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/

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.