y+7,2x-7)]$$

Fig. 3. Description of locus in Example 1 as provided by the prototype.

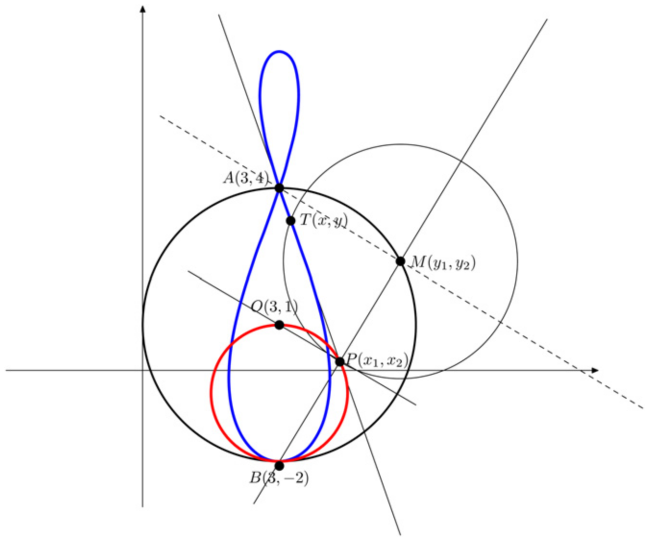


Fig. 4. Locus described by T (and P) as M runs along its circle.

5.3. Example 3: detecting bad mover positions

When illustrating the prototype (Fig. 2) we considered the limaçon of Pascal, showing that the extra circle $x^2+y^2-6x-8y+24=0$ comes from a degeneration due to the coincidence of points M and B. It can happen that a degeneracy of the construction forces the GröbnerCover algorithm to consider the whole space of parameters as solution. More concretely, the first segment of the GröbnerCover algorithm is called the *generic* segment. It is the unique open segment in the whole parameter space, i.e. it consists of the whole parameter space except a variety (of dimension less than the one of the parameter space itself).

We assumed in the definition of the locus, that the generic segment has basis $\{1\}$, i.e. there is no solution of the system on it, as the locus is expected to be of dimension less than the parameter space. Nevertheless, as mentioned above, it could happen that a construction *collapses* for some values of the variables. For these values, the number of constraints decreases and almost all points are valid parameter values for the system having a solution.

As an example, consider the following locus construction (see Fig. 4). The point $M(y_1, y_2)$ runs over the circle with center at $O(3,1)$ and radius OA, where $A=(3,4)$. We construct the line parallel to the line AM passing through O and the line perpendicular to it passing through the point $B=(3,-2)$. Both lines intersect at point $P(x_1,x_2)$. Construct the line AP and the circle with center M and radius MP. We define this intersection as the tracer point(s): $T(x,y)$.

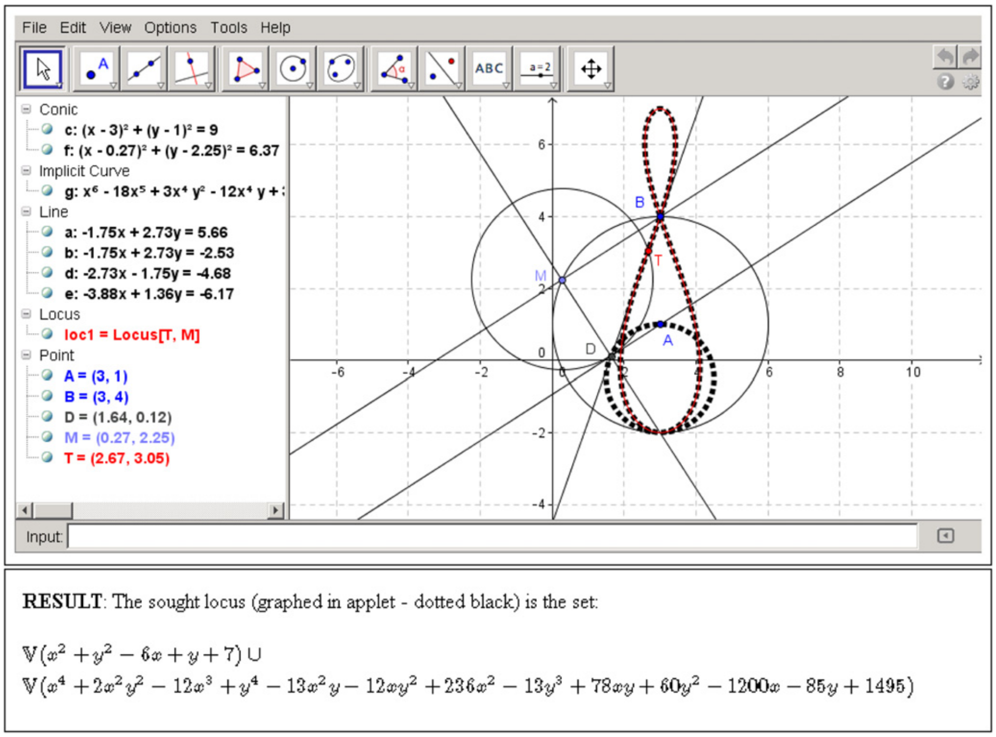
The polynomial system describing the problem is the ideal F given by

$$\begin{aligned} F = \langle & (y_1-3)^2+(y_2-1)^2-9,\\ & (4-y_2)(x_1-3)+(y_1-3)(x_2-1),\\ & (y_1-3)(x_1-3)-(4-y_2)(x_2+2),\\ & (4-x_2)x+(x_1-3)y+3x_2-4x_1,\\ & (x-y_1)^2+(y-y_2)^2-(y_1-x_1)^2-(y_2-x_2)^2\rangle. \end{aligned} \tag{5}$$

When M coincides with A (i.e. $y_1=3, y_2=4$), the above system reduces to

$$\begin{aligned} F = \langle & (3-3)^2+(4-1)^2-9,\\ & (4-4)(x_1-3)+(3-3)(x_2-1),\\ & (3-3)(x_1-3)-(4-4)(x_2+2),\\ & (4-x_2)x+(x_1-3)y+3x_2-4x_1,\\ & (x-3)^2+(y-4)^2-(3-x_1)^2-(4-x_2)^2\rangle. \end{aligned} \tag{6}$$

Thus, every point in the plane satisfies it. Note that since the line AM is undefined, there are no constraints on T, which can be then placed anywhere in the plane. Since we are computing loci at most linear, the generic segment can be discarded without losing solutions. This is the approach currently used in the prototype.



Conic
c: $(x-3)^2 + (y-1)^2 = 9$
f: $(x-0.27)^2 + (y-2.25)^2 = 6.37$
Implicit Curve
g: $x^6 - 18x^5 + 3x^4 y^2 - 12x^4 y +$
Line
a: $-1.75x + 2.73y = 5.66$
b: $-1.75x + 2.73y = -2.53$
d: $-2.73x - 1.75y = -4.68$
e: $-3.88x + 1.36y = -6.17$
Locus
loc1 = Locus[T, M]
Point
A = (3, 1)
B = (3, 4)
D = (1.64, 0.12)
M = (0.27, 2.25)
T = (2.67, 3.05)

RESULT: The sought locus (graphed in applet - dotted black) is the set:

$\mathbb{V}(x^2+y^2-6x+y+7)\cup$
$\mathbb{V}(x^4+2x^2y^2-12x^3+y^4-13x^2y-12xy^2+236x^2-13y^3+78xy+60y^2-1200x-85y+1495)$

Fig. 5. Description of locus as provided by the prototype.

Finally, the result consists of two irreducible normal components

$\mathbf{V}(x^2-6x+y^2+y+7)$
$\mathbf{V}(x^4-12x^3+2x^2y^2-13x^2y+236x^2-12xy^2+78xy$
$\quad -1200x+y^4-13y^3+60y^2-85y+1495).$

This locus is Example 9 in our prototype [35]. By clicking the *Find locus* button, we obtain the locus description shown on Fig. 5.

5.4. *Example 4: automatic deduction of the Steiner–Lehmus theorem*

In the previous locus examples, the mover point is constrained to a one dimensional object, so to sample its path, the user/system has to perform a *bound dragging*, as defined in [6]. There, Arzarello et al. introduce other types of dragging in order to analyze different kinds of student interactions with dynamic constructions. The process of searching for plausible geometric conjectures is often closely related to dragging manipulations. One of these dragging modalities, the *dummy locus dragging*, is defined as

> …moving a basic point so that the drawing *keeps* a discovered property; the point which is moved follows a path, even if the users do not realize this: the locus is not visible and does not 'speak' to the students, who do not always realize that they are dragging along a locus.

We show a non trivial illustration of this kind of locus: we use our Locus algorithm to prove the classic Steiner–Lehmus theorem that establishes necessary and sufficient conditions for a triangle to have two equal-length bisectors (we refer the reader to [39] for details, where a detailed study of the theorem in relation to the GröbnerCover algorithm is discussed).

More concretely, let us consider the locus set of points C for which the theorem is true; that is, given a triangle ABC, we search for the points C for which one bisector at angle A is equal to one bisector at angle B (internal or external bisectors, see Fig. 6).

Setting points A and B as origin and unit respectively, and assigning coordinates $C(x,y), M(x_1,y_1), T(x_2,y_2), P(p,0), R(r,0),$

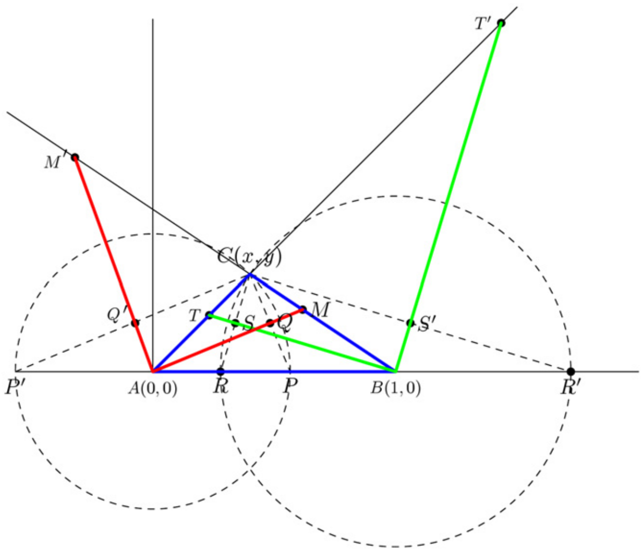


Fig. 6. One bisector at A ($\overline{AM}$ or $\overline{AM'}$) is equal to one bisector at B ($\overline{BT}$ or $\overline{BT'}$).

the construction leads to the following polynomial system:

$$\begin{cases} x^2+y^2-p^2, \\ yx_1-(x+p)y_1, \\ y(1-x_1)+(x-1)y_1, \\ (x-1)^2+y^2-(r-1)^2, \\ y(1-x_2)+(x+r-2)y_2, \\ xy_2-yx_2, \\ x_1^2+y_1^2=(x_2-1)^2+y_2^2. \end{cases}$$

The GröbnerCover algorithm applied to this system provides 9 segments, each of them having specific properties concerning the number of solutions, and the Locus algorithm group them into components. From the locus perspective we are only interested in

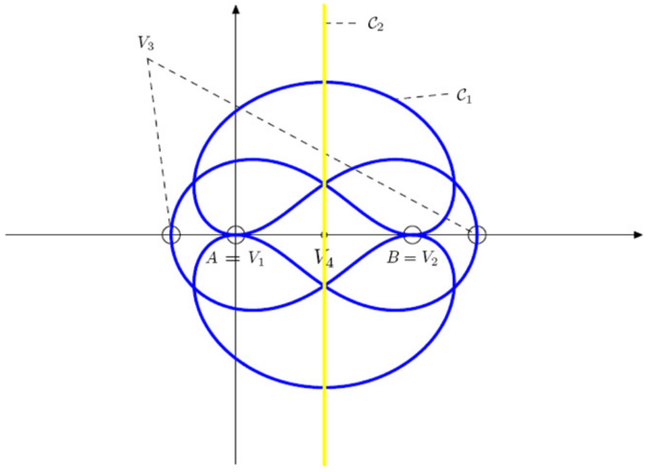


Fig. 7. Locus of the Steiner–Lehmus theorem.

the normal and accumulation solutions. Applying the Locus algorithm to the grobcov output, we obtain two normal components and a degenerate one. In the description, the following curve appears:

$$\begin{aligned}\mathcal{C}_1 = \mathbf{V}(&8x^{10} - 40x^9 + 41x^8y^2 + 76x^8 - 164x^7y^2\\ &- 64x^7 + 84x^6y^4 + 246x^6y^2 + 16x^6 - 252x^5y^4\\ &- 164x^5y^2 + 8x^5 + 86x^4y^6 + 278x^4y^4 + 31x^4y^2\\ &- 4x^4 - 172x^3y^6 - 136x^3y^4 + 20x^3y^2 + 44x^2y^8\\ &+ 122x^2y^6 + 14x^2y^4 - 10x^2y^2 - 44xy^8 - 36xy^6\\ &+ 12xy^4 + 9y^{10} + 14y^8 - y^6 - 6y^4 + y^2).\end{aligned}$$

The components, with their character, are:

$\mathcal{C}_1 \setminus \big(\mathbf{V}(y, x) \cup \mathbf{V}(y, x-1) \cup \mathbf{V}(y, 2x^2 - 2x - 1)\big)$ Normal
$\mathbf{V}(2x - 1) \setminus \mathbf{V}(y, 2x - 1)$ Normal
$\mathbf{V}(y)$ Degenerate.

In [39], the whole GröbnerCover algorithm output is analyzed, using the sign of the variables p and q on the solutions, and a detailed study of which parts of the curves and special points correspond to which equalities between bisectors of A and B. In particular, the second component corresponds to the classical Steiner–Lehmus theorem, well known since the XIX century, where the inner bisector of A is equal to the inner bisector of B. The fact that the outer bisectors are also equal over this component is a new result obtained as a side product.

It is worth remarking that the first component has only been known since the development of computer algebra methods. The third component, as well as the holes of the two normal components, correspond to degenerate triangles. Fig. 7 shows all three locus components.

Using our approach, this kind of ''dummy'' locus computation in DG systems could be easily automated, thereby allowing students to tackle general *what if* questions.

6. Conclusion

In this paper, a taxonomy for locus computation in dynamic geometry is proposed. By using the efficient GröbnerCover algorithm for parametric polynomial systems solving, any interactive construction involving a linear locus is automatically analyzed, and the locus solutions are grouped in such a way that the geometrically relevant locus components are returned.

This taxonomy efficiently classifies the algebraic parts of loci, allowing users and systems to advance into a more reliable automatic description of geometric constructions. Using this classification, a prototype web application based on a well-known dynamic geometry system that automatically identifies the different components of a locus has been implemented.

Although limited to the complex field, experimental results show that our approach is both effective and efficient from a practical point of view, making this technology mature enough to be incorporated in forthcoming versions of standard interactive environments. However, since most of these systems currently use a variety-oriented representation of geometric objects, an extension of their data structure would be required for the systems to completely profit from our results.

Acknowledgments

Authors are partially supported by the Spanish ''Ministerio de Economía y Competitividad'' and by the European Regional Development Fund (ERDF), under the Project MTM2011-25816-C02-02. The third author was supported by projects Gen. Cat. DGR 2009SGR1040 and MICINN MTM2009-07242.

References

[1] Laborde JM, Bellemain F. Cabri geometry II. Dallas: Texas Instruments; 1998.
[2] Jackiw N. The geometer's sketchpad v 4.0. Key Curriculum Press; 2002.
[3] GeoGebra, http://www.geogebra.org [Last accessed February 2014].
[4] Java Geometry Expert. http://www.cs.wichita.edu/~ye/ [Last accessed February 2014].
[5] Gao XS. Automated geometry diagram construction and engineering geometry. In: Gao XS, Wang D, Yang L, editors. ADG 1998. Lecture notes in artificial intelligence, vol. 1669. Heidelberg: Springer; 1999. p. 232–57.
[6] Arzarello F, Olivero F, Paola D, Robutti O. A cognitive analysis of dragging practises in Cabri environments. ZDM, Zentralbl Didakt Math 2002;34(3): 66–72. http://dx.doi.org/10.1007/BF02655708.
[7] Sutherland IE. Sketchpad: a man–machine graphical communication system. Tech. rep. 574. Computer Laboratory, University of Cambridge; 2003.
[8] Botana F. Interactive versus symbolic approaches to plane loci generation in dynamic geometry environments. In: Sloot PM, Hoekstra A, Tan CK, Dongarra JJ, editors. Computational science—ICCS 2002. Lecture notes in computer science, vol. 2330. Berlin (Heidelberg): Springer; 2002. p. 211–8. http://dx.doi.org/10.1007/3-540-46080-2-22.
[9] Schumann H. A dynamic approach to simple algebraic curves. ZDM, Zentralbl Didakt Math 2003;35:301–16.
[10] Losada R, Recio T, Valcarce JL. On the automatic discovery of Steiner–Lehmus generalizations, in: Richter-Gebert J., Schrek P, editors. Proceedings of ADG'2010, München. 2010. p. 171–4.
[11] Botana F, Abanades M, Escribano J. Exact internet accessible computation of paths of points in planar linkages and diagrams. Comput Appl Eng Educ 2011; 19:835–41.
[12] Lebmeir P, Richter-Gebert J. Recognition of computationally constructed loci. In: Botana F, Recio T, editors. Automated deduction in geometry. Lecture notes in computer science, vol. 4869. Berlin (Heidelberg): Springer; 2007. p. 52–67. http://dx.doi.org/10.1007/978-3-540-77356-6_4.
[13] GeoGebra GSoC. [link]. URL: http://www.geogebra.org/trac/wiki/LocusLineEquation.
[14] Kortenkamp U, Richter-Gebert J. Using automatic theorem proving to improve the usability of geometry software, in: Proceedings of the mathematical user-interfaces workshop, vol. 2004. 2004.
[15] Kortenkamp U. Foundations of dynamic geometry (Ph.D. thesis). Zurich: Swiss Federal Institute of Technology; 1999.
[16] Richter-Gebert J, Kortenkamp U. The interactive geometry software cinderella. Berlin: Springer; 1999.
[17] Bates DJ, Hauenstein JD, Sommese AJ, Wampler CW. Numerically solving polynomial systems with Bertini, vol. 25. SIAM; 2013.
[18] Buchberger B. Gröbner bases: an algorithmic method in polynomial ideal theory. In: Bose NK, editor. Multidimensional systems theory. Dordrecht (Netherlands): Reidel; 1985. p. 184–232. [Chapter] Gröbner bases: an algorithmic method in polynomial ideal theory.
[19] Buchberger B. Bruno Buchberger's Ph.D. thesis 1965: an algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. J Symbolic Comput 2006;41(3–4):475–511. http://dx.doi.org/10.1016/j.jsc.2005.09.007. (http://www.sciencedirect.com/science/article/pii/S0747717105001483).
[20] Wu WT. Mechanical theorem proving in geometries. Vienna: Springer; 1994.
[21] Chou SC. Mechanical geometry theorem proving. Dordrecht, Netherlands: D. Reidel Publishing Company; 1988.
[22] Chou SC. Proving elementary geometry theorems using Wu's algorithm. In: Bledsoe WW, Loveland DW, editors. Automated theorem proving: after 25 years. Contemporary mathematics, vol. 29. American Mathematical Society; 1984. p. 243–86.

[23] Roanes-Macías E, Roanes-Lozano E. Búsqueda automática de lugares geométricos. Bol Soc Puig Adam 1999;53:67–77.
[24] Roanes-Macías E, Roanes-Lozano E. Automatic determination of geometric loci. 3D-extension of Simson–Steiner theorem. In: Campbell J, Roanes-Lozano E, editors. Artificial intelligence and symbolic computation. Lecture notes in computer science, vol. 1930. Berlin (Heidelberg): Springer; 2001. p. 157–73. http://dx.doi.org/10.1007/3-540-44990-6_12.
[25] Wang D. Geother: a geometry theorem prover. In: McRobbie MA, Slaney JK, editors. Automated deduction—CADE-13. Lectures notes in computer science, vol. 1104. Springer; 1996. p. 166–70.
[26] Kapur D. Geometry theorem proving using Hilbert's Nullstellensatz. In: Char BW, editor. SYMSAC'86: proceedings of the fifth ACM symposium on symbolic and algebraic computation. New York (NY, USA): ACM Press; 1986. p. 202–8.
[27] Kapur D. Using Gröbner bases to reason about geometry problems. J Symbolic Comput 1986;2(4):399–408.
[28] Kutzler B, Stifter S. Automated geometry theorem proving using Buchberger's algorithm. In: Char BW, editor. SYMSAC'86: proceedings of the fifth ACM symposium on symbolic and algebraic computation. New York (NY, USA): ACM Press; 1986. p. 209–14.
[29] Recio T, Vélez MP. Automatic discovery of theorems in elementary geometry. J Automat Reason 1999;23:63–82.
[30] Botana F, Valcarce JL. A software tool for the investigation of plane loci. Math Comput Simul 2003;61(2):139–52.
[31] Gerhäuser M, Wassermann A. Automatic calculation of plane loci using Gröbner bases and integration into a dynamic geometry system. In: Schreck P, Narboux J, Richter-Gebert J, editors. Automated deduction in geometry. Lecture notes in computer science, vol. 6877. Berlin (Heidelberg): Springer; 2011. p. 68–77. http://dx.doi.org/10.1007/978-3-642-25070-5_4.
[32] Escribano J, Botana F, Abánades MA. Adding remote computational capabilities to dynamic geometry systems. Math Comput Simul 2010;80:1177–84.
[33] Montes A, Wibmer M. Gröbner bases for polynomial systems with parameters. J Symbolic Comput 2010;45:1391–425.
[34] Grothendieck A, Dieudonné J. Eléments de géométrie algébrique. Le langage des schèmes. Vol. 166. Springer-Verlag; 1971.
[35] Locus Prototype, http://webs.uvigo.es/fbotana/LocusGC/. 2012.
[36] Singular, http://www.singular.uni-kl.de/ [Last accessed February 2014].
[37] Sage, Sage cell server [Last accessed February 2014]. URL: https://github.com/sagemath/sagecell.
[38] Botana F. On the parametric representation of dynamic geometry constructions. In: Murgante B, Gervasi O, Iglesias A, Taniar D, Apduhan B, editors. Computational science and its applications—ICCSA 2011. Lecture notes in computer science, vol. 6785. Berlin (Heidelberg): Springer; 2011. p. 342–52. http://dx.doi.org/10.1007/978-3-642-21898-9_30.
[39] Montes A, Recio T. Generalizing the Steiner–Lehmus theorem using the Gröbner cover. Math Comput Simul 2014; [in press] URL: http://dx.doi.org/10.1016/j.matcom.2013.06.006.