USING AUTOMATIC THEOREM PROVING TO IMPROVE THE USABILITY OF GEOMETRY SOFTWARE

ULRICH KORTENKAMP AND JÜRGEN RICHTER-GEBERT

ABSTRACT. Dynamic or interactive Geometry software (DGS) is the mathematical version of vector based drawing software: the objects (points, lines, circles, conics, polygons, etc.) are both graphical and mathematical entities. This allows adding relations between the objects that govern their behavior. Thus, DGS is used as an input tool for *constructions*, as opposed to simple *drawings*.

The additional information that is present in a construction can be used to greatly enhance the usability of DGS. We show how automatic theorem proving can be used to

- remove redundant elements in a construction that obstruct a smooth work flow
- clarify the semantics of user actions, and
- improve the graphical rendering of elements.

Finally we discuss the various possibilities of transforming the *mathematical* tool DGS into an *educational* tool. Here, automatic theorem proving is used to analyze user actions and to react properly.

1. INTRODUCTION

Dynamic or interactive Geometry Software (DGS) has been established as a major tool in education during the last decade. The most widely used packages are the "classical" ones, *Geometers' Sketchpad* [16] and *Cabri Geometry* [20], that were the first to introduce this type of software into classrooms in the beginning of the 1990's. Today, there are numerous other packages. This includes free software, shareware and commercial implementations.

All these packages share the same constructive approach to geometry: Starting with some *free* elements, it is possible to create *dependent* elements that are calculated on the basis of other elements. Geometric constructions are build up step by step. There are few restrictions on what the user is doing with the software. An important one is that the dependency graph of the elements must not contain a cycle; however, this constraint is maintained automatically. As each construction step can only use elements that are already constructed, we have a canonical linear ordering of the constructed elements that is compatible with the dependency relation.

1.1. **Cinderella.** For several years we, the authors, have been developing another DGS called *Cinderella* [26]. Its primary focus is not on K-12 education, but it has been developed as a tool for mathematicians in research, publishing and teaching. This is reflected by its native support of non-euclidean geometries, multiple viewports (for example, the Poincaré disc), high quality print output, and an integrated randomized theorem prover. An in-depth description of many of these can be found in [17].

The software is written in Java and thus available on a wide variety of platforms. It is also possible to export interactive constructions, animations and exercises for web pages, that are accessible using a standard browser with no extra software installation [19]. Thus, it is extremely suitable for distance education web sites. Many educational sites use Cinderella, for example the MADIN (*Mathematik-Didaktik im Netz* — Mathematics Education on the Internet) project [30].

1.2. **Automatic Theorem Proving.** Geometry is an important area for Automatic Theorem Proving (ATP), the field of using automated methods for creating mathematical proofs (for example, by using a computer). The exactness and broad theoretical foundation that is present in geometry *and* the beauty and elegance of geometry make it a wonderful testbed for new algebraic and other methods.

This wakes the desire for matching the strengths of the formal computational methods in ATP with the user interface approach to geometry as presented by DGS. One of the first packages to do this was Geometry Expert (*GEX*, formerly *GE*) of Gao, Zhang and Chou [8]. There, the DGS is used as an input tool for several theorem proving methods, like Wu's method or Gröbner basis approaches. Many other approaches have been made, on all levels of integration. The ATP methods used are not restricted to symbolic algebraic computation with polynomials, but they also include, e.g., logic reasoning in Prolog[14, 13] Other recent examples are

Supported by the DFG research center "Mathematics for key technologies" (FZT 86) in Berlin.

MMP/Geometer of the same authors [9], GeoView [2] that combines the dynamic geometry drawing tool *GeoplanJ* and the *Coq* theorem prover, or *Discover* [4].

Actually, the development of Cinderella was also initiated by the need of an input tool for the so-called binomial prover of Richter-Gebert [24]. This prover was still present in the earlier versions of Cinderella, but was dropped for a different approach later. We will explain this decision in Sec. 2, but we highlight two different principal approaches to automatic theorem proving first.

We will not give an overview of ATP using computer algebra here, instead we refer to the substantial amount of existing literature, and we suggest to start with the proceedings of the workshops on Automatic Deducation in Geometry [32, 10, 27, 33]. What is more important for us is the fact the methods using computer algebra are not applicable for all situations, but one has to carefully select the right method for each situation. If the wrong method is used, the method may fail, or the time to prove or disprove the theorem is in the range of a few hours, days or years, both of which is not acceptable.

There are also other approaches. Given an instance of a construction that shows an example for a theorem — will this be accepted as a proof? This is certainly not the case. What about two, three, thousand examples? Still this is no proof, as the examples could be carefully selected single instances. The situation changes dramatically when the examples are chosen randomly from a proper set of possible instances. If we can give an estimate of the probability to choose a counterexample if there is one, we can bound the probability of failing if we assume that a theorem is true based on a few random examples of it. This method is also referred to as *randomized theorem proving* or *checking*, and was introduced 25 years ago [29].

Randomized methods at first sight might seem to be less exact than computer algebra methods. But if one takes care one can be sure that the result is as reliable as any other, symbolic, method: Moreover, if the probability of a wrong decision is less than the probability of a software or hardware failure, then there is no more reason to distrust the method than there is with any other computer-based approach.

The main advantage of randomized methods is that they are universally applicable and usually they have a guaranteed maximal running time. We pay for that by not being able to prove all theorems, and we do not get a certificate of any kind for the proof. Thus, by randomized proving methods one can hope to get a little bit more information within a reasonable amount of time, but there is *no guarantee* to get any information.

Cinderella uses a method that is roughly based on the Schwartz-Zippel-Lemma that estimates the number of zeros of a multivariate polynomial of a given maximum total degree. This is combined with continuation methods to generate other instances of a conjecture on the same connected component. Again, we refer to [17] for a more detailed description. Here, it is only important that the software is able to quickly decide whether two elements in a construction coincide by coincidence or because of a theorem that forces them to coincide. We accept “no theorem” as an answer if the prover was not able to prove the theorem, even if it is possible to prove the theorem with other methods. On the other hand, it is not allowed for the theorem prover to falsely announce a “theorem”.

2. Applications of ATP for the User Interface

Up to now, attention was concentrated on using DGS to improve the user interface of automatic theorem provers. In this article we investigate and describe the new possibilities that arise from using an ATP to *support the user interface of DGS*, i.e., we do not use the DGS to create input for the ATP, but we use the ATP to modify or create output of the DGS. This implies that we have to be able to run an automatic proving method with *any user input* coming from the DGS in *realtime*. For these reasons, our method of choice is randomized theorem proving, although it might be assisted by other symbolic methods.

2.1. **Proving Mechanism.**

The theorem prover in Cinderella is not triggered explicitly by a user action, but it is run automatically every time an element is added to the construction. Usually, the users do not have to care at all about this — they will probably not notice that Cinderella tries to analyze the construction on the fly.

When an element is added, Cinderella checks whether the new element is *identical* to an already existing element first. An element A is considered to be identical to another element B, if A’s coordinates are the same as B’s coordinates under movement of any free elements.

If the prover finds an element that is identical to the new one, all further actions will use the already existing element instead, and the new element is not added to the construction. Thus, by induction, we maintain

an important property of the construction sequence: For every pair of elements there is an instance[1] of the construction such that they have different coordinates — there is no redundancy of elements.

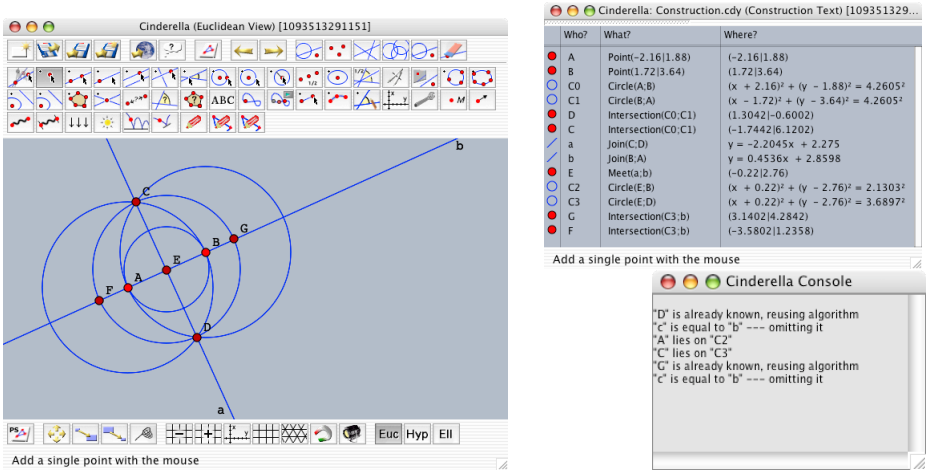

Figure 1: Messages of the randomized prover while doing a construction

Next, if we are really adding a new element, Cinderella checks for every existing element whether the new element is incident to it (e.g., whether a point lies on a line or conic, or a conic meets a point). This information is stored in a list of incidences with the elements.

Many constructions force trivial incidences: for example, a line ℓ defined as the connecting line (join) of two points A and B will be incident to them; it should not be necessary to run the prover for these "conjectures". In fact, Cinderella generates these trivialities in advance and filters them from the prover input. This avoids unnecessary calculations and increases the reliability of the prover's results. It is also possible to use other proving methods like symbolic methods to generate "already known facts", which would combine the speed and simplicity of the randomized prover with other exact methods.

2.2. **Removing Redundancy.** As mentioned above, we are able to guarantee a certain uniqueness of the elements contained in a construction. This is probably the most visible use of ATP in Cinderella.

2.2.1. *Removing Double Entries.* A common user interface problem in DGS is created by double elements. Let us assume that a user adds two free points A and B to a construction, creates their midpoint M and then adds two lines, the connecting line of A and M and the connecting line of B and M. Clearly, these two are identical for every position of A and B. Symbolically, they are different, and almost every DGS will allow the user to add both. If the software renders lines correctly[2], we will no be able to see them both, but one will hide the other. Now assume that the users wants to construct the intersection of *this line* (he cannot distinguish them) with another one later. The reaction of the software differs from package to package, but usually the users is either prompted for a choice between these two (identical!) lines, or the software will refuse to intersect three lines. In any case, further interaction or thought is needed that could be avoided if the two identical lines had been replaced by a single line.

Even if he does not want to create an intersection, but just place a point on the line through A, B and M, this "double line" will create this kind of confusion: It can happen that the *point on the line* will become the intersection of the two identical lines and thus be invalid; or, again, further interaction is required to choose the right one of these two lines.

If a user knows that the software automatically cleans up its data structures using this technique, he can also avoid many unnecessary mouse actions. Here is the most basic example: Say, you want to construct a line through A and B and a point C on that line. With Cinderella, you use the "insert line with two points" mode and do a press-drag-release action with the mouse to draw A and B and the line. Then, instead of switching the mode, you just press and drag again starting at B and ending on the line. This inserts another line that goes through B and a new point C on the first line. As they are provably identical, the line is omitted from

[1] An instance of a construction is an assignment of coordinates to all elements that is compatible to all construction steps

[2] Many packages have problems with this due to numerical inaccuracies. We will not discuss this here.

the construction and you end up with the desired result. This technique can be used in many places, and it decreases the number of mouse actions required for many constructions considerably.[3]

2.2.2. *Using Existing Algorithms.* Several elementary constructions in DGS are ambiguous: There are two intersections of a circle and a line, there are two angular bisectors of two lines, there are four intersections of two conics, etc. It is common to these constructions that calculating one element is not possible without calculating the other ones, too — or at least it is not much faster to calculate only one. Let us illustrate this with the intersections of two circles: At some point in the calculation it is necessary to solve a quadratic equation. The standard solution is composed of a linear part and a square root part that is either added or subtracted. At almost the same cost as for one solution we can calculate both solutions, it is just a matter of one additional addition or subtraction.

Cinderella uses this wherever it can by sharing the calculation between elements. If we need both intersections of two circles, we just call the intersection code once, and we get back the coordinates of both intersections.[4]

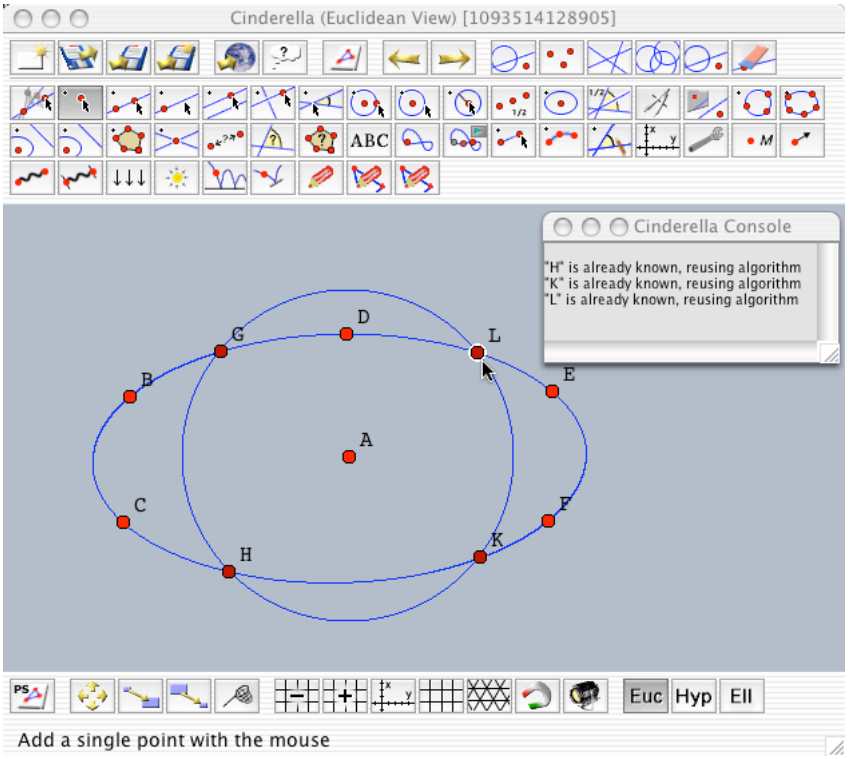

Figure 2: Reusing intersection algorithms. The calculation of the intersection point G is automatically reused for H, K and L.

This calculation sharing is achieved automatically using a simple trick: The yet unused elements that are calculated by an algorithm are included in the proving mechanism described above. If we can prove that an unused element is identical to the new element to be inserted, we just bring it to life, add a label to it and insert it in the list of construction elements.

In fact, although it does not seem so, this *is* a user interface application of ATP. Re-using existing algorithms ensures that different output elements of algorithms will be distinct. A striking example from the earlier days of Dynamic Geometry Software was that it could happen that a mirror point coincided with the original point — caused by identifying the two different outputs of the circle-line-intersection used to construct the mirror point.

[3]The number of mouse clicks and drags to do a construction in several geometry packages has been counted and compared by Jean-Marie Laborde, but unfortunately this seems to be unpublished.

[4]Due to the continuity methods in Cinderella, we even *have to* calculate both intersections if we only need one, because we cannot decide in advance which one we will need.

2.3. **Improved Rendering.** Fast and correct rendering of geometric elements is a key ingredient for a good user interface of geometry software. We already mentioned that it is problematic to have identical renderings of identical lines due to numerical inaccuracies. We will now show a few similar applications of ATP.

2.3.1. *Automatic Endpoints of Line Segments.* In Cinderella there are two types of line segments: One type is kind of "graphics-oriented", these segments are given by their two endpoints. The other type has its roots in incidence geometry: A line can be clipped to its endpoints — those two extreme points on both ends that are "next to infinity".

The second type is very handy when it comes to visualizing geometric facts. For example, when we add the intersection of two heights in a triangle, then a perpendicular line through the third vertex of the triangle can be clipped automatically to the intersection of them. If we were using ordinary lines, we would not get the immediate visual feed back. In fact, it is not possible to insert segments in the first place, as the second endpoint of the segment is not available when the height is constructed. This means, we also remove the need for constructing the intersection of heights with lines and hiding them again (See Fig. 3, where this is demonstrated with another typical triangle incidence theorem).

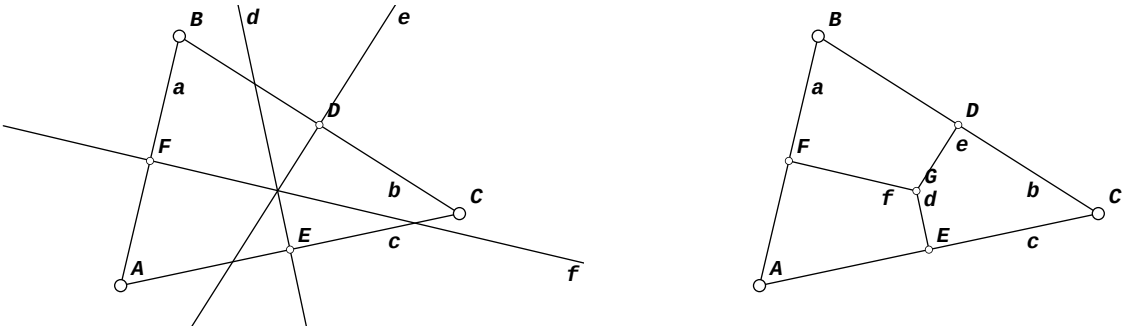

Figure 3: The intersection of the three perpendicular bisectors in a triangle. All bisectors are lines that clip automatically to their endpoints (left), so inserting the intersection point clips all three of them immediately (right). Without this type of auto-clipping lines we had to insert the segments manually and hide the lines after that. Without ATP the software could not recognize that it should clip *all three* lines.

Another effect is that, if we added the base points of the heights, then the base line of the triangle will be extended as necessary (Fig. 4). In order to implement this behavior, we completely rely on the incidence information created by the theorem prover.

Figure 4: Automatic extension of the base line of a triangle. The point *D* is incident to the line *AC*, and the triangle side is given as a line with automatic endpoint clipping.

2.3.2. *Circles replacing Generic Conics.* Cinderella supports generic conics in addition to (Euclidean) circles. There are no primitives in Java that support drawing conics as exact as we need them, so we had to implement a rendering algorithm that finds a polyline approximation of the conic. This algorithm uses both a lot of additional memory and time, and it is desirable to use the circle drawing primitives supplied by the graphics library (which are handled in hardware these days) whenever possible.

Again, we can use ATP to prove that a conic is always incident to the two special points I and J with complex homogeneous coordinates $(1,i,0)$ and $(1,-i,0)$, resp. If so, then the quadratic equation defining the conic will have no mixed terms and is of the form $ax^2+ay^2=r^2$. The conic is a euclidean circle and we can switch to the faster circle drawing routines.

2.3.3. *Conics, Circles, Lines replacing Generic Loci.* Loci are curves generated by dependent elements under movement of a free or semi-free (bound to one-dimensional space) elements. Famous examples are the curves generated by 4-bar-linkages or any other classic curves (cycloids, etc.).

Cinderella supports the automatic generation of these curves. Unfortunately, it is quite expensive to correctly generate enough points for a proper rendering: The whole construction that creates the locus has to be moved at a varying speed, and there is no easy way to suggest the correct speed of the free element to generate a dense enough, but not too dense, set of interpolation points. As these curves can extend to infinity, we have to choose the right speed, depending on the currently visible portion of the Euclidean plane. Some heuristics can be applied, but still it is a problem to render loci correctly.

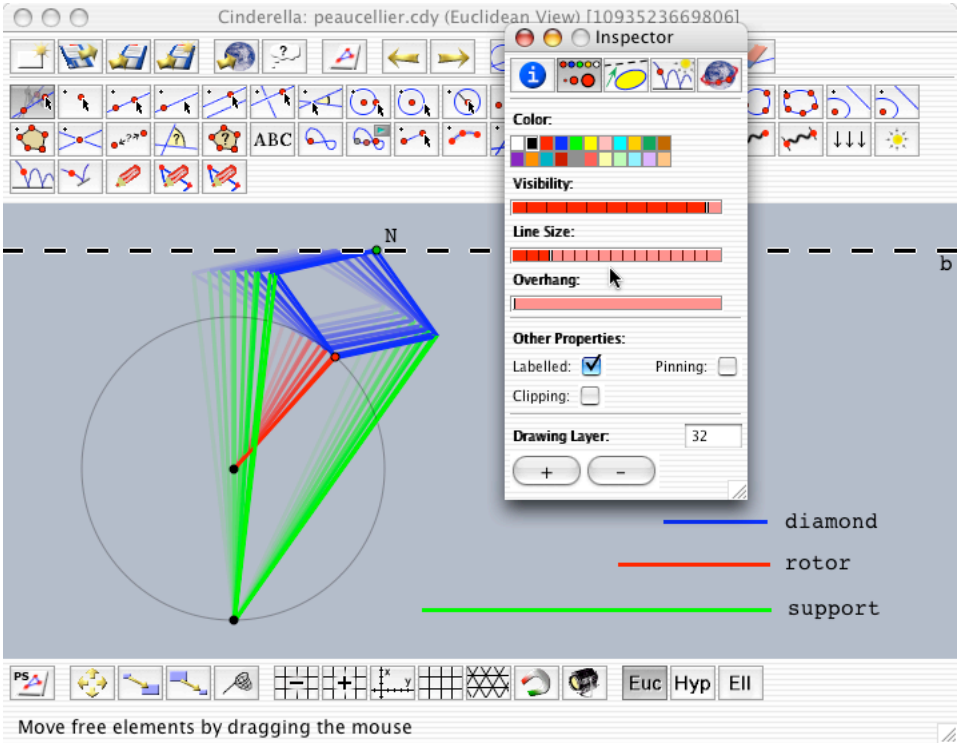

Figure 5: The locus of N is shown as a dashed and labeled line b. Cinderella uses the information gained from the internal ATP to render it. For generic loci, Cinderella does not support automatic placement of labels.

In many cases, the loci that are generated are much simpler curves, namely circles, conics, or even lines. A nice example is the Peaucellier linkage, that is able to convert a rotary motion into a linear motion (see [7] for an interactive demonstration done with Cinderella).

If we extend our notion of identity between elements and let linear, circular or conical loci be identical to the lines, circles. or conics they describe, we have a powerful way of replacing the locus tracing with the direct drawing of a line, circle, or conic.

We implemented this using the following approach: Whenever a locus is added, we also add a line, a circle and a conic that are defined by the first two, three, or five interpolation points of the locus. Next, we use the randomized prover to check whether these coincide with all other points of the locus, even under movement of other free elements. If so, we replace the new locus with the line, circle, or conic, and drop the two others. If not, we drop all three from the construction.

Of course, it is not only possible to improve the rendering using this technique, but it is also possible to use the loci for further constructions. For example, one can construct intersections of these curves with other objects. This is not just a nice-to-have feature, but requested by users, as can be seen in this forum posting [6]:

> EUKLID DynaGeo Forum: Wünsche und Vorschläge:
> Erkennen von Ortslinie Kreis
>
> Ich finde es schade, dass das Programm im Fall einer Strecke mit festem Radius und einem fixierten Punkt die Ortsline des zweiten Punktes nicht als Kreis erkennt. Die Linie wird als nur aufgezeichnet interpretiert und kann nicht dynamisiert und schon gar nicht in einen Kreis umgewandelt werden. Vielleicht lässt sich da mittelfristig ja was machen?

In English:

> EUKLID DynaGeo Forum: Wishes and Suggestions:
> recognition of circle locus
>
> It's a pity that the software does not recognize that the locus of the second point of the second point of segment is a circle when the other point is fixed. It's not possible to make the locus dynamic at all, and it cannot be changed into a circle. Maybe it's possible to introduce this in a later version?

The answer by the author of the DGS EUKLID Dynageo [22], Roland Mechling, points out that there might be chance to include this in a later version of his software. He suggests that the software could try to guess whether it should replace the locus by a circle based on the circularity of it. We want to stress that the ATP method included in the 1.0 version of Cinderella is doing this already in a reliable and mathematically correct way. In Fig. 6 this is demonstrated with the Peaucellier locus line and a circle.

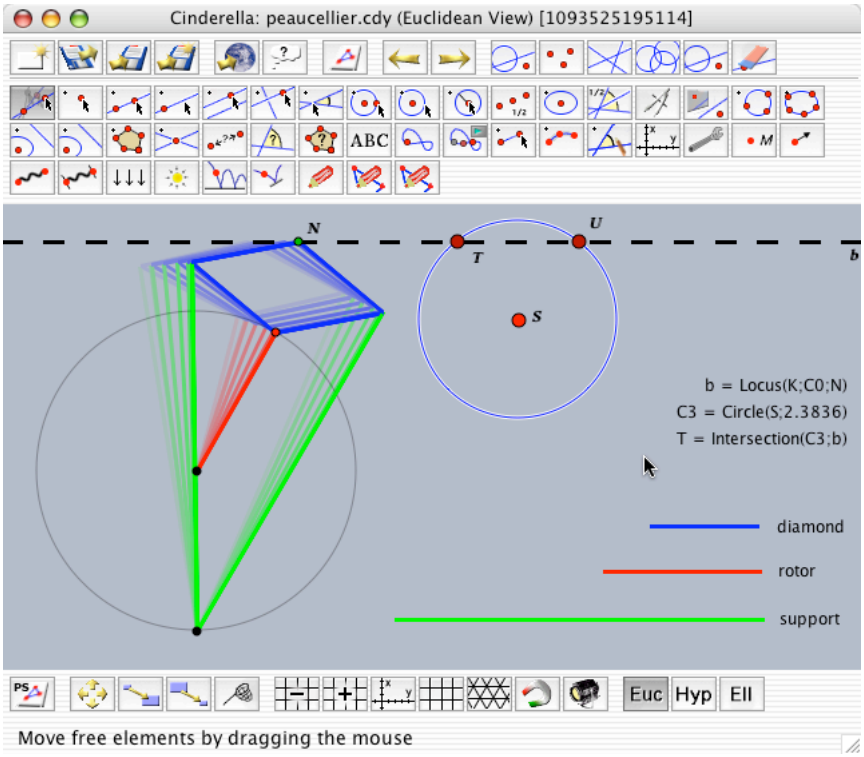

Figure 6: The locus of N is the line b. This can be intersected with another circle or line; here we use the circle $C3$ with midpoint S.

At this point, we have to mention that there is another new geometry tool that is specialized on using algebraic methods to analyze loci called *Lugares* [5, 3]. Another approach of using the combination of Computer Algebra and Dynamic Geometry is *SPICY* [31].

The latest release of Cabri Geometry, *Cabri Geometry II Plus*, also shows the equations of loci; we do not know whether this information is used for rendering them. However, the result is derived numerically, and there is no performance guarantee:

> For a locus, the algorithm used to determine its equation is numerical, and produces algebraic functions of degree no greater than 6. For loci whose points are of very different magnitudes, numerical errors appear very rapidly as the degree increases. The result produced is to be interpreted with caution. In particular, it may be necessary to restrict the domain of definition of the locus, in order to obtain a correct equation from the algorithm. If, for example, a locus is generated by a point moving on a line, a better result may be obtained by restricting the point to a segment of the line. [1]

2.3.4. *Scribbling.* In [18, 21] we describe an extension of Cinderella that supports pen-based input and preserves the sketched look of points, lines, and circles. However, we want to avoid that the sketches are mathematically incorrect — a user using a computer tool for geometry can expect to have correct drawings, and it should not be possible to create fakes.

The general idea is to use the incidence information gathered by the prover to force lines to go through their correct position (when straight) wherever there is an incident point. For every line and circle we store the deviation of sketched line from the straight line, and modulate this data by a wave that meets the straight line at the incidences. This will change the look of a sketch slightly, but it both preserves a sketched look similar to the original one, and ensures the exactness of the drawing.

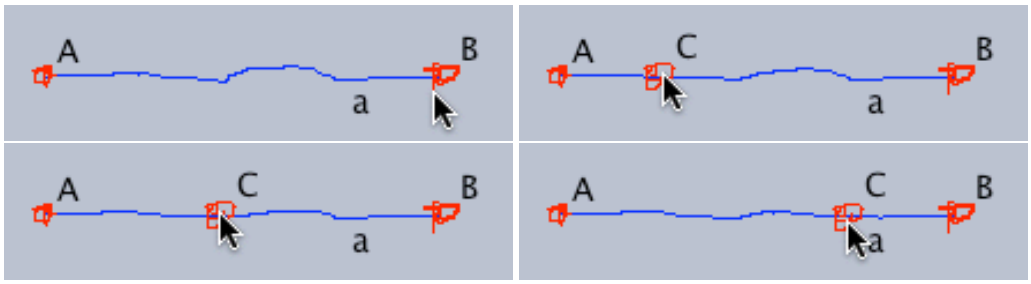

Figure 7: Changes to a scribbled line segment *AB* caused by an incident point *C*

In Fig. 7 the effect on a scribbled line by an incident point is shown. A much nicer example is shown in Fig. 8: The circumcircle of a triangle has been constructed, and Cinderella automatically adjusts the circle to be incident to all three vertices.

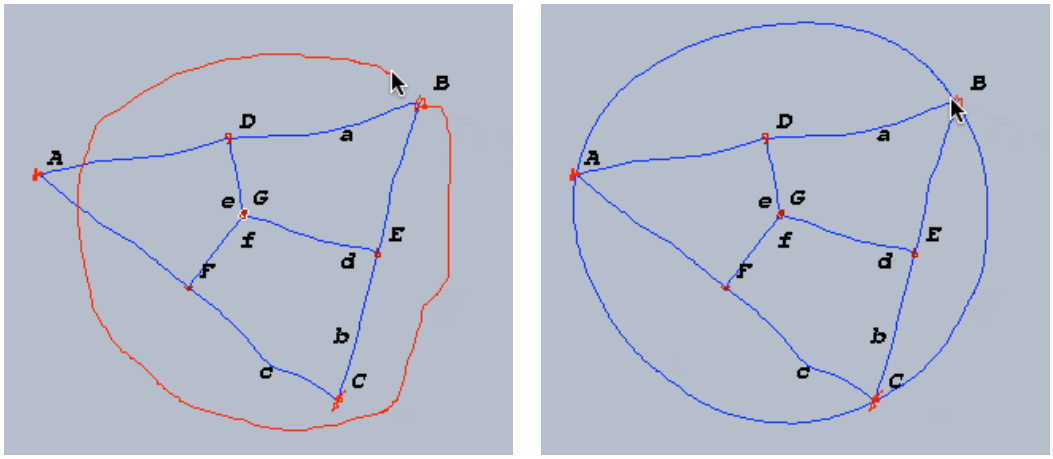

Figure 8: Changes to the circumcircle: The user is drawing a circle starting at *B* aroung *G* that is not incident to *A* and *C* — contradicting the circumcircle theorem. The left screenshot shows the situation immediately before the user releases the pen, the right screenshot shows the situation immediately after this action. Using the information gained from the ATP engine Cinderella adjusts the circle to be correct while preserving its sketchy look.

2.4. **User Input Interpretation.** Sometimes the semantics of a users' action can be unclear. For example, when he tries to insert the intersection of three lines by placing a point on it, it is unclear whether he assumes that these always meet in one point or not. A software that "knows" whether the three lines are always concurrent can react properly in that situation: When they are, it is not necessary to ask which pair of points should define the intersection, as all three define the same one. When they are not, the software can signalize that it needs the attention of the user to resolve an ambiguity.[5]

[5]In Cinderella, we do not use this approach, although we have the necessary information. However, the default action of Cinderella is to take any pair of lines, without caring too much about which one the user really wants. If Cinderella chooses the wrong ones, the user can always undo his action and redo it using another magnification or the specialized "define intersection by selection" tool. This is different from other DGS, e.g., Cabri Geometry II, where a pop-up menu or a similar widget is used to disambiguate. The ATP method described here could be used in such a scenario.

Most ambiguities caused by the user that arise in DGS are handled anyway by the removal of double elements as described in the beginning of this section. Other approaches are possible, but are not implemented yet; they are described in Sect. 2.5.

2.4.1. *OpenMath Interface.* A recent extension of Cinderella [28] is an interface to computer algebra software via OpenMath [23]. Here we want to use Cinderella in the way of its original concept: as an input tool for theorem proving algorithms.

This first prototype converts a construction into OpenMath code (using the preliminary content dictionary for elementary geometry), translates this into OpenMath objects describing polynomial equations, and feeds this using a suitable *phrasebook* into a computer algebra software (here GAP [11]) that is used for proving a conjecture.

Which conjecture is chosen is decided using the information of the internal randomized prover; there is no additional user interaction necessary besides constructing an instance of the theorem. The OpenMath encoder automatically uses the last theorem that was found by the randomized prover and asks for a symbolic proof. This combines both methods and makes it very easy to explore geometric relations.

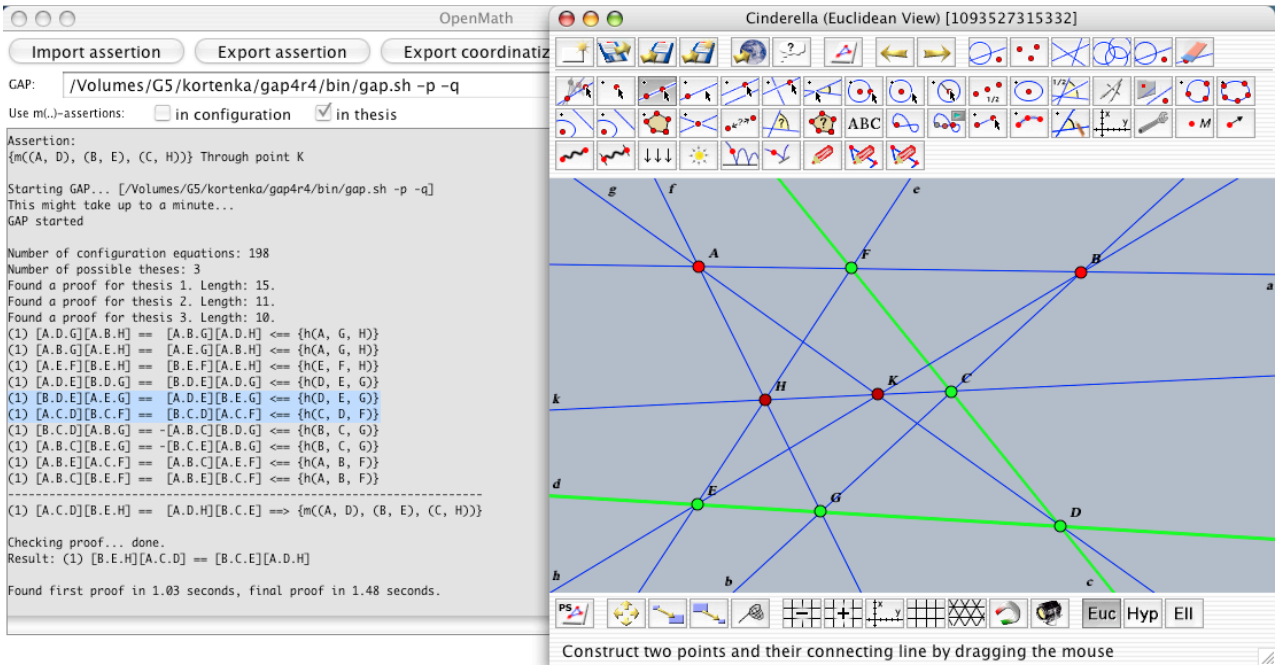

Figure 9: Using the OpenMath interface for proving theorems. On the right you see Pappos' theorem as constructed by the user. On the left you see the output of the symbolic proving engine that first converts the construction into OpenMath code that encodes the hypothesis. The conjecture is automatically taken from the randomized prover of Cinderella ("K lies on k" — where k is the meet of AD, BE and CH.), and also encoded in OpenMath. This is piped to GAP, and the proof is send back to Cinderella. Highlighting the two identities in the proof also highlights the corresponding geometric facts on the right (in green).

2.5. **Extensions.** There are many more ways to incorporate ATP into the user interface of DGS. An obvious application would be to use it to find polynomial relations between elements that might simplify the construction sequence. Or, one could try to find constants (like the 60 degrees that occur in an equilateral triangle). Whenever we find two identical elements, we could also use this information to stabilize the construction numerically, or make its calculation more efficient.

Finally, it is also a possibility to use the additional information of the prover to make more choices when it comes to interpreting the users' actions. We want to illustrate this with an example.

Both in the context of pen-based input of constructions [18] and with traditional input there often can be several possibilities to interpret user input. Say, the user defines a circle by dragging from its midpoint to a boundary point. There can be several possible positions of both. In Cinderella, both points snap to existing points, intersections or single lines, in that order. That is, when an existing point is near an intersection, it will be preferred to the point at the intersection. With ATP in place, one could define another priority: For

all possible placements of midpoint and boundary point (at the mouse position, at existing points or intersections near the mouse position, or other “snap points”) we test whether how many new theorems (unexpected incidences or identities) will be found. The placements that defines the most new theorems will be used, as it seems to be the most interesting one — and we assume that the user wants to do something interesting.

This

3. Exercises

As a last aspect of using ATP in DGS we want to briefly mention interactive, self-checking exercises. They are explained in much more detail in [19, 17, 26, 25], but as they constitute a major application of the techniques described here we cannot omit them.

Suppose you want to design a construction exercise or exploration activity that should be done with a DGS. As a teacher, you will have the problem that you cannot supervise all of your students, not only when they do the assignment at home, but even when they work in the classroom. You need a tool that is able to recognize the students’ actions and to provide assistance that is suited to the current progress of the student within the activity. When the exercise has been finished satisfactorily, the software should be able to identify this.

This identification is the key for us to employ automatic theorem proving: If we want to know whether a construction (by the student) is equivalent to another construction (by the teacher), we have to be able to prove geometric theorems automatically. The same is true for intermediate steps that can be used as “milestones” on the solution path.

In Cinderella we support these exercises by storing a — hidden — example construction of the teacher that may be revealed step by step when the student is asking for a hint. As the randomized theorem prover also proves identity of the hidden elements with the elements constructed by the student, the software is able to rate the progress of the student (Fig. 10).

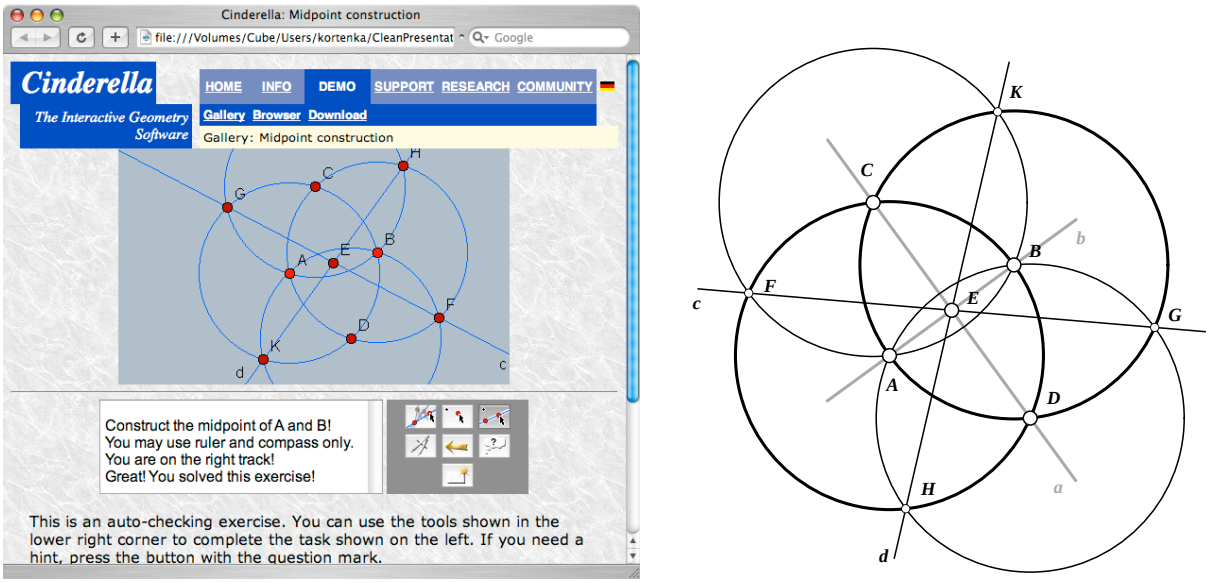

Figure 10: An exercise running in a web browser (here: Safari, left). On the right the comparision of the hidden standard construction for the midpoint (in gray) and the students’ construction.

It is of importance that we cannot just compare the two construction sequence literally. Not only the order of the construction steps may be different, but also the basic construction steps. It is important for the learning process of the student that he is able to use alternate solutions and is not restrained to the exact solution path given by the teacher. We should take care that computers are used a tool that stimulates creativity, motivates students and encourages them to think independently.

For a further discussion of interactive exercises with Cinderella, we refer to [17, 26].

4. Conclusion and Acknowledgments

In this article we have shown how several improvements in the user interface of Cinderella are directly caused by its integrated randomized prover. The most impact is created by removing double elements in a

construction automatically: There are less ambiguities to be resolved, and users can rely on what they see on screen and are not confused by hidden elements that change the semantics of their actions. The number of mouse actions and mode changes is reduced.

In the rendering step, the additional information of the prover is also used to improve the visual quality and the rendering speed. In a development version of Cinderella it is also used to create mathematically correct sketches. Finally, ATP is crucial for the implementation of the educational extensions of Cinderella, where students are guided through self-checking exercises.

We do not want to conceal the fact, that from a didactical point of view it is not at all clear, whether the user interface of a Dynamic Geometry Software package should be improved in the ways we describe. Removing the obstacles in handling might also remove welcome occasions of reflection for the students. A software that is able to guess what the user intends will be of help to the experienced user, but may be (irritating) magic to beginners. Actually, it might happen that the concepts of geometry that are acquired by the learners are changed due to the use of DGS [12, 15]; and this effect could be amplified by a "better" user interface.

However, we strongly support the opinion that it has to be considered harmless to remove artificial obstructions that have no relation to mathematical concepts, like the double-lines mentioned in Sec. 2.2.1. It is in the responsibility of the teacher to slow down the process according to the individual learners' knowledge, it should not be caused by technical deficiencies of the software.

Many thanks to Dirk Materlik, who joined the Cinderella team recently and is of invaluable help.

REFERENCES

1. Eric Bainville, *Cabri Geometry II Plus User Manual*, Cabrilog, 9 2002, http://www.chartwellyorke.com/cabriplusmanual.pdf.
2. Yves Bertot, Frédéric Guilhot, and Loïc Pottier, *Visualizing geometrical statements with GeoView*, 2 2004, http://www-sop.inria.fr/lemme/geoview/geoview.ps.
3. Francisco Botana, *Interactive versus symbolic approaches to plane loci generation in dynamic geometry environments*, Computational Science - ICCS 2002: International Conference, Amsterdam, The Netherlands, April 21–24, 2002. Proceedings, Part II (Heidelberg) (P.M.A. Sloot, C.J. Kenneth Tan, J.J. Dongarra, and A.G. Hoekstra, eds.), Lecture Notes in Computer Science, vol. 2330, Springer-Verlag, 8 2003, http://springerlink.metapress.com/openurl.asp?genre=article\&issn=0302-9743\&volume=2330\&spage=211, p. 211ff.
4. Francisco Botana and José L. Valcarce, *A dynamic-symbolic interface for geometric theorem discovery*, Computers and Education **38** (2002), no. 1–3, 21–35, http://dx.doi.org/10.1016/S0360-1315(01)00089-6.
5. ———, *A software tool for the investigation of plane loci*, Mathematics and Computers in Simulation **61** (2002), no. 2, 139–152, http://dx.doi.org/10.1016/S0378-4754(02)00173-8.
6. Andreas Briegel, *Erkennen von Ortslinie Kreis*, Message in the EUKLID Dynageo user forum, 2 2004, http://www.dynageo.de/discus/messages/14/232.html?1076183239.
7. Leo Dorst, *Compact peaucellier cell*, http://carol.wins.uva.nl/~leo/lego/peaucellier.html.
8. Xiao-Shan Gao, *Geometry Expert*, Software Package., http://www.mmrc.iss.ac.cn/~xgao/gex.html.
9. Xiao-Shan Gao and Qiang Lin, *MMP/Geometer - A Software Package for Automated Geometric Reasoning*, Automated Deuduction in Geometry, 4th International workshop, ADG 2002 (Franz Winkler, ed.), Lecture Notes in Computer Science, vol. 2930, Springer-Verlag, 2004, http://springerlink.metapress.com/openurl.asp?genre=article\&issn=0302-9743\&volume=2930\&spage=44, pp. 44–66.
10. Xiao-Shan Gao, Dongming Wang, and Lu Yang (eds.), *Automated deduction in geometry, second international workshop, adg'98, beijing, china, august 1-3, 1998, proceedings*, Lecture Notes in Computer Science, vol. 1669, Springer, 1999.
11. The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.4*, 2004, http://www.gap-system.org.
12. Tansy Hardy and Dave Wilson, *Musings upon Geometry and Dynamic Geometry Software*, World Conference on Computer Education, Workshop on Dynamic Geometry (Birmingham), 7 1995, http://s13a.math.aca.mmu.ac.uk/Daves_Articles/Dynamic_Geometry_Software/DGS.html.
13. Gerhard Holland, *Geolog 2002 professional*, http://www.uni-giessen.de/~gcp3/geolog.htm.
14. ———, *GEOLOG-WIN. Konstruieren, Berechnen, Beweisen, Problemlösen mit dem Computer*, Reihe Computer-Praxis Mathematik, Dümmler, Bonn, 1996.
15. Reinhard Hölzl, *Im Zugmodus der Cabri Geometrie*, Ph.D. thesis, Universität Augsburg, 1994.
16. Nicholas Jackiw, *The Geometer's Sketchpad*, Key Curriculum Press, Berkeley, 1991–1995, http://www.keypress.com/sketchpad.
17. Ulrich Kortenkamp, *Foundations of Dynamic Geometry*, Ph.D. thesis, ETH Zürich, 11 1999, http://kortenkamps.net/papers/diss.pdf.
18. Ulrich Kortenkamp and Dirk Materlik, *Pen-based input of geometric constructions*, Proceedings of the MathUI Workshop 2004, 2004, Accepted for publication. http://kortenkamps.net/papers/2004/Scribbling-article.pdf.
19. Ulrich Kortenkamp and Jürgen Richter-Gebert, *Geometry and education in the internet age*, Proceedings of the ED-MEDIA & ED-TELECOM 1998 World Conference on Educational Multimedia, Hypermedia and Telecommunications (Freiburg), Association for the Advancement of Computing in Education, March 1998, http://www.cinderella.de/papers/geo-i.pdf.gz, pp. 790–799.
20. Jean-Marie Laborde and Franck Bellemain, *Cabri-Geometry II*, Texas Instruments, 1993–1998, http://www.cabri.net.

21. Dirk Materlik, *Using sketch recognition to improve the user interface of geometry software*, Master's thesis, Freie Universität Berlin, 2004, http://kortenkamps.net/diplomarbeiten/DirkMaterlik.pdf.
22. Roland Mechling, *EUKLID Dynageo*, Shareware, http://www.dynageo.de.
23. The OpenMath Consortium, *Openmath*, Website, http://www.openmath.org.
24. Jürgen Richter-Gebert, *Mechanical theorem proving in projective geometry*, Annals of Mathematics and Artificial Intelligence **13** (1995), no. 1–2, 139–172.
25. Jürgen Richter-Gebert and Ulrich Kortenkamp, *Die interaktive Geometriesoftware Cinderella*, Book & CD-ROM, HEUREKA-Klett Softwareverlag, Stuttgart, 1999, German school-edition of the Cinderella software.
26. _____, *The Interactive Geometry Software Cinderella*, Springer-Verlag, Heidelberg, 1999, http://cinderella.de.
27. Jürgen Richter-Gebert and Dongming Wang (eds.), *Automated deduction in geometry, third international workshop, adg 2000, zurich, switzerland, september 25-27, 2000, revised papers*, Lecture Notes in Computer Science, vol. 2061, Springer, 2001.
28. Dan Roozemond, *Automatic geometric theorem proving*, Bachelorproject, Eindhoven University of Technology, 7 2003, http://www.win.tue.nl/~amc/ow/bachproj/BachelorProjectAGTP.pdf.
29. Jacob T. Schwartz, *Probabilistic algorithms for verification of polynomial identities.*, Symbolic and algebraic computation, EUROSAM '79, int. Symp., Marseille 1979, Lect. Notes Comput. Sci. 72, 200-215 (1979).
30. Martin Stein, Uwe Tietze, Hans-Georg Weigand, and Thomas Weth, *MaDIN – Mathematikdidaktik im Internet*, Website, 2004, http://www.madin.net.
31. Duco van Straten and Oliver Labs, *A visual introduction to cubic surfaces using the computer software* SPICY, Algebra, Geometry, and Software Systems (Michael Joswig and Nobuki Takayama, eds.), Springer-Verlag, 2003, http://enriques.mathematik.uni-mainz.de/labs/dagstuhl_proceedings_2001_10_OliverLabs.pdf.
32. Dongming Wang (ed.), *Automated deduction in geometry, international workshop on automated deduction in geometry, toulouse, france, september 27-29, 1996, selected papers*, Lecture Notes in Computer Science, vol. 1360, Springer, 1997.
33. Franz Winkler (ed.), *Automated deduction in geometry, 4th international workshop, adg 2002, hagenberg castle, austria, september 4-6, 2002, revised papers*, Lecture Notes in Computer Science, vol. 2930, Springer, 2004.

TECHNISCHE UNIVERSITÄT BERLIN, DIDAKTIK DER MATHEMATIK, SEKRETARIAT MA 7-3, STRASSE DES 17. JUNI 136, 10623 BERLIN, GERMANY

TECHNISCHE UNIVERSITÄT MÜNCHEN, ZENTRUM MATHEMATIK, LEHRSTUHL FÜR GEOMETRIE UND VISUALISIERUNG, BOLTZMANNSTR. 3, 85747? GARCHING, GERMANY