A Parametric Approach to 3D Dynamic Geometry

Francisco Botana

Departamento de Matemática Aplicada I, Universidad de Vigo, Campus A Xunqueira, 36005 Pontevedra, Spain

Abstract

Dynamic geometry systems are computer applications allowing the exact on–screen drawing of geometric diagrams and their interactive manipulation by mouse dragging. Whereas there exists an extensive list of 2D dynamic geometry environments, the number of 3D systems is reduced. Most of them, both in 2D and 3D, share a common approach for the management of geometric constructions, using numerical data for the present diagram instance and elementary methods for computing derived objects.

This paper deals with a parametric approach for automatic management of 3D Euclidean constructions. An open source library, implementing the core functions in a 3D dynamic geometry system, is described here. The library deals with constructions by using symbolic parameters, so allowing a full algebraic knowledge about objects such as loci and envelopes. This parametric approach is also a prerequisite for performing automatic proof, and basic functions are defined for symbolically checking the truth of statements. Using recent results coming from the theory of parametric polynomial systems solving, the bottleneck in the automatic determination of geometric loci and envelopes is solved. As far as we know, there is no comparable library in the 3D case, except the paramGeo3D library (designed for computing equations of simple 3D geometric objects, but it lacks specific functions for finding loci and envelopes).

Keywords: 3D Dynamic Geometry, automated deduction, Gröbner bases, parametric polynomial systems, degenerated conditions

1. Introduction

A method for mechanical theorem proving in geometries was proposed by Wu [28, 29] during the eighties, and, almost simultaneously, a novel applica-

tion for the algorithm of Buchberger [8], also related with automated deduction, was studied (see, for instance, [15]). Ending this decade two learning environments for geometry appeared in the educational software market, The Geometer's Sketchpad, GSP, [38] and Cabri [32]. These programs marked the birth of the dynamic geometry (DG) paradigm, and formed the core of mathematical sofwtare used in schools.

A first implementation of the algorithm of Wu was done by Chou [9], followed by the work of Wang [26], and a DG environment using Wu's method, Geometry Expert, GEX, was developed [13]. Regarding the DG paradigm, the novel approach in GEX relied on using formal proving methods rather than the visual convincement offered by Cabri or GSP. Some authors tried to remedy this shortcoming of popular DG software through intercommunication with Computer Algebra Systems (CAS). For example, Roanes–Lozano et al. [21] use GSP constructions to perform proving tasks by means of a Maple library with a simple implementation of Wu's method. Botana and Valcarce [1, 3] develop a DG environment that communicates with CoCoA [10] and Mathematica, and where Gröbner bases are used for loci finding and automated proof and discovery. Botana has also implemented a Mathematica based web application [4] for remotely dealing with 3D geometric constructions, and Roanes–Lozano et al. [22] have replied this approach using Maple with local access only.

In this paper we deal with a parametric description of 3D Euclidean constructions and the involved algorithms to perform automatic discovery processes in 3D DG systems. In Section 2 we state what a parametrical description of a geometric construction is and we review the state of the art concerning these approaches in 3D. Section 3 describes an open source library implementing our parametric setting, extends an elimination based approach for the automatic finding of new objects (loci and envelopes) in geometric constructions, and illustrates the proposal through selected examples. Section 4 discusses some drawbacks of the elimination techniques and introduces a new application of the theory of parametric systems solving for overcoming these limitations.

2. Parametric description of dynamic geometry constructions

2.1. Issues in 2D

Most DG systems incorporate an option that allows users to see the constructive steps of constructions. Although there is no a common terminology

for it, standard systems such as the above cited Cabri and GSP, or the emerging educational standard GeoGebra [35], can exhibit an almost natural language description of the steps in a construction. For instance, consider a circle in the plane defined by its center $O(0,0)$ and passing through $A(0,2)$. Define a point P lying on the circle and a new circle centered at P with radius 1. Drawing a line through A and P, define a common point X on the line and the last circle. Finally, ask for the locus of X when P moves along the basic circle. Figure 1 shows this construction and its step–by–step description in GeoGebra.

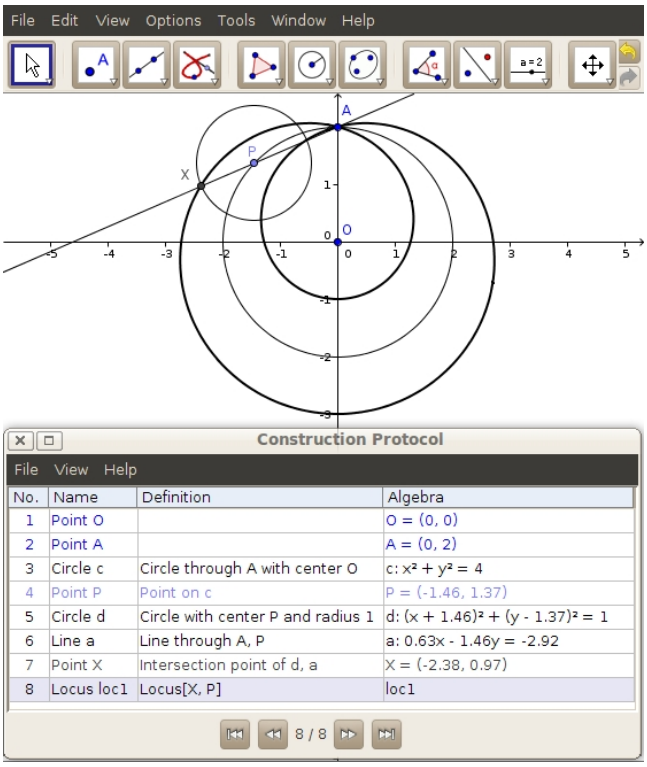

Figure 1: A GeoGebra construction and its construction protocol (bottom).

Although primarily introduced for didactic purposes, this kind of descriptions has been used to capture the ordered constructive steps for other goals. Being the first standard DG systems, GSP and Cabri, proprietary software, their code was not released and their constructions were encrypted or, at least, specified through a not easily accessible grammar. These were the main reasons why this option of exporting the construction protocol was used for

further processing. The appearance of the GeoGebra open source system which specifies its constructions by using XML (a decision also adopted by other systems), has made unnnecesary recurring to the protocol of constructions. In any case, we assume here that the constructive part of a DG diagram can be easily read. We do not give here the details of such an approach, referring to [6], where a prototype reading GeoGebra constructions is described. Indeed, the European project Intergeo [37] had among its goals the specification of a common format for file exchange between DG systems. Although this format is currently unfinished, some experiments about embedding it in an automated deduction environment are reported in [7].

Going back to the GeoGebra construction, note that the first three steps consist of defining points O and A and the circle c. Whereas the points do not have any text on the Definition field of the Construction protocol, the circle is recorded via its constructive statement. At the same time, the Algebra field shows the coordinates of the points and the equation of the circle. As it is well–known, if the user drags, let us say, A, the present position of the point changes and so they do accordingly the trace and equation of the circle. We can interpret this behaviour as a parametric one: the value of the variable (the circle c) is refreshed whenever its parameters (the points O, A) are updated. Nevertheless, DG developers usually rely on numeric assignments, that is, once moved A the system recomputes the equation of c and gets this numeric expression as the whole algebraic knowledge about the circle. Now, placing a point P on c is again a parametric issue: given the position of the pointer on the screen and the equation of c, compute P coordinates satisfying both requirements and with a predetermined numeric precision. Thus, refreshing the drawing when some element is dragged is just a problem of moving through the data structure that encompasses the construction. So, any question involving a generic point P placed on c cannot be generally answered. This numeric approach faces its limitations when computing the last object in the considered example. In order to compute the locus of X when P moves along circle c, the standard strategy in DG consists of sampling the path of P and getting for each sample element a position for X. Joining the X positions, a line on the screen is returned as the sought locus. This method has two inherent limitations: no algebraic knowledge about the locus is obtained, and, since the sampling strategy must be a general one, returned loci sometimes exhibit anomalous behavior (see [2] for an in–depth study of this issue).

Using any of the above cited methods for geometric theorem proving

this locus can be easily computed. Nevertheless, since both are based on coordinates, the construction must be rewritten, for instance, as follows:

```
Point O(0,0); Point A(0,2); Circle(c,O,A); PointOn(P,c)
Circle(d,P,1); Line(a,A,P); Intersection(X,d,a); Locus(loc1,X,P)
```

Once done, each item can be used to extract algebraic knowledge about it: `Circle(c,O,A)` gives $x^2 + y^2 - 4$, `PointOn(P,c)` defines a point on c, so it can be represented as $P(x_1, x_2)$ where $x_1^2 + x_2^2 - 4 = 0$, and so on. Thus, a parametrical description of the construction is

- Point O: coordinates $(0, 0)$
- Point A: coordinates $(0, 2)$
- Circle c: equation $x^2 + y^2 - 4 = 0$
- Point P: coordinates (x_1, x_2), constraints $x_1^2 + x_2^2 - 4 = 0$
- Circle d: equation $(x - x_1)^2 + (y - x_2)^2 - 1 = 0$
- Line a: equation $x(x_2 - 2) - x_1(y - 2) = 0$
- Point X: coordinates (x_3, x_4), constraints $(x_3 - x_1)^2 + (x_4 - x_2)^2 - 1 = 0, x_3(x_2 - 2) - x_1(x_4 - 2) = 0$
- Locus $loc1$: equation $(x^4 + 2x^2y^2 + y^4 - 9x^2 - 9y^2 + 4y + 12)(x^2 + y^2 - 4y + 3) = 0$

In this list, all items except the last one are characterised in an elementary way. If the item is a point, assign it the coordinates given by the user if it is a basic point, or two symbolic coordinates and its definitory constraints. If the item is not a point, take its algebraic description (note that the scope of this approach is then limited to an algebraic realm. Segments, for instance, cannot be properly described). The Locus item is not an elementary one, and it can be computed through the procedure described in [3]. We shortly recall the method for this case. Collecting all the information we have about points we get the following polynomials:

$$x_1^2 + x_2^2 - 4, (x_3 - x_1)^2 + (x_4 - x_2)^2 - 1, x_3(x_2 - 2) - x_1(x_4 - 2).$$

An algebraic description for $loc1$ can then be computed, roughly speaking, through the elimination of variables x_1, x_2. This elimination returns (after

substituting X coordinates by the generic x, y ones) a polynomial which factors to

$$(x^4 + 2x^2y^2 + y^4 - 9x^2 - 9y^2 + 4y + 12)(x^2 + y^2 - 4y + 3).$$

We take the 0–set of the above polynomial as the set defining the locus. As it is well–known, the equation of this limaçon of Pascal includes only the first factor. The second factor, a circle centered at A with radius 1 appears due to the degeneration of line a when P and A coincide. So, the locus equation must be understood as a superset of the true locus.

This locus computation illustrates the advantages of working with a parametric description of the geometric construction. It should be noted that replacing the numeric coordinates of basic points O, A with symbolic ones, the procedure would return a polynomial in six variables. Any valuation of the basic points coordinates would solve a specific instance of the locus problem. This approach, optional when computing derived objects, such as loci and envelopes, is necessary when performing general deduction tasks. While some standard DG systems offer integrated checkers testing the correctness of statements about a construction, they are usually limited to test it numerically.

Besides the algebraic limitation of the parametric approach, there are other sources of imprecision in the translation between standard constructions and parametrized ones. One refers to quadratic objects: while DG software usually defines two different points when intersecting, say a circle and a line, a coherent parametric setting just defines one, delaying the computation of the present value of common points, in a kind of lazy evaluation. This strategy is more general than a literal translation, and it is able to cope with intersection of curves with higher degrees. The second source of imprecision deals with different results in both approaches. Whenever a dependent point with one degree of freedom is used to define a new element, DG developers use its present numeric coordinates, whereas in the parametric view the symbolic coordinates of the point are considered. So, it may happen that both descriptions refer to different constructions. The finding of a conchoid (Fig. 2) will illustrate this issue. Consider the points $A(0, 2)$, B and C moving on the x and y axes, respectively. The trace of the point X lying on the line AB and on the circle centered at B and with radius $\overline{AC}$ is returned by GeoGebra as a conchoid branch (note that in order to get the whole conchoid it would be necessary tracing the other intersection point).

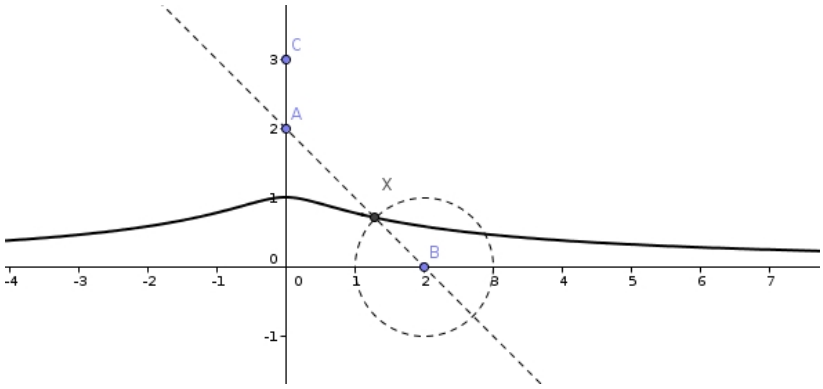

Figure 2: Tracing a conchoid.

A parametric description is

- Point A: coordinates $(0, 2)$
- Point B: coordinates (x_1, x_2), constraints $x_2 = 0$
- Point C: coordinates (x_3, x_4), constraints $x_3 = 0$
- Line AB: equation $(y - x_2)x_1 - x(x_2 - 2) = 0$
- Circle $B, \overline{AC}$: equation $(x - x_1)^2 + (y - x_2)^2 - x_3^2 - (2 - x_4)^2 = 0$
- Point X: coordinates (x_5, x_6), constraints $(x_6 - x_2)x_1 - x_5(x_2 - 2) = 0, (x_5 - x_1)^2 + (x_6 - x_2)^2 - x_3^2 - (2 - x_4)^2 = 0$
- Locus $loc1$: equation $0 = 0$

where the last equation, computed through elimination, means that the locus is (or is contained in) the whole plane, that is, the elimination ideal I_{x_1,x_2,x_3,x_4} of

$$\langle x_2, x_3, (x_6 - x_2)x_1 - x_5(x_2 - 2), (x_5 - x_1)^2 + (x_6 - x_2)^2 - x_3^2 - (2 - x_4)^2 \rangle$$

is $\langle 0 \rangle$, and the generated variety is $\mathbb{C}^2$. The reason behind this surprising answer is that C is defined as a moving point on the y axis without other constraints (i.e. x_4 can have any value). So, despite the graphical intuition, the circle is not a specific one, but it sweeps the entire plane, as consequently the point X also does.

2.2. State of the art in 3D

While the list of DG systems in 2D can be impressive, there are just a few systems implementing the DG paradigm for 3D: Cabri3D [33], Calques3D [34], Archimedes3D [31] and a beta version of GeoGebra 5.0 [36], to mention the most known ones. As their 2D counterparts, they allow some mechanism to inspect their constructions (via the XML specification of constructions or any ad–hoc procedure, as for the 2D case). We do not study here the problem of getting the textual specification of 3D geometric constructions. Our interest here is reviewing past work on the parametric description of their configurations.

As far as we know, there are just two published proposals about dealing parametrically with 3D DG constructions: LD–3D[4] and paramGeo3D[20]. They are libraries written in Mathematica and CoCoA (LD–3D) and Maple (paramGeo3D) where the user can introduce basic and constructed 3D geometric objects, in such a way that the obtained data are parametrically expressed. The main interest of these libraries comes from their working implementation of Wu and Gröbner methods for performing automatic deduction. Both authors have also explored the integration of their libraries with the same 3D DG system, Calques3D (which runs under Windows only). To this end, the author of Calques3D extended its system after our request, allowing the textual export of its constructions in a Mathematica–like sintax in 2002, and to Maple sintax some years after. Nevertheless, there are not significative differences between both formats, being a trivial task the translation between them.

LD–3D offered a web–based access to solve loci finding tasks (although some kind of automatic proof was also included). Uploading the Calques3D specification of a locus problem, a remote server returned the equation of the locus and an applet with the graphical representation of the object. On its side, paramGeo3D offers only local access while incorporating a more general approach to automatic proof, based on well-known stablished approaches. The mechanism is quite similar: the user loads, in a Maple session, the paramGeo3D library, a translator procedure and the Calques3D textual specification. Once done, the user must interact with Maple in order to get the sought results. This is the key difference between both approaches: the main goal of LD–3D was automating loci computations. That is, enriching the expressive power of Calques3D by means of symbolic techniques in such a way that no further human intervention is necessary.

Some words must be said concerning the open/proprietary character of the libraries and the ways to access them. There are economic and scientific issues involved.

Once deployed the 3D–LD server, its sustainability required paying a yearly fee, an important point in current times of budget cuts in non-profit institutions. Furthermore, choosing a proprietary CAS as development platform, one can be forced to use algorithms under commercial patents or, even worse, undocumented ones. There are also intercommunication issues: whereas 3D–LD used CoCoA for efficiently computing Gröbner bases, we have moved to Singular [12] since it gives, at least, a similar efficiency, and currently only Singular has implementations for other algorithms required in our field of work.

Concerning the ways of accessing the libraries, we think that it is unfair, from an non–profit scientific institution point of view, distributing DG systems or related materials requiring the use of expensive CAS. From an educational perspective, it is necessary that the libraries could be remotely used, since they involve huge pieces of code, currently without easy installation in present educational hardware.

3D–LD and paramGeo3D share some of the above criticism: they are developed under commercial CAS, so making their incorporation hard to 3D DG environments. Although 3D–LD offers remote access, it is mainly focused to loci finding, forgetting other derived objects in dynamic geometry and without general abilities in automatic deduction. The paramGeo3D library was designed for local work, and, although it can be used for automatic proof, it shows a strong lack of automatism when finding loci, and does not compute other important objecs such as envelopes. Furthermore, the symbolic approach carried on by these libraries can be outperformed by the application of recent results concerning the automatic management of degenerated conditions and refinement of results.

3. The library

3.1. Rationale of the library

Trying to answer the above critics, we have developed an open source library for automatic discovery in 3D dynamic geometry. We are not aware of any other attempt of linking such a library with a 3D DG system (apart from the ones above discussed). We will not focus on connecting the library with some given environment, since there are few candidates and we think

that it exists a kind of universal grammar for describing elementary 3D constructions. The arguments in preceding sections show that interfacing DG software with the library is more a development issue than a scientific one. So, a short description of the library is given in this section. Ongoing work such as a 3D version of GeoGebra [36] or the completion of porting the InterGeo common file format to 3D will open new lines of development, given the previsible spread of this new version of GeoGebra and the intercommunication possibilities opened by a common 3D DG specification.

The AutDed3D library is a Sage [23] piece of code implementing a parametrical approach for describing elementary 3D geometric constructions. It uses a Gröbner–based elimination process to compute important objects in the DG paradigm such as loci and envelopes and to perform automatic proof. Regarding the constructed objects, the aim of AutDed3D is to compute automatically (i.e., without any human intervention) their exact algebraic description, so assisting an eventual 3D DG system to get a deeper knowledge of these objects. Regarding the proof, the library can return the truth of some elementary statements. In this paper, our main interest concerns loci and envelopes, so the general proving part is included for the sake of completeness, and it will thoroughly studied in the future.

AutDed3D heavily relies on using specialized software such as Singular. In fact, the Sage ability for integrating software has been one of the reasons for selecting it as platform development. Since Sage and all the other software needed by the library are open source, there are no restrictions for non commercial uses of the library. Furthermore, the pythonic character of Sage, the possibility of using Sage as a Python library and the ongoing work about a Simple Compute Server [39], open ways for remotely using the library (i.e., via connecting it with DG environments).

3.1.1. Using the library

There are two ways of accessing the library, both via the main Sage server. The library textfile can be downloaded from `http://sagenb.org`, locating the published worksheet `AutDed3D` [40]. For testing purposes, one should copy it. Once done, one can follow the standard procedure using either the Sage notebook, or the terminal. The timings for all the examples described in this paper are in the worksheet, showing that the algorithms are adequate for connecting the library with a 3D DG system. Another way for accessing the library consists of using the experimental Simple Compute Server `http://aleph.sagemath.org`. This server accepts anonymous requests and

its technology will be used to automate remote communication between 3D DG systems and the AutDed3D library. Detailed instructions about this way of accessing the library can be found in [41].

3.2. Data structure

Each library object is described by a dictionary detailing its characteristics. Among them, we have its type (basically, points, lines and surfaces), its parents, its coordinates or equation and a string containing the command used for constructing it. For instance, there are two types of points, `FreePoint` and `BoundedPoint`. A free point, in turn, can be constructed twofold: by using the function `FreePoint(<name>,#,#,#)` which defines the point *name* with given numerical coordinates, or by `SymbPoint(<name>)`, which defines a point with symbolic coordinates (u_1, u_{i+1}, u_{i+2}). A point constructed by the user with coordinates, say $(1, 2, 3)$, is described by

```
{'hist': ['FreePoint', 'a', 1, 2, 3], 'parents': {}, 'eq': {},
 'coords':(1, 2, 3), 'type': 'FreePoint'}
```

while a sphere centered at a symbolic point b with coordinates (u_1, u_2, u_3) and radius 1 is

```
{'radius2': 1, 'center': 'b', 'type': 'Surface',
 'eq': {(Z - u3)^2 + (Y - u2)^2 + (X - u1)^2 - 1},
 'parents': {'b'}, 'hist': ['Sphere', 'sph', 'b', 1]}
```

All the construction objects are encoded similarly as dictionaries, and the whole construction is itself a dictionary called `All`. Any construction element or element item can then be easily accessed via the standard key:value structure.

3.3. Basic and auxiliary functions

The library has a list of functions for defining the usual basic 3D elements. Free points (points without constraints can be defined via `FreePoint` or `SymbPoint`), whereas points with constraints use `BoundedPoint`. Other elementary functions are `Line`, `Plane`, `Sphere`, `Surface`, `MidPoint`, and the standard ones involving parallelism and perpendicularity (`ParallelLine`, `ParallelPlane`, ...). Points on objects are defined through `PointOnObject`, simply valuating the object equations by the point coordinates. For instance, assuming there is a sphere as above, the function `PointOnObject('p','sph')` introduces in the construction the value

```
{'hist': ['PointOnObject', 'p', 'sph'], 'parents': {'sph'},
'eq': {(u3 - x3)^2 + (u2 - x2)^2 + (u1 - x1)^2 - 1},
'coords': (x1, x2, x3), 'type': 'BoundedPoint'}
```

Multiple constraining of points is allowed by adding new expressions to the `eq` value of points. Concerning intersections, a single function takes care of them. `IntersectionObjectObject` creates a point (or updates an existing one) constraining it with the equations of both parents. Since a parametric treatment of geometric constructions is the reason of this library, we are not forced to compute, at defining times, all present possibilities of intersections. A lazy evaluation is behind this strategy. As an illustration, consider intersecting two spheres: a point lying on both of them is algebraically described by two equations which can have as solutions a circle, a point or no real solution. Our interest is a general description of the common points, and it is accomplished just by the system of equations.

There is also an extensive list of auxiliary functions (for moving across the construction graph, housekeeping tasks, ...). They are not designed for normal user access, so not being described here.

3.4. Loci functions

In [22] the authors illustrate their proposal by computing an ellipsoid as a locus using a natural extension of the gardener's method. In our setting, and using their element names and values, the construction is

```
FreePoint('Pt1',0,0,0); FreePoint('Pt2',1,0,0)
FreePoint('Pt3',0,2,0); Line('L13','Pt1','Pt3')
PointOnObject('Pt5','L13'); Sphere('Sph8','Pt1','Pt1','Pt5')
Sphere('Sph9','Pt2','Pt3','Pt5')
IntersectionObjectObject('Cr10','Sph8','Sph9')
```

They state that "*paramGeo3D has automatically obtained as locus (without the user having to type anything)*" when, asking for `Cr10`, the system returns

$$[4x2^2 + 43^2 + 48k1^2 + 9 - 48k1, 2x1 + 3 - 8k1],$$

that is, the intersection circle of the two spheres is the intersection of a cylinder and a plane, both depending on a parameter. Two more Maple commands are needed in order to get the final ellipsoid polynomial yet

$$12x_1^2 - 12x_1 + 16x_2^2 + 16x_3^2.$$

This approach, although finally successfull, can hardly be incorporated for non expert use, since it involves a deep knowledge of what is going on. If the goal is automating the process of obtaining the ellipsoid using a 3D gardener's method, it should be enough the traditional method in DG. That is, once the spheres intersection is created, selecting a `Locus` tool with arguments this intersection and the moving point, the system should just return the ellipsoid.

As recalled in Section 2.1, there is a simple procedure for computing the algebraic description of a locus. In the library the `Locus` function deals with loci tasks. Regarding the ellipsoid, just executing `Locus('l','Cr10','Pt5')` a new object is added to the construction, being

```
{'type': 'Surface', 'tracer': 'Cr10', 'mover': 'Pt5',
 'hist': ['Locus', 'L', 'Cr10', 'Pt5'],
 'parents': {'Pt5', 'Cr10'}, 'implicit': False,
 'eq': {12*X^2 + 16*Z^2 + 16*Y^2 - 12*X - 9}}
```

Note that this function fully automates the task, being able to be directly incorporated to any standard 3D DG system.

3.4.1. A difference between standard and parametric approaches

In Section 2 we used a conchoid to illustrate a difference between the standard approach taken in DG and the parametrical one proposed here. Here we consider another case, showing a wrong Calques3D locus. Furthermore, this construction proves that a specific `Locus` command is needed, since some elementary objects can only be obtained this way.

A Bohemian dome can be constructed as follows. Given a circle C and a plane E perpendicular to the plane containing C, move a circle K through space such that its center lies on C and remains parallel to E. The wrong answer returned by Calques3D is shown in Fig. 3. The authors of paramGeo3D chose to define specific intersections for elementary algebraic objects, not including the case of planes and spheres, as it is needed here. Even if a `Circle` function was available, computing the envelope of a family of circles would be necessary (a function also absent in paramGeo3D). On its side, the library `Locus` function directly returns as equation the quartic

$$x^4 + 2x^2y^2 + 2x^2z^2 - 4x^2 + y^4 - 2y^2z^2 + z^4,$$

shown in Fig. 4 (for the sake of brevity, we do not include here the construction description, referring to the Sage worksheet for studying it).

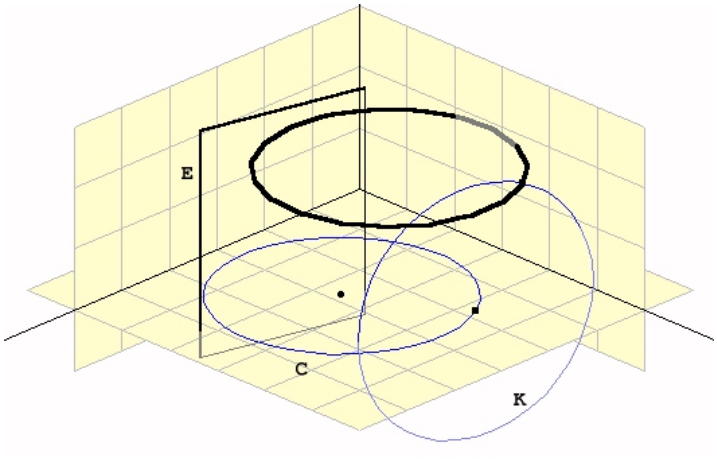

Figure 3: A wrong Bohemian dome (bold) as drawn by Calques3D.

3.4.2. Implicit and general loci

The constructive philosophy of dynamic geometry prevents studying cases where some element is unknown, thus disallowing user discovery. Consider Simson's theorem stating the equivalence between the alignment of the projections from a point to the sides of a triangle and the membership of the point to the circumcircle of the triangle. Whereas it is very easy to check the alignment of projections in a DG system, if a point is placed on the circumcircle, the other way is harder (if one does not know the theorem!). One could ask about the 3D analogous case: which is the locus of space points such that their orthogonal projections on the (extended) faces of a tetrahedron are coplanar? Since a general point in this locus is not easily constructible, 3D systems do not offer any answer. This case illustrates situations where the constraints are imposed after the point is constructed. These loci have been called *a posteriori* or *implicit* ones. Given the lazy strategy in our parametric approach, there is no substantial difference with the standard locus approach. A generic locus point is defined via the `SymbPoint` function and it is constrained a posteriori. Since there is not a mover point, that is, the locus point is not parametrically determined by its ancestors, but by its descendants, we consider all variables appearing on them as parameters. The usual elimination process can then be applied to get the locus. In the case of the 3D version version of Simson's theorem, the `Locus` function returns, for

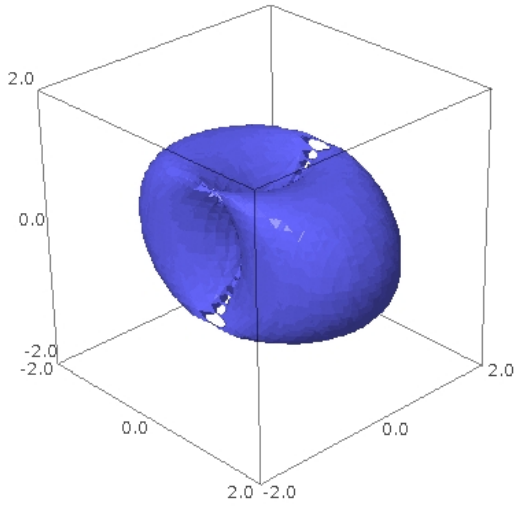

Figure 4: A true Bohemian dome.

a tetrahedron with vertices the origin and the three unit points, the cubic

$$x^2y + x^2z + xy^2 + xz^2 + y^2z + yz^2 - xy - xz - yz,$$

shown in Fig. 5.

In this implicit locus, we are reasoning about a specific tetrahedron (that is, having its free points numeric coordinates), therefore losing generality. The `Locus` function also accepts general constructions (where their basic points are now defined via `SymbPoint`). Thus, the loci obtained this way are not objects in the affine space anymore, being statements about geometric elements in a n–dimensional space. For instance, replacing the fourth vertex $(0, 0, 1)$ of the tetrahedron by a symbolic point with coordinates (u_1, u_2, u_3), the locus will be expressed as

$x^2zu_2^2u_3^4 + x^2zu_3^6 - 2xyzu_1u_2u_3^4 - 2xz^2u_1u_3^5 + y^2zu_1^2u_3^4 + y^2zu_3^6 - 2yz^2u_2u_3^5 + z^3u_1^2u_3^4 + z^3u_2^2u_3^4 + x^2yu_3^5 - x^2zu_2u_3^4 + xy^2u_3^5 + xz^2u_3^5 - xzu_2^2u_3^4 - xzu_3^6 - y^2zu_1u_3^4 + yz^2u_3^5 - yzu_1^2u_3^4 - yzu_3^6 - z^3u_1u_3^4 - z^3u_2u_3^4 + z^2u_1^2u_2u_3^3 + z^2u_1u_2^2u_3^3 + z^2u_1u_3^5 + z^2u_2u_3^5 - xyu_3^5 + xzu_2u_3^4 + yzu_1u_3^4 - z^2u_1u_2u_3^3.$

Nevertheless, given the cost of the involved algorithms, results about general constructions in the automated approach discussed here can be sometimes out of scope for current computational devices.

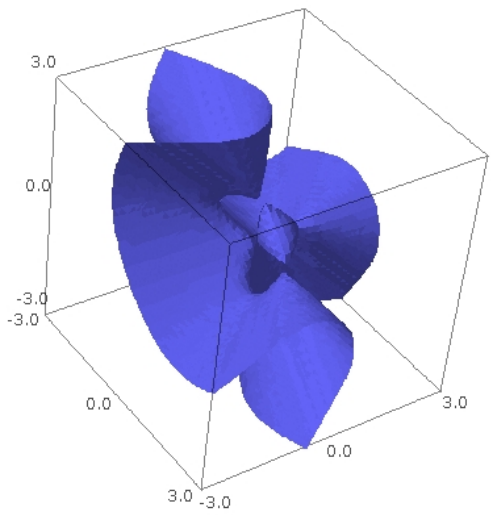

Figure 5: The cubic in a 3D version of Simson's theorem.

3.5. Envelopes

While most 2D DG systems can suggest envelopes of families of curves by tracing an element family, no 3D system exists yet offering a similar ability. In [5] we gave an algorithm for automatically computing envelopes of plane curves, and we showed its efficient integration into a 2D DG environment. Here we extend this algorithm to deal with envelopes of families of general surfaces. Its implementation in our parametrical setting is illustrated through different examples. Regarding paramGeo3D and 3D–LD, the former does not include any way of calculating them, while the second can only compute a restricted type of canal surfaces (those where the spheres move along a straight line).

Finding envelopes can be seen as an elimination problem. Rougly speaking, the envelope of a family of surfaces [24] is a surface that is tangent at each of its points to a surface of the family. Suppose that the family of surfaces is given by the system

$$\left.\begin{aligned} f(x,y,z,\alpha_1,\ldots,\alpha_n) &= 0\\ g_1(\alpha_1,\ldots,\alpha_n) &= 0\\ &\vdots\\ g_{n-1}(\alpha_1,\ldots,\alpha_n) &= 0 \end{aligned}\right\}$$

Then the envelope can be computed by adjoining this system to the equation

$$\begin{vmatrix} \frac{\partial f}{\partial \alpha_1} & \frac{\partial f}{\partial \alpha_2} & \cdots & \frac{\partial f}{\partial \alpha_n} \\ \frac{\partial g_1}{\partial \alpha_1} & \frac{\partial g_1}{\partial \alpha_2} & \cdots & \frac{\partial g_1}{\partial \alpha_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial g_{n-1}}{\partial \alpha_1} & \frac{\partial g_{n-1}}{\partial \alpha_2} & \cdots & \frac{\partial g_{n-1}}{\partial \alpha_n} \end{vmatrix} = 0,$$

after eliminating the parameters $\alpha_1, \ldots, \alpha_n$. The library function `Envelope` incorporates this procedure, allowing the computation of an extensive list of envelopes. The simple case of the envelope of moving spheres centered at the plane xy and with radius 1 is described by

```
FreePoint('o',0,0,0);FreePoint('i',1,0,0);FreePoint('j',0,1,0)
Plane('xy','o','i','j'); PointOnObject('p','xy')
Sphere('s','p',1); Envelope('env','s','p')
```

being the envelope the parallel planes $z = 1$ and $z = -1$:

```
{'eq': {Z^2 - 1}, 'hist': ['Envelope', 'env', 's', 'p'],
 'parents': {'p', 's'}, 'mover': 'p', 'type': 'Surface',
 'tracer': 's'}
```

In a similar way, we compute the 1-offset of a paraboloid. Although the paraboloid is defined, in the Sage worksheet, in a constructive way, here we introduce it in the construction using the `Surface` function, which allows to directly define algebraic surfaces. With the input

```
Surface('parab',[z-x^2-y^2-1]); PointOnObject('P','parab')
Sphere('s','P',1); Envelope('offset','s','P')
```

the offset is returned as the sextic shown in Fig. 6.

Another illustration of the computation of envelopes deals with the case where the elements of the family are computed as loci. Since in our approach a locus is a geometric element obtained by eliminating all parameters involved in its definition, there is no way to get the expression of a parametric family of loci (even the use of symbolic points is excluded for this case). Consider, for instance, the ellipsoid in 3.4. Replacing one of its foci by a moving point along the x axis, the computed locus would be the whole space, since the constraints define a three dimensional part of the space. In order to skip

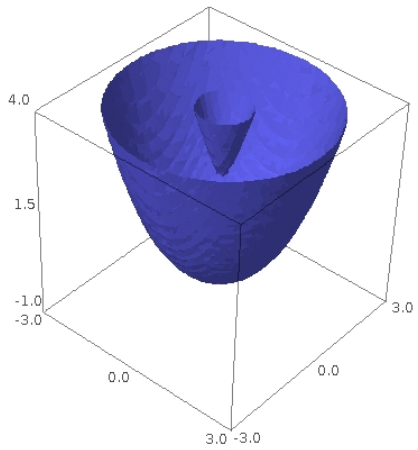

Figure 6: The 1–offset of the paraboloid $z = x^2 + y^2 + 1$, with equation $16x^6 + 48x^4y^2 + 16x^4z^2 + 48x^2y^4 + 32x^2y^2z^2 + 9x^4 + 54x^2 + 16y^6 + 16y^4z^2 - 72x^4z - 144x^2y^2z - 32x^2z^3 - 72y^4z - 32y^2z^3 + 18x^2y^2 + 96x^2z^2 + 9y^4 + 96y^2z^2 + 54y^2 + 16z^4 - 90x^2z - 90y^2z - 104z^3 + 225z^2 - 162z = 0$.

this restriction, we use a function similar to `Locus`, but allowing to specify a bounded point, which will be used as parameter for the family of surfaces. The strategy followed here is to exclude the coordinates of this point when eliminating parameters in the finding of the locus. So, redefining the point $Pt2$ in the ellipsoid construction to be a moving point on the x axis, the call `Locus2('ellips','Cr10','Pt2','Pt5')` returns as equations for this object

$$\left.\begin{array}{rcl} x_2 & = & 0 \\ x_3 & = & 0 \\ 4x^2x_1^2 - 4xx_1^3 - 16x^2 + 16xx_1 - 16y^2 - 16z^2 + x_1^4 - 8x_1^2 + 16 & = & 0 \end{array}\right\},$$

being x_1, x_2, x_3 the coordinates of the parameter point $Pt2$. So, we have a mono–parametric family of objects defined as special loci, and its envelope can then be computed (note that forcing $Pt2$ to lie on a 2–dimensional object we could obtain a bi–parametric family). Finally, the envelope is returned as the surface defined by the equation

$$(x^2 + y^2 + z^2)(x^4 - 8x^2 - 16y^2 - 16z^2 + 16) = 0.$$

A plot of this surface is shown in Fig. 7.

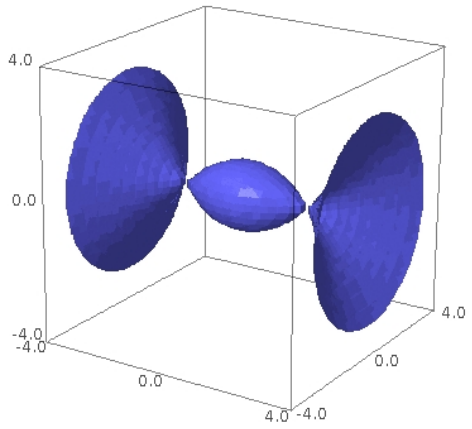

Figure 7: A surface that envelopes a family of ellipsoids.

Two things must be noted concerning this envelope. The point $(0, 0, 0)$ returned as part of the surface is not tangent to an element of the family and it should be dropped out (it appears when an ellipsoid of the family degenerates. Currently, we do not have any test to detect such cases). Regarding the rest of the surface, it is clear that only the central part envelopes the ellipsoids. There is a notable difference between our aim and the given algebraic specification. While we talked about a *family of ellipsoids*, its algebraic description, i.e. the 0–set of

$$4x^2x_1^2 - 4xx_1^3 - 16x^2 + 16xx_1 - 16y^2 - 16z^2 + x_1^4 - 8x_1^2 + 16,$$

taking x_1 any real value, refers to more objects than the ellipsoids. This family describes ellipsoids if $-2 < x_1 < 2$, degenerated quadrics if $x_1 = \pm 2$ and hyperboloids of two sheets in other case. So, the returned surface also envelopes the hyperboloids.

3.6. Automatic proof

The library includes, for the sake of completeness, some procedures related to automatic proof. Since current 3D DG systems offer limited abilities (if any!) for this subject, we restrict here to test the correction of simple statements. There are two functions for performing automatic proof. The first one, `prove_sat`, applies the protocol in [11] returning a boolean that informs if the statement is generally true or false. The second one, `prove_deg`,

derived from [19], can also find degenerated conditions in case of truth (although in a non complete way).

Using these functions is straightforward. Once a geometric constructions is specified, they are called with the statement to prove as argument. As an illustration, we study the concurrency of the segments joining the midpoints of the three pairs of opposed edges in an arbitrary tetrahedron. Defining the tetrahedron and the segments by

```
FreePoint('a',0,0,0); FreePoint('b',1,0,0),
SymbPoint('c'); SymbPoint('d'); MidPoint('mab','a','b')
MidPoint('mac','a','c'); MidPoint('mad','a','d')
MidPoint('mbc','b','c'); MidPoint('mbd','b','d')
MidPoint('mcd','c','d'); Line('ab-cd','mab','mcd')
Line('ac-bd','mac','mbd'); Line('ad-bc','mad','mbc')
```

(points a, b can be set as origin and unit without loss of generality) and we define a common point of a pair of segments

```
IntersectionObjectObject('12','ab-cd','ac-bd')
```

Testing the membership of this point to the remaining segment with

```
prove_sat(are_aligned('12','mad','mbc'))
```

True is obtained since the points are aligned. Using the other function, `prove_deg`, the answer is

```
[u3*u5-u2*u6, u3*u4-u1*u6+u6, u2*u4-u1*u5+u5]
```

meaning that the statement is True under the above conditions of degeneration. Note that `FreePoint` could be used, instead of `SymbPoint`, when defining points c and d as follows:

```
FreePoint('c',u1,u2,0); FreePoint('d',u3,u4,u5)
```

since c can be restricted to this plane without loosing generality. In this case, the degenerated conditions are more simple

```
[u2*u5, u1*u5-u5, u2*u3-u1*u4+u4]
```

Besides the collinearity condition, the library includes a list of geometric statements (`parallel`, `perpendicular`, ...). This list, although reduced, can be easily extended. Furthermore, if a statement cannot be easily expressed with them, it can be also passed as argument. In order to check that the concurrency point is the midpoint of each segment, we can use a polynomial condition involving point coordinates (which are obtained with the functions `xc`, `yc`, `zc`)as follows:

```
prove_sat([(xc('12')-xc('mab'))^2+(yc('12')-yc('mab'))^2
           +(zc('12')-zc('mab'))^2-(xc('12')-xc('mcd'))^2
           -(yc('12')-yc('mcd'))^2-(zc('12')-zc('mcd'))^2,
           ['12','mab','mcd']])
```

where it must be noted that the function argument is a list containing the polynomial condition plus the involved geometric elements (since knowing these elements is necessary for moving across the construction graph and hence correctly defining the polynomial ring).

4. Refining the algebraic descriptions of loci and envelopes

The symbolic determination of the equations of loci and envelopes and its implementation in DG environments has been an important step through the automatic management of knowledge in geometric systems. Nevertheless, the elimination–based approaches, such as the described here, suffer drawbacks coming from two sources: there is no a general theory about geometric degeneracy, and the used elimination can introduce extraneous parts in the obtained equations. In this Section we use some recent results related to the theory of parametrical polynomial systems that solve both problems for most geometric constructions.

As an illustration of the rol of degeneracy we recall the limaçon of Pascal in Section 2. The locus returned there includes a circle appearing since a line defined through two points collapses when the points coincide. An identical situation occurs when generalizing this limaçon to 3D: Given a sphere with center at the origin and passing through $A(0,0,2)$, let us consider a point B on it, a second sphere centered at B and with radius 1. The locus of points C lying on the line AB and on the second sphere, computed through the `Locus` function, consists of the sphere

$$x^2 + y^2 + z^2 - 4z + 3 = 0$$

and the quartic

$$x^4 + 2x^2y^2 + 2x^2z^2 - 9x^2 + y^4 + 2y^2z^2 - 9y^2 + z^4 - 9z^2 + 4z + 12 = 0.$$

Here again, the spheric part of the locus is returned since line AB is undefined when $A = B$. One could explicitly remove this case by stating somehow that $A \neq B$. If the system accepts this kind of statements, a dummy new variable should be included in the elimination, and the sphere would not appear. Nevertheless, this procedure also relies in an efficient discovery by the user of all cases of degeneracy, so being an unsafe process.

Another source of imprecision of results comes from the own elimination process. Since this process returns the Zariski closure of the sought objects, it can happen that the adherence contains superfluous parts. A pedal surface illustrates this case. Since the pedal surface of a given surface with respect to a point is the locus of points intersecting the normal lines to tangent planes of the surface with these planes, a simple function for computing such pedals is included in the library as `PedalSurface`. The construction

```
FreePoint('o',0,0,0);Surface('e',[x^2+y^2+20*z^2-1])
PedalSurface('ped','e','o')
```

computes the pedal surface of the ellipsoid $x^2+y^2+20z^2-1 = 0$ with respect to the origin. The returned equation is

$$20x^4 + 40x^2y^2 + 40x^2z^2 - 20x^2 + 20y^4 + 40y^2z^2 - 20y^2 + 20z^4 - z^2 = 0.$$

Since it contains the origin, it cannot be a correct description of the sought pedal (Fig. 8). Trying to get the pedal of this pedal, with the above equation, with respect to the origin, we would obtain the whole space. It is necessary to exclude the point $(0, 0, 0)$ from the equation for getting a sound second pedal. This fact highlights the need of substituting the current description of objects in parametrical DG in terms of equations (i.e., varieties) by constructible sets (i.e., union of differences of varieties).

In this Section, we will use recents algorithms coming from the field of parametric polynomial systems solving for a new approach to automatically detect degeneracies in loci and envelopes, and also to remove superfluous parts on them.

The search for loci and envelopes in our parametrical approach always involves solving parameterized polynomial systems. We search for solutions

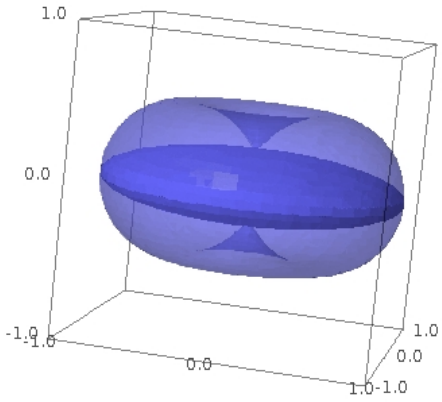

Figure 8: An ellipsoid and its pedal (light) wrt the origin.

of the system in terms of parameter values, and if a solution exists, we are interested in its structure (dimension, number, ...). Looking at the coordinates of bounded points as parameters and those of the locus or envelope as variables, a parametric polynomial system over $\mathbb{Q}$ is a finite set of polynomials $p_1, \ldots, p_r \in \mathbb{Q}[\bar{a}, \bar{x}]$ (that is, the geometric constraints) in the variables $\bar{x} = x_1, \ldots, x_n$ and parameters $\bar{a} = a_1, \ldots, a_m$. The goal is studying the solutions of the algebraic system $\{p_1(a, \bar{x}), \ldots, p_r(a, \bar{x})\} \subset \mathbb{Q}[\bar{x}]$ which are obtained by specializing the parameters to concrete values $\bar{a} \in \mathbb{C}^m$.

The main computational approach for studying these systems is based on the method of Gröbner bases. Several articles about applicating it to the parametrical case are [27, 25, 17, 18, 16], just to mention a few. In this paper we employ the GröbnerCover (GC) algorithm proposed in [18], since it can be efficiently used for solving both mentioned tasks for most geometric constructions.

Recalling the 3D limaçon, the polynomial constraints are (where B is a bounded point with coordinates (x_1, x_2, x_3) and the locus point is $C(x, y, z)$)

- $x_1^2 + x_2^2 + x_3^2 - 4$, B lies on the sphere $((0, 0, 0), 2)$
- $(x_3 - z)^2 + (x_2 - y)^2 + (x_1 - x)^2 - 1$, C lies on the sphere $(B, 1)$
- $x_1 y - x_2 x, (z - 2)x_1 - (x_3 - 2)x, (z - 2)x_2 - (x_3 - 2)y$, A, B, C aligned

Solving the above system GC outputs 5 segments as follows:

- Segment 1
  - segment: $\mathbb{C}^3 \setminus (\mathbb{V}(x^2+y^2+z^2-4z+3) \cup \mathbb{V}(x^4+2x^2y^2+2x^2z^2-9x^2+y^4+2y^2z^2-9y^2+z^4-9z^2+4z+12))$
  - basis: $\{1\}$
- Segment 2
  - segment: $(\mathbb{V}(x^2+y^2+z^2-4z+3) \setminus (\mathbb{V}(2z-3, 4x^2+4y^2-3)) \cup (\mathbb{V}(x^4+2x^2y^2+2x^2z^2-9x^2+y^4+2y^2z^2-9y^2+z^4-9z^2+4z+12) \setminus (\mathbb{V}(2z-3, 4x^2+4y^2-3) \cup \mathbb{V}(z-2, x^2+y^2))$
  - basis: $\{(2x^2+2y^2+2z^2-4z)x_3+(-x^2z-2x^2-y^2z-2y^2-z^3+2z^2-3z+6), (2x^2+2y^2+2z^2-4z)x_2+(-x^2y-y^3-yz^2+4yz-3y), (2x^2+2y^2+2z^2-4z)x_1+(-x^3-xy^2-xz^2+4xz-3x)\}$
- Segment 3
  - segment: $\mathbb{V}(z-2, x^2+y^2) \setminus \mathbb{V}(z-2, y, x)$
  - basis: $\{1\}$
- Segment 4
  - segment: $\mathbb{V}(z-2, y, x) \setminus \mathbb{V}(1)$
  - basis: $\{4x_3-7, 16x_1^2+16x_2^2-15\}$
- Segment 5
  - segment: $\mathbb{V}(2z-3, 4x^2+4y^2-3) \setminus \mathbb{V}(1)$
  - basis: $\{x_2+2yx_3-4y, x_1+2xx_3-4x, x_3^2-3x_3+2\}$

Segment 1 expresses that out of the sphere and the quartic (the factors of the locus previously obtained), there is no solution since the basis is $\{1\}$. Segment 2 guarantees that the sphere minus the points simultaneously lying on the circle given by the plane $2z-3=0$ and the cylinder $4x^2+4y^2-3=0$, and the quartic minus the same circle plus the point $(0,0,2)$ satisfy the locus constraints (that is, a solution in terms of x_1, x_2, x_3 can be computed via the given basis). Segment 3 must be rejected, since it introduces a difference between imaginary points, so not being relevant here. Finally, segments 4

and 5 state that the points on the above circle and the point $(0, 0, 2)$ can also be solved in terms of the parameters.

The structure of the space of solutions returned by GC is a key step when solving the problems studied here. Nevertheless, the compactness of the GC output produces that sometimes the solutions in a segment cannot be described by a Gröbner basis. In such cases, the algorithm uses regular functions [30]. We do not have currently a strategy for dealing with such GC outputs, so we will study using other algorithms in a future work.

4.1. Removing degenerated components

The problem tackled here deals with removing degenerated components in loci and envelopes. Thus, we assume that their algebraic description is known, that is, that we have a variety describing the general locus. Since each component in the variety comes from a value assignment of the parameters, the key idea is examining the corresponding dimensions of the space of parameters (for this valuation) and the space of variables (for the studied component). Our heuristic is that a component is degenerated if its dimension is greater than the corresponding one for the space of parameters, provided that the parameter set is 0–dimensional. This heuristic has been incorporated in the library function `deg_comp`, which returns a list of degenerated components for a previously computed locus or envelope.

Recalling the 3D limaçon, its locus contains two components, the degenerated sphere and a quartic that constitutes the true locus. In order to remove the sphere, we proceed as follows. We locate the GC segment where this locus factor appears as a minuend in the constructible set description of the segment. Once done, the dimension of the space of parameters is calculated. This space is determined by eliminating all variables (except those of relevant parameters) in the ideal formed by the segment basis plus the locus factor. Finally, this dimension is compared with the factor dimension and the above heuristic is applied. Since the sphere is 2–dimensional and the space of parameters here is the point $(0, 0, 2)$, the sphere is declared as a degeneracy, and hence it should be excluded from the locus. The function call `deg_comp('loc')` prints

```
The component  x4^2 + x5^2 + x6^2 - 4*x6 + 3  is degenerated
 for  [x3 - 2, x2, x1] !
```

and returns a list containing the component (note that currently the found components are expressed using the coordinates of the locus point, instead

the generic variable names x, y, z). The quartic is of course not degenerated since its dimension and the one corresponding to its parameters space are both equal to two.

If the GC output uses regular functions, as it happens when finding the following pedal surface (for the sake of brevity, we use again the option of defining an ellipsoid directly as a surface)

```
Surface('ellips',[12*x^2 + 16*y^2 + 16*z^2 - 12*x - 9])
FreePoint('P',1/2,0,0)
PedalSurface('ped','ellips','P')
```

the strategy implemented in the `deg_comp` function changes the roles of variables and parameters, trying to see if no regular functions appear this time. This is the case for the above pedal, getting as answer

```
Regular functions. Variables and parameters change!
[]
```

meaning that the strategy had to be followed and that no degeneracy was found. This pedal also illustrates a heuristic to be used when incorporating these technique to a 3D DG system. If a locus is irreducible, there is no need for testing degeneracy, since the locus is not empty (otherwise it would be detected) and so it cannot contain a degenerated component.

4.2. Finding spurious parts

The structure of the space of variables in the GC algorithm can also be used to detect wrong parts of loci and envelopes that are introduced through the elimination process. Recalling the pedal surface shown in Fig. 8, we got it as the variety

$$\mathbb{V}(20x^4 + 40x^2y^2 + 40x^2z^2 - 20x^2 + 20y^4 + 40y^2z^2 - 20y^2 + 20z^4 - z^2),$$

and we noted that it contains the origin as extra point. If the locus is to be determined by an equation, there is no way to reject this point. Nevertheless, turning out to a representation of the object as a constructible set, it can be described by

$$\mathbb{V}(20x^4+40x^2y^2+40x^2z^2-20x^2+20y^4+40y^2z^2-20y^2+20z^4-z^2)\setminus\mathbb{V}(x,y,z).$$

The variable space is partitioned in four segments by the GC algorithm, having $\{1\}$ as basis all segments except the following:

- Segment 2
  - segment: $\mathbb{V}(20x^4+40x^2y^2+40x^2z^2-20x^2+20y^4+40y^2z^2-20y^2+20z^4-z^2) \setminus \mathbb{V}(z, x^2+y^2)$
  - basis: $\{(20x^2+20y^2+20z^2)x_3-z, (x^2+y^2+z^2)x_2-y, (x^2+y^2+z^2)x_1-x\}$,

meaning that all the points in the quartic with the exception of the origin are exactly those which satisfy the construction constraints. Thus, this one is the exact locus of the pedal. The `remove_deg_comp` function takes care of this issue, returning the sought pedal as a constructible set (where x_4, x_5, x_6 must be understood as the coordinates of a generic pedal point)

```
[20*x5^4 + 40*x5^2*x6^2 + 20*x6^4 + 40*x5^2*x4^2 + 40*x6^2*x4^2
 + 20*x4^4 - 20*x5^2 - x6^2 - 20*x4^2] \ [x6, x5^2 + x4^2]
```

It should be noted that this approach for removing spurious parts would not work if regular functions in the GC output have been found when solving the structure of the space of variables. Even if the structure of the parameters space is known, we ignore how to exploit this information for concluding anything about spurious parts in the variables space.

A final remark deals with exploiting the kwnoledge about the structure of the variables space for solving difficult problems in CAD. For instance, detecting self intersections in offsets is a non trivial problem. Let us consider the offset of a paraboloid shown in Fig. 6. There is a part of the offset that appears due to self intersections produced by the moving spheres. Fig. 9 shows the analogous plane case and the self intersection part of the paraboloid offset.

It must be noted that the self intersection part of the paraboloid offset is limited by a point (top) and a circle (bottom), which are exactly found in the GC segment partition: the segment $\mathbb{V}(4z-9, x, y) \setminus \mathbb{V}(1)$, that is, the point $(0, 0, 9/4)$, and the segment

$$\mathbb{V}(64z^3-144z^2+108z-135, 12x^2+12y^2+16z^2-36z+9) \setminus \mathbb{V}(1),$$

which is a circle. Following [14], each point between both parts can be dropped out when removing self intersections. Other future application of these ideas will deal with trimming bisectors from an implicit description of the involved surfaces.

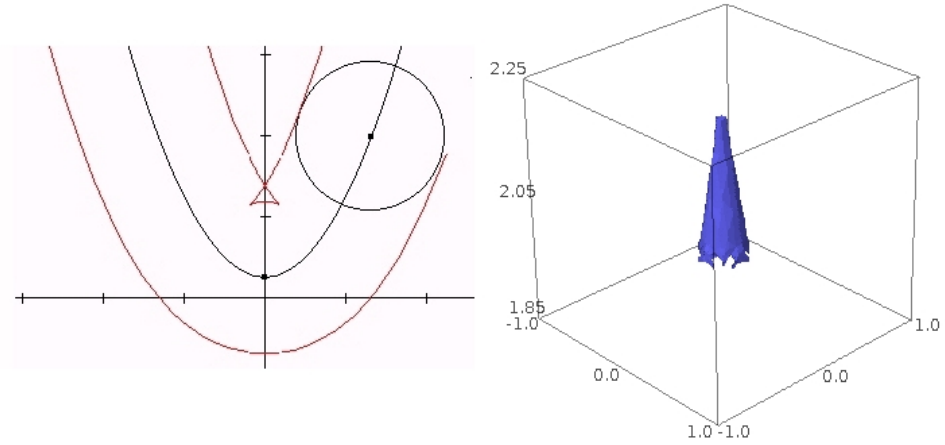

Figure 9: Self intersections in the offsets of a parabola and the given paraboloid.

5. Conclusion

The main interest of this work is to provide an open source library allowing the automatic management of parametric descriptions of 3D geometric constructions. Besides the usual basic functions, it provides efficient implementations of functions for automatic proof and automatic discovery of loci and envelopes. Since loci (and envelopes, although in a lower degree) are essential objects in any DG environment, no parametric library can be considered complete if lacking such ability. Furthermore, the library does not depend on any expensive CAS, it is not bound to any specific 3D DG system, and it can be easily linked to forthcoming systems and protocols, in a local or remote access basis.

We would like to underline that our solution for current drawbacks relating imprecisions coming from elimination or degeneracies is, as far as we know, completely new for the 3D DG case. The extensive list of considered examples should make clear that the standard use of equations for objects must be replaced by a richer description as constructible sets.

There are two main lines for future work. On the one hand, deploying a web server incorporating the library for allowing remote access of users of any compliant 3D DG system, and disseminating the developed technology for it. On the other hand, we plan extending the library for CAD tasks related to offsets, blends, equidistance surfaces, ...

Acknowledgment

The author was partially supported by research grant MTM2008-04699-C03-03/MTM from the Spanish MICINN.

References

[1] F. Botana, J.L. Valcarce, A dynamic–symbolic interface for geometric theorem discovery, Computers and Education 38 (1–3) (2002) 21–35.

[2] F. Botana, Interactive versus symbolic approaches to plane loci generation in dynamic geometry environments, in: P.M.A. et al. (Eds.), Computational Science, ICCS 2002, Springer-Verlag, LNCS 2330, Berlin-Heidelberg, 2002, pp. 211–218.

[3] F. Botana, J.L. Valcarce, A software tool for the investigation of plane loci, Mathematics and Computers in Simulation 61 (2) (2003) 141–154.

[4] F. Botana, Automatic determination of algebraic surfaces as loci of points, in: P.M.A. et al. (Eds.), Computational Science, ICCS 2003, Springer-Verlag, LNCS 2657, Berlin-Heidelberg, 2003, pp. 879–886.

[5] F. Botana, J.L. Valcarce, Automatic determination of envelopes and other derived curves within a graphic environment, Mathematics and Computers in Simulation 67 (1–2) (2004) 3–13.

[6] F. Botana, A symbolic companion for interactive geometric systems, in: J.H. Davenport et al., Intelligent Computer Mathematics, Calculemus–MKM 2011, Springer-Verlag, LNCS 6824, Berlin-Heidelberg, 2011, pp. 285–286.

[7] F. Botana, M.A. Abánades, Automated deduction in dynamic geometry using Sage, in: P. Quaresma, R.–J. Back (Eds.), CTP Components for Educational Software, THedu11, CISUC Technical Report 2011/001, Coimbra, 2011, pp. 21–25.

[8] B. Buchberger, Applications of Gröbner bases in non–linear computational geometry, in: J.R. Rice (Ed.), Mathematical Aspects of Scientific Software, Springer–Verlag, 1988, pp. 59–87.

[9] S.C. Chou, Mechanical Geometry Theorem Proving. Reidel, 1988.

[10] CoCoATeam, CoCoA: a system for doing Computations in Commutative Algebra. Available at http://cocoa.dima.unige.it.

[11] P. Conti, C. Traverso, Algebraic and semialgebriac proofs: methods and paradoxes, in: J. Richter-Gebert, D. Wang (Eds.), Automatic Deduction In Geometry, ADG 2000, Springer-Verlag, LNAI 2061, Berlin-Heidelberg, 2001, pp. 83–103.

[12] W. Decker, G.M. Greuel, G. Pfister, H. Schönemann, SINGULAR 3-1-1 — A computer algebra system for polynomial computations (2010), http://www.singular.uni-kl.de.

[13] X.S. Gao, J.Z. Zhang, S.C. Chou, Geometry Expert. Nine Chapters, 1998.

[14] C.M. Hoffmann, P.J. Vermeer, Eliminating extraneous solutions in curve and surface operations, International Journal of Computational Geometry and Applications 1(1) (1991) 47–66.

[15] D. Kapur, Using Gröbner bases to reason about geometry problems, Journal of Symbolic Computation 2 (4) (1986) 399–408.

[16] D. Kapur, Y. Sun, D. Wang, A new algorithm for computing comprehensive Gröbner systems, in: W. Koepf (Ed.), Symbolic and Algebraic Computation, International Symposium, ISSAC 2010. ACM 2010, pp. 29–36.

[17] D. Lazard, F. Rouillier, Solving parametric polynomial systems, Journal of Symbolic Computation 42 (6) (2007) 636–667.

[18] A. Montes, M. Wibmer, Gröbner bases for polynomial systems with parameters, Journal of Symbolic Computation 45 (12) (2010) 1391–1425.

[19] T. Recio, M.P. Vélez, Automatic discovery of theorems in elementary geometry, Journal of Automated Reasoning 23 (1) (1999) 63–82.

[20] E. Roanes–Macías, E. Roanes–Lozano, A Maple package for automatic theorem proving and discovery in 3D–geometry, in: F. Botana, T. Recio (Eds.), Automated Deduction in Geometry, ADG 2006, Springer-Verlag LNAI 4689, Berlin–Heidelberg, 2007, pp. 171–188.

[21] E. Roanes-Lozano, E. Roanes-Macías, M. Villar Mena, A bridge between dynamic geometry and computer algebra, Mathematical and Computer Modelling 37 (9–10) (2003) 1005–1028.

[22] E. Roanes-Lozano, N.v. Labeke, E. Roanes-Macías, Connecting the 3D DGS Calques3D with the CAS Maple, Mathematics and Computers in Simulation 80(6) (2010) 1153–1176.

[23] W.A. Stein, Sage Mathematics Software (Version 4.7), The Sage Development Team, 2011, http://www.sagemath.org.

[24] M.A. Spivak, A Comprehensive Introduction to Differential Geometry, vol. 3. Publish or Perish, 1979.

[25] A. Suzuki, Y. Sato, An alternative approach to comprehensive Gröbner bases, in: T. Mora (Ed.), Symbolic and Algebraic Computation, International Symposium ISSAC 2002. ACM 2002, pp. 255–261.

[26] D. Wang, GEOTHER 1.1: handling and proving geometric theorems automatically, in: F. Winkler (Ed.), Automated Deduction in Geometry, ADG 2004, Springer-Verlag LNAI 2930, Berlin–Heidelberg, 2004, pp. 194–215.

[27] V. Weispfenning, Comprehensive Gröbner bases, Journal of Symbolic Computation 14 (1) (1992) 1–30.

[28] W. Wen–Tsun, On the decision problem and the mechanization of theorem–proving in elementary geometry, A.M.S. Contemporary Mathematics 29 (1984) 213–234.

[29] W. Wen–Tsun, Some recent advances in mechanical theorem–proving of geometries, A.M.S. Contemporary Mathematics 29 (1984) 235–242.

[30] M. Wibmer, Gröbner bases for families of affine or projective schemes, Journal of Symbolic Computation 42 (8) (2007) 803–834.

[31] URL: http://raumgeometrie.de.

[32] URL: http://education.ti.com/educationportal/sites/US/productDetail/us cabri geometry.html.

[33] URL: http://www.cabri.com/download-cabri-3d.html.

[34] URL: http://www.calques3d.org.

[35] URL: http://www.geogebra.org.

[36] URL: http://www.geogebra.org/webstart/5.0/geogebra-50.jnlp.

[37] URL: http://i2geo.net.

[38] URL: http://www.keypress.com/x5521.xml.

[39] URL: https://github.com/jasongrout/simple-python-db-compute.

[40] URL: http://sagenb.org/home/pub/3094.

[41] URL: http://webs.uvigo.es/fbotana/Using-AutDed3D-with-SCS.