Diss. ETH N° 13403

Foundations of Dynamic Geometry

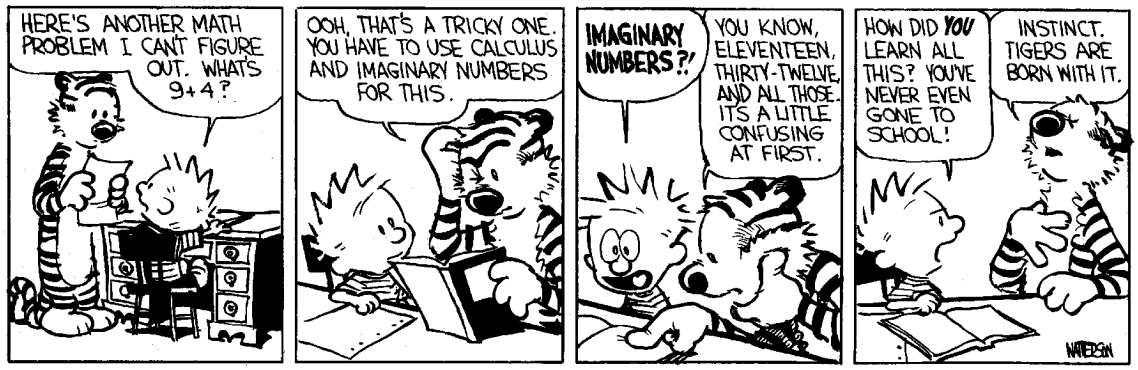
A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH
for the degree of Doctor of Technical Sciences

presented by
Dipl.-Math. Ulrich Kortenkamp
born 18th October 1970
citizen of Germany

accepted on the recommendation of
Prof. Jürgen Richter-Gebert, examiner
Prof. Günter M. Ziegler, co-examiner

1999

Preface



Although it is known that Calvin and Hobbes tell the truth about life, I was surprised that Bill Watterson knew in 1988 what Jürgen Richter-Gebert and I had to learn ten years later: Sometimes it is necessary to use imaginary numbers even for seemingly trivial tasks.

This thesis shall explain the details of a method called *complex tracing*, and lay the foundations of Dynamic Geometry, a new field of research that opened up after we solved the continuity problem for interactive geometry software.

I came into this project right after I decided not to write my thesis on *Cinderella*, the interactive geometry software which at that time was a project of Jürgen Richter-Gebert and Henry Crapo, but on neighborly polytopes. After the first few weeks of implementing the new version of *Cinderella* in Java I understood why it is really hard to write "just another geometry software." I had to try to implement a dynamic geometry software in order to understand why it is difficult to create a software that "behaves as expected." It needs a mathematical theory, and it was not clear to us what to do three years ago.

It was in early 1998 when we had our "ultimate break-through." The time since then was spent for the implementation of the theory, and it is nice to see that this implementation was published before we even started to write papers about it. This "proof of concept" did help me very much while preparing this thesis.

I want to thank Jürgen for all the discussions, arguments, quarrels, overnight just-in-time hacks, lessons in geometry, motivating break-throughs and especially for all the fun

that was needed to create the stimulating atmosphere that made the *Cinderella* project possible. Thank you!

Most of the results in this thesis are collaborative efforts of Jürgen and myself, and I will not try to separate who of us contributed what piece of the big puzzle. I feel very lucky to be in a work group that tries to concentrate on getting work done rather than non-disclosing ideas to avoid that they are "stolen."

Another person that I want to thank is Günter Ziegler, who gave me enough freedom to leave Berlin without feeling too guilty. I promise that I will work on neighborly polytopes again, someday. I learned a lot from Günter, and I admire his ability to put mathematical pieces in the right context and to translate between the different languages of mathematicians. He also enabled Jürgen and me to work on the *Cinderella* project in Berlin. Without that support it would have been much harder.

There are more persons that made this thesis possible. Among all the others I want to emphasize my friends Eva-Maria Feichtner and Alex Below, who had to share offices with me. Thanks go also to all the people in the work groups and the guests of Günter and Jürgen, and the many friendly mathematicians I met at conferences.

Last but definitely not least I want to thank my wife Doro and my children Mara and Julius, who had to suffer a lot. First I was busy to finish the software, then I was busy finishing this thesis – both took much longer than expected – and probably there will be something else next. Without your love and help I could never have done this, and there would be no reason to go through all this trouble. Thank you!

Wherever you find the words "his" or "him" or "he" in this thesis, you can try to replace them by "her" or "she." If the sentence still makes sense, then I meant to include both versions.

Ulrich Kortenkamp
Zürich, August–October 1999

Contents

List of Figures

Zusammenfassung

Diese Arbeit behandelt die mathematischen, informationstechnischen und didaktischen Grundlagen der Dynamischen Geometrie.

Bisher gab es kein Computerprogramm, welches die dynamische Manipulation von zweidimensionalen Konstruktionen mathematisch zufriedenstellend gelöst hätte. Die Umsetzung des Poncelet'schen Kontinuitätsprinzips ist nur in der reinen Mathematik gelungen: der Grundsatz, dass mathematische Eigenschaften auch dann erhalten bleiben müssen, wenn sie nicht sichtbar sind, blieb auf der Strecke, obwohl er in der Geometrie geboren wurde. Es geht gar so weit, dass die Gültigkeitsbereiche von Konstruktionen untersucht werden, die aber eben nicht durch die Mathematik vorgegeben werden, sondern durch die jeweils eingesetzte Software.

In dieser Arbeit versuche ich, die Mathematik und die Informatik wieder ein Stück näher zusammen zu bringen. Im ersten, mathematischen Teil, wird anhand von konkreten Beispielen untersucht, woher die merkwürdigen, unerwarteten Effekte in Geometrieprogrammen kommen, wieso manchmal – scheinbar unmotiviert – Elemente verschwinden oder über grosse Entfernungen springen. Es wird die Grenze gesucht zwischen trivialer Implementation und den echten mathematischen Herausforderungen: Welche Konstruktionen erzeugen die Probleme? Was ist noch einfach?

Zunächst muss dafür ein formaler Rahmen für Dynamische Geometrie definiert werden, innerhalb dessen klar formuliert werden kann, welche Eigenschaften ein Geometriesystem hat. An Punkten und Geraden wird gezeigt, wie mit homogenen Koordinaten, Projektiver Geometrie und Cayley-Klein Geometrien ein in sich geschlossenes System gebildet wird, welches sowohl deterministisch ist als auch kontinuierliches Verhalten zeigt. Zudem können Inzidenzsätze innerhalb dieses Systems einfach mit Methoden der Randomisierung bewiesen werden.

Dann wird der Versuch gestartet, auch Kreise (oder Kegelschnitte) in das System einzugliedern. Eine Hauptaussage wird sein, dass wir dann nicht erwarten können, ein deterministisches *und* kontinuierliches System zu erhalten. Die Frage, ob es überhaupt möglich sei, kontinuierliches Verhalten zu erzeugen, – schon allein dadurch berechtigt, dass es bislang keine Geometriesoftware gab, die dieses zeigte – kann dann aber schliesslich positiv beantwortet werden. Der Weg zu einem kontinuierlichen System führt wieder zu den Methoden, die sich aus der Geometrie hinaus gebildet haben, und die Lösung des Kontinuitätsproblems der Dynamischen Geometrie liegt in der Zuordnung geeigneter Riemannscher Flächen zu den Konstruktionselementen.

Die konkrete Anwendung der Theorie komplexer Funktionen erfordert aber auch eine Umsetzung auf rechnerischer Ebene. Diese wird im zweiten Teil der Arbeit behandelt. Einige der Implementations-Details wie sie der Geometriesoftware *Cinderella* zugrunde liegen werden vorgestellt, und auf die Unterschiede und neuen Probleme im Gegensatz zu "klassischen" Geometrieprogrammen hingewiesen. Zusätzlich wird darauf eingegangen, wie die neuen Möglichkeiten des weltumspannenden Internets durch den Einsatz der Sprache Java genutzt werden können, welche Probleme dabei auftreten, und wie sie gelöst werden können.

Im dritten Teil der Arbeit gehe ich darauf ein, wieso selbst für schulmathematische Zwecke ein derart komplexes mathematisches Fundament von Nöten ist. Die Notwendigkeit der korrekten mathematischen Behandlung von Geometrie auf dem Rechner sowie der flexiblen, modularen und möglichst allgemeinen Programmierung wird am Beispiel des kreativitätsbildenden Einsatzes des Computers im Mathematikunterricht diskutiert. Als konkretes neues Einsatzgebiet dynamischer Geometriesoftware kann, basierend auf den mathematischen Methoden des ersten Teils und der Anbindung an das Internet, die im zweiten Teil beschrieben wurde, die Erstellung interaktiver Geometrie-Arbeitsblätter genannt werden. Diese können von Schülern und Studenten online bearbeitet werden, und das Programm gibt selbstständig Hilfestellungen und überprüft die Lösung. Dabei wird der Lösungsweg nicht starr vorgeschrieben, sondern kann durchaus von der vorgegeben Lösung abweichen. Ohne den im mathematischen Teil eingeführten Begriff des geometrischen Satzes wäre dieses automatische Überprüfen von Lösungsvorschlägen gar nicht möglich, es wäre noch nicht einmal klar, was es überhaupt heisst, dass eine Konstruktion das korrekte Ergebnis liefert.

Ausserhalb der zweidimensionalen computergestützten Geometrie gibt es noch weitere Anwendungsfelder, beispielsweise parametrisches CAD, computergestütze Kinematik oder auch das ganze Gebiet der "virtual reality," die mit dem gleichen oder ähnlichen Methoden bearbeitet werden können. Diesen Anwendungen widmet sich der letzte Teil, in dem auch offene und weiterführende Fragen angesprochen werden.

Abstract

This thesis is about the foundations of Dynamic Geometry with respect to mathematics, computer science and education.

Up to today there was no software which could handle dynamic manipulation of two-dimensional constructions in a mathematically satisfying way. Only in the pure mathematics the principle of continuity of Poncelet, stating that any mathematical property must remain valid even if it is not visible, has found its place; geometry, which was its birthplace, went on untouched. Even worse, nowadays the domains of validity of constructions are examined, which are not caused by mathematics, but by specific implementations within the software used.

This thesis is trying to close the gap between mathematics and computer science a little bit. The first, mathematical part, will explore the strange and unexpected effects in geometry software using concrete examples. It will be asked, why sometimes – seemingly without reason – elements disappear or jump from one position to the other. The border between trivial implementation and real mathematical challenges will be made visible: Which constructions are causing the problems? And what is still easy to do?

At first there must be a formal framework for Dynamic Geometry, which will enable us to speak about the properties of a geometry system. Using points and lines as a test example we will show how homogeneous coordinates, Projective Geometry and Cayley-Klein geometries work together as a closed system. This system is determined by its input elements and continuous at the same time. In addition, it is easily possible to prove incidence theorems with methods of randomization within that system.

Next we will try to add circles (or conics) to this system. A main point will be that we cannot expect to get a system that is uniquely determined by the input *and* continuous. The question whether it is possible at all to get continuous behavior – a reasonable question, since there was no continuous geometry software up to now – can be answered positively in the end. Going back to the methods that were once coming from Geometry we will create a continuous system. The solution to the problem of continuity in Dynamic Geometry will be based on assigning suitable Riemann surfaces to the construction elements.

The application of the theory of complex functions requires also a transfer to computational methods. We will address this in the second part of the thesis. Some of the implementation details of the geometry software *Cinderella* will be presented. The differences and new problems in contrast to the "classic" geometry packages will be explained.

The new possibilities of the world wide connecting Internet can be accessed by using Java as the implementation language, and the problems that are caused by this choice as well as the corresponding solutions are discussed.

In the third part I will explain why even for school-level geometry such a complex mathematical foundation is necessary. The importance of a correct mathematical treatment of geometry on a computer as well as a flexible, modular and most general programming will be shown using the process of building creativity with the help of the computer in mathematics classes. As a concrete and new application of geometry software, which is based on the mathematical methods of the first part and its connection with the Internet as described in part two, I will show interactive exercises. These can be done online, and the software will give context-sensitive hints to the solution automatically, and it will also check whether a solution is correct. Here the path to the solution is not fixed, but can differ from the given solution. Without the notion of geometric theorems as developed in the mathematical part this automatic checking of solutions would not be possible; moreover, it would not be clear at all what it means for a construction to give a correct result.

Apart from two-dimensional computer based geometry there are several other areas where the same or similar methods can be applied; parametric CAD, computational kinematics or the whole field of virtual reality, just to mention a few. These applications are the topic of the last part, which is also devoted to open and continuing questions.

Chapter 1

What is Dynamic Geometry?

Twenty-five years ago computers were not at all suited for visual manipulation of geometry. They were lacking the high-resolution graphics devices we can find today in all computers, they did not have easy-to-use input devices like mice, some were even fed with punch-cards still, and their memory and CPU could not handle large amounts of data in the blazing speed we became used to today. Also, there were much more basic problems in data processing that had to be solved before – the book series "The Art of Computer Programming" by Donald Knuth [43, 44, 45] was still brand new (and everybody was sure that the next volume will be available soon, which has not changed since then), it was just a few years ago that the revolutionary programming language Pascal had been introduced by Niklaus Wirth [88], algorithms like QuickSort [45], which is now taught to every computer science student, were still exciting.

In these twenty-five years a lot has changed. Computers are on every desktop, 8-years old seem to know more than their teachers about it, high-speed 3D-rendering is affordable and any good computer game needs it. You can buy personal digital assistants that fit in your pocket and outperform the big computers of the 70's by far. The computers talk to each other via phone or cable lines, everybody has email and a homepage, the Internet is the revolutionary new medium at the end of the millennium. The world of computers has changed from the black/green character-based terminals that only gurus could handle to the multi-colored streaming video virtual reality for everybody.

This radical change has opened new areas for the use of computers. It is not surprising that during the 80's a new discipline called "Dynamic Geometry" had appeared, that at first tried to use computers as a ruler and compass replacement. That is, using a mouse and a high-resolution display you can draw lines and circles, use their intersections and make a printout of your drawing. A first benefit of using a computer here is the increased accuracy of the drawing – you will be able to use the exact intersection of two lines, even if you are not very skilled in drawing. But the important enhancement with respect to the old tools ruler and compass is that the computer can record the way you *constructed* lines, points and circles, and the software is able to quickly redo the construction after you changed some parameter. This is the key to interactivity: Grab a point, move it and

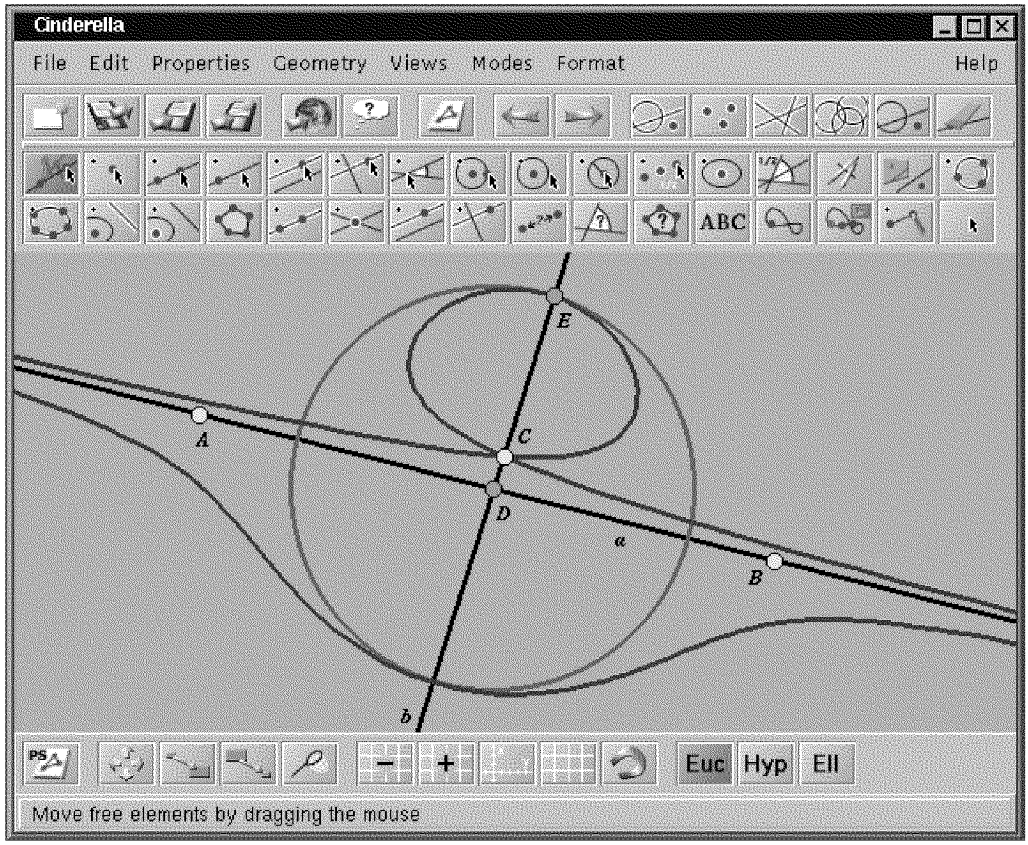


Figure 1.1: A typical session with a Dynamic Geometry software. The user has a toolbar with different tools like "draw a line" (ruler), "draw a circle" (compass) which he chooses to perform certain tasks. Below is a drawing surface where these tasks are carried out, comparable a sheet of paper.
The screenshot shows a construction of a conchoidal curve, which can be done with a marked ruler (see [21]).

see immediately how the construction changes.

This *dragging* capability is the fundamental improvement versus the mechanical devices ruler and compass. Most times when you start a construction from scratch, you have no control over the intermediate construction elements, and it might *(will!)* happen that some parts of the construction are clustered or lie outside the drawing surface. If you have the chance to move free objects while you construct or after you are done you can avoid these effects.

Much more important is the fact that you can explore the *dynamic behavior* of a construction by moving it. You can see what parts of the construction change and which remain the same. You get by far more insight into this particular construction and geometry in general if you can experience what happens under movements. A more sophisticated software will also let you give another dimension of understanding by supporting loci, the

traces of objects under movement of other objects, that are adjusted dynamically as well.

The first software packages that could be used for Dynamic Geometry were Geometer's Sketchpad [36], which appeared first in 1989, and Cabri Géomètre [54], dating back to 1988, and they started another revolution: Computers could be used in school for teaching geometry, and since then a lot of work has been done that discusses the many aspects of using Dynamic Geometry Software in education. But the software since then has almost remained the same, the latest versions of both Sketchpad and Cabri were released in 1995 – it seems like there has been no real substantial progress for ten years. Of course, there were minor improvements with respect to user interface design or other new features like conics in Cabri II. But the core of Dynamic Geometry hasn't changed since. Today, there are more than 40 packages for Dynamic Geometry, but none of them has gone other than the straight-forward way of implementing Dynamic Geometry. So the quality of Dynamic Geometry Software today seems to be just a question of the user interface, the educational support or – the price.

This would not be a pity if the core of Dynamic Geometry were a completely understood matter. But, unfortunately, this is not the case. Jean-Marie Laborde, the main designer of Cabri Géomètre, pointed out the need of a true mathematical foundation of Dynamic Geometry [52]:

> "I think we need a real mathematical treatment of all consequences of stretching geometry in some way to a wider system. This system cannot be simply the projective one if we want to maximize the way the environment takes into account the special characteristics of non-static objects which are at the core of Dynamic Geometry."

"Where's the problem?" you might ask. "Ambiguities!" is the answer. While a construction is done the user is responsible to resolve ambiguities that arise from operations like "intersection of a circle and line" or "angular bisector." For both constructions there are several possibilities, a circle and a line have two intersections (if they intersect properly), two lines define two angular bisectors (which are perpendicular to each other). If you work on screen with a mouse you can decide visually which of the two solutions you prefer.

But already when you use a keyboard interface, a textual description of a construction, you cannot give enough information to the computer by just specifying the construction steps. You will have to add additional information, like "take the intersection of line a and circle *c that is closer to line b*." This additional information cannot, and this is crucial, be generated automatically in general.

The real problem arises when the user interacts with a construction and moves base points. The construction shall be re-done by the computer, and for this the construction has to be stored in some format in the computer's memory. Now each time a base element has been moved the decisions that were made by the user for the first figure have to be made by the computer. Not arbitrarily, but – this would be perfect – the same decisions the user would make if we could ask him again.

How should a construction behave under movements? The most natural thing would be a continuous movement of dependent elements: When a free element is moved only a little bit, then the dependent elements will move only a little bit, too. We do not want elements to "jump around" wildly.

This is the core problem of Dynamic Geometry: *To find a well-defined method for handling ambiguities while some parameters of a construction are changed continuously.*

The same questions arise in other contexts: An example for an area that is understood very well are analytic functions on Riemann surfaces. If you regard a function as a "construction sequence" consisting of basic operations – addition, multiplication, square roots, etc. – you have to handle ambiguities. There is not one square root of a real number, there are two, possibly complex ones. If we denote the the solution of the equation $y^2 = x$ by $\sqrt{x}$, an expression like

$$f(x) = \sqrt{x - \sqrt{x}} \tag{1.1}$$

is ambiguous, there are two sign decisions that have to be made.

For a certain value of x we can give the decisions explicitly: For example, let $x = 4$, then the inner root shall be -2 and the value $f(4)$ shall be $+\sqrt{6}$, the square root of six that is larger than zero. Or let $x = 9$, let the inner root be 3, and the value $f(9)$ shall be again $+\sqrt{6}$. Were these decisions consistent with the ones at $x = 4$?

If we really want to create a *function* $f(x)$ that is continuous, we must be able to find the right, consistent decisions for any x, and we have to extend the definition space of f. Using analytic continuations we can create a continuous function on a Riemann surface that is a four-cover of complex space in this example. In this thesis this concept will be extended to cover ambiguous geometric constructions, and in fact we will find a lot of parallels between the theory of complex analysis and Dynamic Geometry.

The similarities do not lead to a one-way transfer of knowledge from complex analysis to Dynamic Geometry. There are several open questions in Dynamic Geometry which were not investigated until now in their complex analysis counterpart (as far as I know): for instance, how hard is it to decide whether there is a continuous path from one instance (evaluation) of a complex function to another one, or "Can you get lost on Riemann surface?"

It is not only the geometry part that I want to cover with this thesis:

> *Dynamic Geometry is the theory of construction-like descriptions of function-like objects under parameter changes.*

When talking about Dynamic Geometry, it is useful to think geometrically, because this provides the necessary intuition, but one should always keep in mind that this way of thinking might be applicable to other problems as well.

Chapter 2

The Interactive Geometry Software *Cinderella*

This thesis is based on the research work that was done while Jürgen Richter-Gebert and I were writing what is now called *The Interactive Geometry Software Cinderella* [71, 70]. In this chapter I want to describe the software phenomenologically without going into details. Many of these details will be presented in the other parts of the thesis. Unfortunately, it was not possible to cover every detail that was needed for the implementation in this thesis, since it would have created a book of several hundred pages. Nevertheless, it was a symbiotic process that lead to the theory and the implementation: the one could not have been without the other. Many of the little tricks that went into the source code just work because we can make certain mathematical assumptions, and without these tricks, the software would be slow and therefore unusable. Because of these connection I would like to consider the implementation part of this thesis, even if it is not included. You will find some of the more important implementation issues in Sec. 9.2.

2.1 Basic Functionality

Cinderella is a software package for doing geometry on a computer. In a way it replaces pencil, paper, ruler and compass with equivalent computer tools. With the mouse you "draw" in a window on screen, i.e. you place points, connect them by lines, erect perpendiculars, etc.

This functionality is not very exciting, but already useful if you want to do exact constructions. You can be sure that you exactly – only restricted by the floating point accuracy of the computer – hit an intersection point of two lines, or draw an exact parallel to some other line. Also some constructions that are tedious to do by hand are easily done with the computer, for example inversions at a circle (or conic). The additional possibility of rescaling a figure can help you if the construction you are doing exceeds the limits of the window.

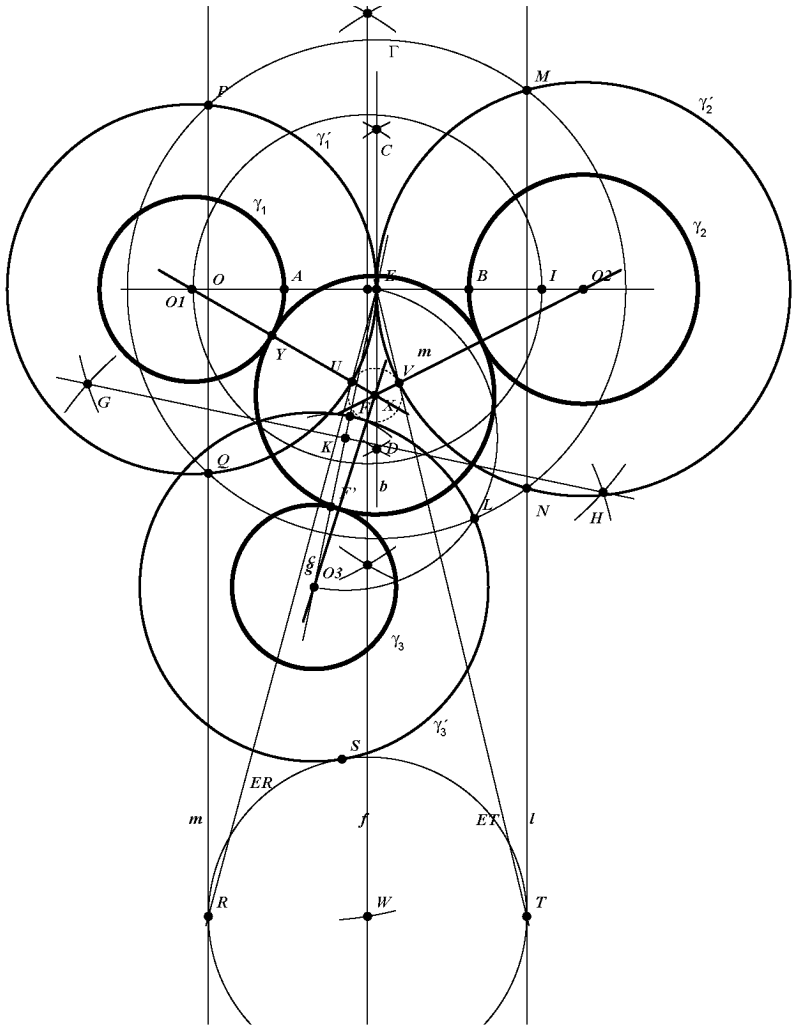


Figure 2.1: An example for a construction that is easy with *Cinderella*, but hard to do with ruler and compass on a sheet of paper. This construction is due to Appolonius. It solves the problem to find a circle that is tangent to three given circles.

The important distinction of *Cinderella* to other drawing software – Corel Draw, Microsoft Word or others – is that it keeps track of your construction steps and is able to re-do them, even for a different placement of the base elements. This happens instantly and interactive, so you can pick a point and drag it to some other position, and the rest of construction is updated during the move. This looks and feels like having build the construction from matter.

This feature, which is characteristic for interactive geometry software, is very useful while you do the construction: You can adjust it in order to avoid crowded parts of a drawing, to make it look nicer, or to have access to elements that lie outside the drawing region. But is even more useful when you have finished the construction, since you can explore the geometric properties of it. "What happens if?" is a question that can be answered with dynamic geometry software, and, even better, you can build intuition and feeling for geometry using the drag-mode of interactive geometry software.

Cinderella supports points, lines, circles, arbitrary conics, segments, polygons, measured distances and angles as well as loci. Among the basic constructions are intersections of lines and conics, parallels and perpendiculars, angular bisectors, circles defined by their centers and a point on the circle or by three points, and conics defined by five points.

2.2 Main Features

Besides the basic characteristic of interactive geometry software, the ability to move points and lines while the constructions is immediately updated, *Cinderella* has several unique traits, which we want to emphasize. Most of these have only been possible due to the mathematical foundation of *Cinderella* – we just note this to stress the importance of a profound mathematical background.

2.2.1 Multiple, different, simultaneous views

With *Cinderella*, you can view a single construction in different geometrical interpretations at the same time. Internally, an abstract model of the construction is maintained, which can be displayed through one or more windows – or *viewports*, as we call them. These come in many different flavors, and each of it has its own strengths.

Euclidean plane

The usual viewport, which is also the default viewport at startup, is the *Euclidean plane*. Of course, this is not really the mathematical Euclidean plane, but computer display version of it. This is the default viewport, since most people are used to it and want to work in it. The pixels on the screen are mapped to a rectangular part of the Euclidean plane embedding of the projective plane (which is the abstract world of the Kernel, see Ch. 5 and 6), or vice-versa. Lines look like lines, circles look like circles.

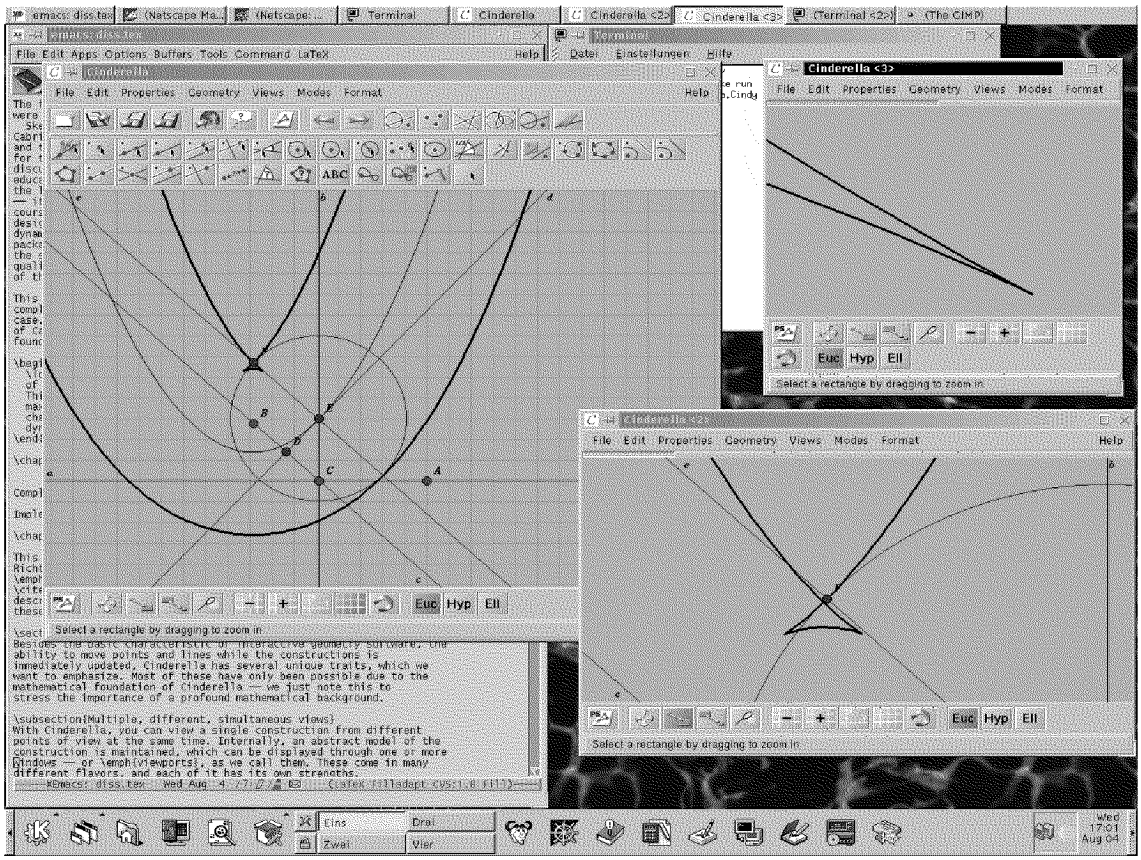


Figure 2.2: A *Cinderella* session with three different Euclidean viewports. The construction shows an offset curve of an parabola, which is a challenge for every CAD system. The window in the lower right of the screen shows the two internal cusps of the offset curve, the window in the upper right shows a detail view of the right cusp.

You can choose the section of the Euclidean plane you want to see by zooming in or out and by translating. Just imagine you have a variable-sized rectangle you can put on the (abstract) Euclidean plane, and everything inside that rectangle will be displayed on screen.

Even if you only had this single kind of viewport, it would make sense to have multiple, simultaneous copies. A possible scenario: You want to explore a locus – an algebraic curve created as the trace of an object – that has an interesting bend. You want to zoom into the construction, since you cannot see whether there is a cusp at that position or not. At the same time you want to have a global overview over the construction, because you want to move points to see how the locus changes at that position. With *Cinderella* you can open a second (or third, ...) Euclidean viewport to have several views at the same time. You move points in either of the views, and you get an immediate visual feedback in all of them, see Fig. 2.2.

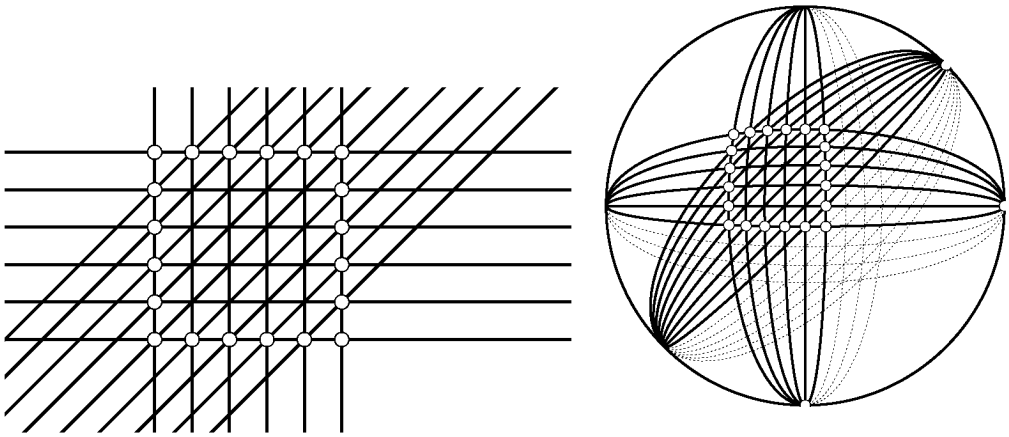


Figure 2.3: Three parallel bundles, on the left in a Euclidean viewport, on the right in the spherical projection. The dashed lines show the antipodal elements on the back hemisphere. The equator of the sphere is the circle in the two dimensional projection; the three parallel bundles meet in three different points on the equator.

Spherical Projection

Inside *Cinderella* the Euclidean plane is augmented by a "line at infinity," which embeds the plane into a space called the projective plane. This projective plane can be visualized as a double-cover of the unit sphere in three dimensional space: Points are mapped to pairs of antipodal points on the sphere, and lines are mapped to great-circles around the sphere. The map is given by a central projection of the plane located at $z = 1$ in 3-space through the origin onto the unit sphere. The "north pole" of the sphere touches the plane. The equator of the sphere does not correspond to any point of the Euclidean plane, instead we can find the "points at infinity" there. See Sec. 5.1.1 for a detailed description of this embedding.

We can use the spherical projection viewport to get a better understanding of the concept of infinity. Most people have heard that parallel lines meet at infinity, but they could never verify this by experience, or even imagine it. In *Cinderella* you can open a spherical view and move a point straight to the equator, and you can see that all lines meeting this point become parallel in the Euclidean viewport. The position on the equator determines the direction of the parallel bundle (Fig. 2.3).

While it is very instructive to explain infinity to somebody using spherical viewports, its applications go far beyond. For example, you can study the behavior of a locus curve at infinity: Take the offset parabola of Fig. 2.2. In the spherical port you can see that the parabola touches the line at infinity, and the offset curve touches at the same point (Fig. 2.4. When you animate the construction you can see how the offset point C changes from one side of the parabola to the other.

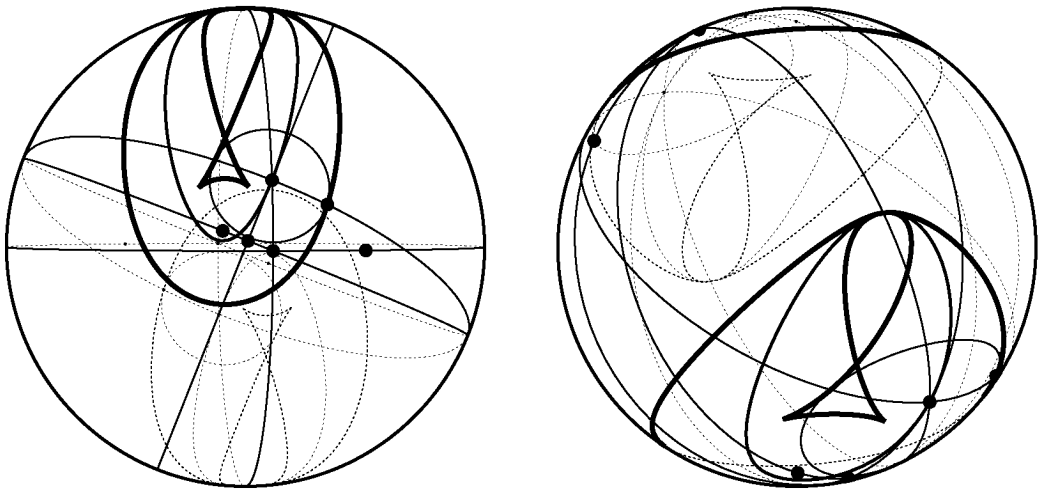


Figure 2.4: The left picture shows the offset parabola in a spherical port with the line at infinity on the boundary of the circle. The right picture shows the same situation after rotating the sphere.

Polar viewports

As explained in Sect. 5.1.1, the roles of points and lines in Projective Geometry can be interchanged. Instead of a line connecting two points you have an intersection point of two lines, and so on. This is called the dual or polar, the original situation is called primal. *Cinderella* can automatically display a construction in a polar Euclidean or polar spherical viewport. This is very useful, for example, when you try to prove a theorem: Sometimes the polar version is much easier to understand and prove.

The polar view is not restricted to points and lines, also conics and loci are supported. The polar of a conic C is the envelope of all dual lines of points that lie on C, it is again a conic, and it can be easily calculated by taking the adjoint of the matrix of C. The locus is just the locus of the polar object.

Poincaré Disc Model of Hyperbolic Geometry

Hyperbolic geometry is probably the most famous non-Euclidean geometry. Several drawings of M.C. Escher [13] show tilings of a circle that become finer and finer to the boundary. These drawings use the Poincaré-model of the hyperbolic plane: The fundamental object, which plays the role of infinity, is the outer circle. Since it is at infinity, we cannot reach it by making a finite number of steps of some constant distance. This is shown in Fig. 2.6: You are trapped inside the circle if you allow only finite hyperbolic distances.

The hyperbolic plane is a wonderful playground for geometric explorations. You can check, for example, how your favorite theorem looks like in the hyperbolic plane. If you

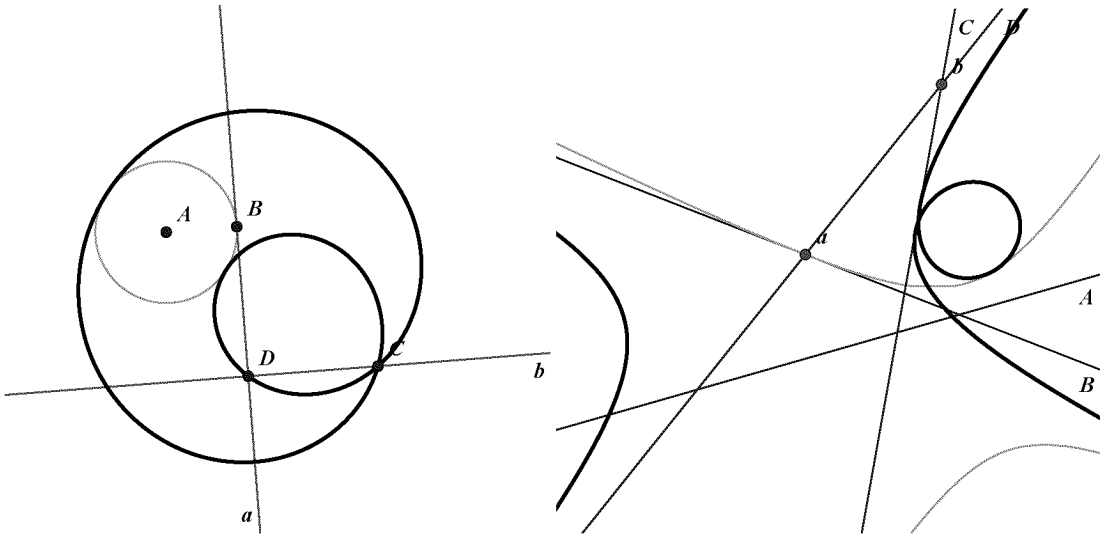


Figure 2.5: On the left the primal picture of a cardiographic curve, on the right the same construction in a polar view. Observe that the dual of the circle is a hyperbola. There is no way of recognizing polar Euclidean circles immediately.

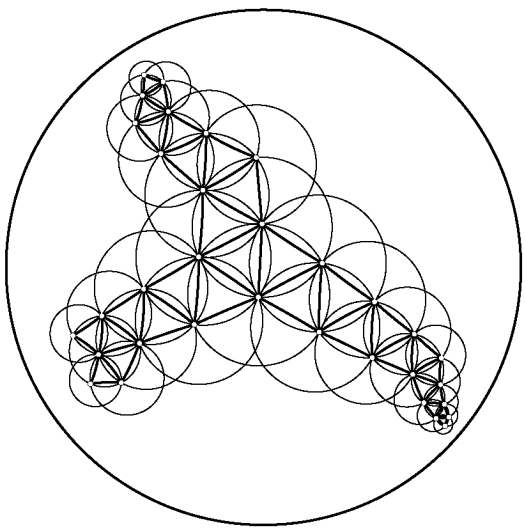
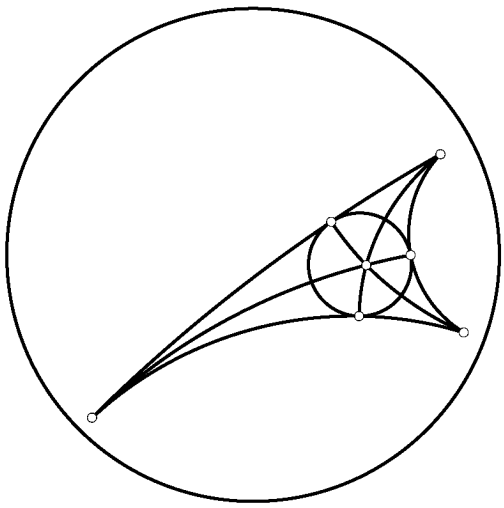


Figure 2.6: Try to reach the boundary of the Poincaré view by making hyperbolic unit steps – you will never succeed. The circle is to the hyperbolic plane what the line at infinity is to the Euclidean plane.



A	Point(0.93|−0.34)	(0.93|−0.34)
B	Point(0.87|0.47)	(0.87|0.47)
a	Join(A;B)	y = −12.89x + 11.71
C	Point(−0.70|−0.71)	(−0.70|−0.71)
b	Join(B;C)	y = 0.75x − 0.18
c	Join(C;A)	y = 0.22x − 0.55
d	AngleBisector(a;b;B)	y = 4.70x − 3.63
e	AngleBisector(b;c;C)	y = 0.44x − 0.40
f	AngleBisector(a;c;A)	y = −1.56x + 1.11
D	Meet(d;e)	(0.76|−0.07)
g	Perpendicular(c;D)	y = 4.93x − 3.80
h	Perpendicular(b;D)	y = −1.62x + 1.16
k	Perpendicular(a;D)	y = 0.45x − 0.41
E	Meet(c;g)	(0.69|−0.4)
C0	Circle(D;E)	2.53x² + 1.26y² − ...
F	Meet(b;h)	(0.57|0.24)
G	Meet(a;k)	(0.91|0)

Figure 2.7: The construction of an incircle shown in the hyperbolic Poincaré viewport. Lines are shown as circular arcs that are perpendicular on the boundary circle. The construction uses hyperbolic measurements and shows an hyperbolic inscribed circle. On the right the construction text view.

can do hyperbolic measurements, you can check which constructions of the Euclidean plane are still valid, and which of them fail. Did you know that two hyperbolic circles may have 4 intersections? What does this mean for constructions that use circle/circle intersections? These and other questions take you to the roots of geometry.

Other geometry software emulates the Poincaré disc model with construction macros for circles in a Euclidean view. A big advantage of a built-in view is its speed, its completeness and its correctness. Especially completeness is hard to achieve with macro-based approaches, since this would mean to provide macros that work on loci – which no Dynamic Geometry software so far can do. The correctness is just a matter of correct macro constructions, but this seems to be not as easy as one might think: For example, the macro should still work for base elements that lie outside the fundamental conic, "behind infinity."

Construction Text

A special viewport is the textual description of the construction. A list of all elements, together with their definitions and current coordinates can be used for information purposes, but also as an easy way to do exact selections. Whenever you have to select an element, you can just click on the corresponding entry in the construction text. This becomes a necessary tool in highly degenerate situations.

Internally, the construction viewport uses the same display mechanism as any other view, so it is also immediately updated whenever you change something in any other port.

See Sec. 9.1.4 for a detailed description of the underlying architecture.

2.2.2 Complex Numbers

Cinderella uses complex numbers for all coordinates and values. Intermediate construction elements may become complex, but they usually do not disappear (this might happen in degenerate positions only). The construction is carried out until the end with these imaginary elements, and if a result is again real, then it will be displayed.

A particularly nice example of why this is useful is the *radical axis* of two circles. The radical axis is a line that is perpendicular to the line connecting the midpoints of the two circles, and which lies between the midpoints cutting their joining segment s into two parts at a certain ratio defined by the distance and the radii of the circles. For equally-sized circles the radical axis happens to be the perpendicular bisector of the segment s.

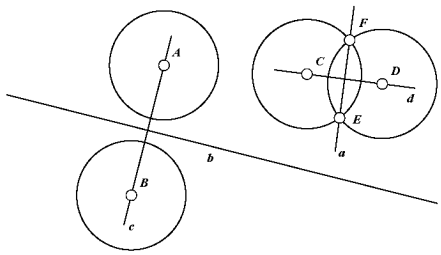


Figure 2.8: The radical axes of two pairs of circles of equal radii. They always bisect the segment joining the centers of the circles, even if the intersections defining the radical axis become complex, as shown on the left.

When the two circles intersect, then the radical axis meets the two intersection points. This means that we can define it also as the join of these two intersections. Usually, this construction is considered valid only if the circles intersect, but algebraically the above is also true when we move the two circles apart. The two intersections did not vanish, they became complex. They are defined by the two solutions of a polynomial of degree two with real coefficients, thus they are complex conjugates and their join is again a real line with real-valued coordinates.

This feature is sometimes confusing for the user, especially in a classroom environment. See Sec. 10.1 for some thoughts about this.

2.2.3 Cayley-Klein Geometries

In the viewport section we already mentioned hyperbolic measurements. *Cinderella* offers three different measurements, the usual Euclidean distances and angles, elliptic measurements, which occur when you want to measure distances and angles on a sphere, and hyperbolic measurements.

You can use any of the measurements in any of the viewports, there is no direct connection between displaying and measuring. Of course, it often makes more sense to use the hyperbolic measurement in the hyperbolic viewport, etc. For example, the Poincaré disc model is coherent with hyperbolic angle measurements, the Euclidean angles you measure between the circular arcs are the same as the hyperbolic angles between the lines.

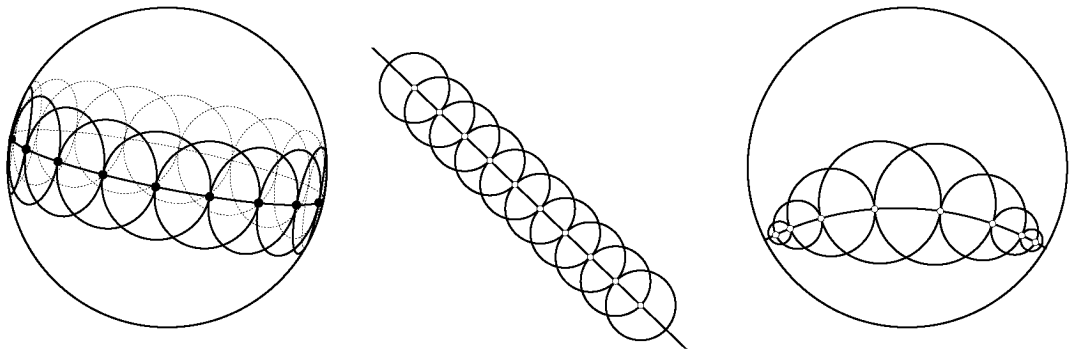


Figure 2.9: Circles in the three types of geometry that *Cinderella* supports. On the left the elliptic geometry in a spherical view, in the middle the usual Euclidean metric in a Euclidean view, and on the right hyperbolic circles in a Poincaré view.

The implementation of different measurements in *Cinderella* is very general. As explained in Sec. 5.4, the measure depends on the choice of a pair of fundamental conics. When you switch between different geometries in *Cinderella*, you do only exchange the fundamental object, and this creates the right measure automatically.

2.2.4 No Jumping Elements

The most important contribution of *Cinderella* is its handling of continuity. It will never happen that a small move of a free object will cause a sudden jump of a dependent object. This is a fundamental new feature compared to traditional geometry software, where this kind of discontinuity can happen all the time.

Usually jumping elements are caused by the intrinsic ambiguities of conic/line and conic/conic intersections, where two or four intersection points have to be assigned the right names. *Cinderella* does these assignments based on nearness conditions, and includes a concept to handle singular (degenerate) situations, that occur for example in tangent cases. Chapter 6 and 7 are explaining the underlying mathematical foundation in depth.

2.2.5 Automatic Theorem Checking

Based on the continuity of elements a randomized theorem checker has been built into *Cinderella*. The theorem checker is surveying all the construction steps done by the user and reports all non-trivial incidence theorems it finds (see Fig. 2.10). It is also used for the correctness checking of interactive exercises, see below, and to keep the internal data structures consistent.

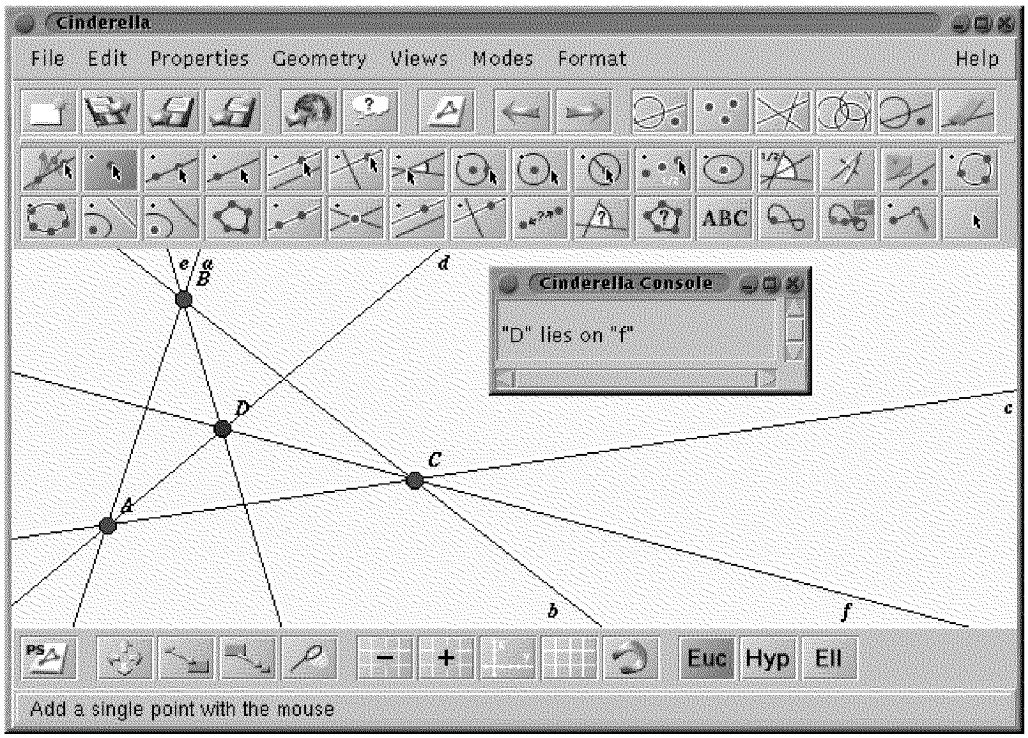


Figure 2.10: The theorem checker at work. The checker automatically discovered that the point of intersection of d and e lies on f, too.

2.2.6 Self-exploring Loci

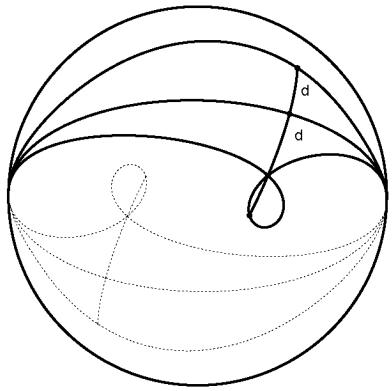


Figure 2.11: A very interesting and computationally challenging locus is the conchoidal curve, that is created by rotating a line around a point and marking the distance *d* on that line from another line.

A locus is the trace of an object under movement of some other object. The first object should depend on the second, otherwise the locus is pretty boring. *Cinderella* includes automatic loci, that it tries to display as good and correct as possible. Unfortunately, correct locus calculations are very time consuming and it is not always possible to spend all the time you would need for a perfect display. The locus display routines are still the best currently available ones. *Cinderella* uses a sophisticated speedup-slowdown algorithm to find good base points for the calculation. Observe that the speed of the object to be traced is some, not explicitly given, function of the speed of the moving element. We must try to find the inverse of that function and let the moving element move with that speed.

Other problems with loci are their behavior at infinity and how to detect which parts of the locus are off screen. Although *Cinderella* can display many loci without problems, the easiest way to crash *Cinderella* is probably to draw a locus. You can read about the implementation of loci in Sec. 9.4, and how you can use them in education is described in Sec. 10.4.

2.2.7 High Quality Postscript Output

Cinderella was used to create the figures in publications from its very beginning, so we took care to ensure a very high quality of the constructions for printing. The printing mechanism built into Java did not meet our quality requirements, since its positioning resolution is at approximately 72 dpi, less than most computer displays offer today.

The best way to provide a platform independent high quality output was to include an export facility to encapsulated Postscript. The postscript code that is produced by *Cinderella* is readable and editable (if you know postscript a little). Most of the figures in this thesis have been done with *Cinderella*, of course. It is currently not possible to include LaTeX or TeX code as labels or texts (as it is the case with IPE [8]), but a workaround is using the psfrag-package for LaTeX which permits replacement of postscript text with any LaTeX code, including typeset math.

2.2.8 Easy WWW-Export

It was easy to add an export constructions to interactive web pages, since *Cinderella* is written in 100% Java. These web pages can be accessed using any recent browser like Internet Explorer 5 or Netscape 4.5, and they offer the full interactivity of the move mode

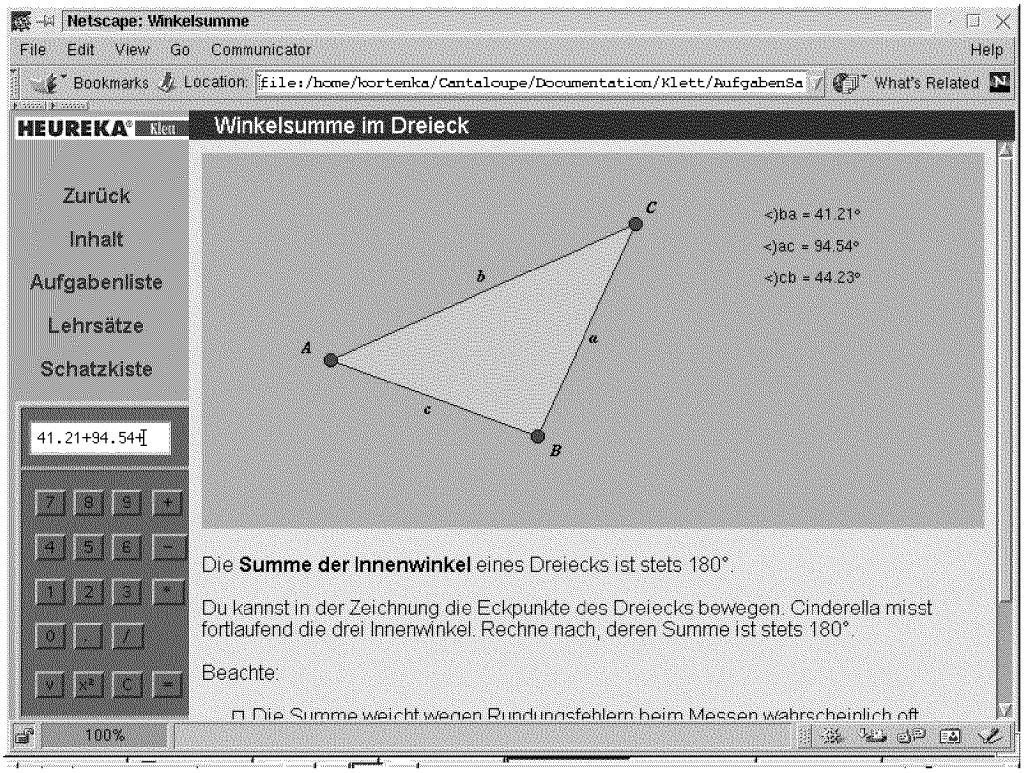


Figure 2.12: An example taken from the educational material included with the German school edition of *Cinderella* [70]. The student shall check whether the sum of inner angles in a triangle is 180 degrees, using the calculator on the left. The calculator is a Javascript object embedded into the HTML code of the page.

or automatic animations, without requiring the standalone version of *Cinderella*. The runtime libraries necessary may be freely distributed. This makes *Cinderella* a handy tool for the preparation of distance learning material.

The export itself is as easy as could be. Once you have created a construction, you can save the necessary HTML code that is used to display the *Cinderella* applet. No further knowledge of web page creation is necessary.

The web export was also used to create the additional educational material in [70]. The use of portable standards makes it possible to create cross-platform multimedia products while still offering extendibility; for instance an external Javascript-based calculator was included in some web pages, which with a different approach would have had to be integrated into *Cinderella*, see Fig. 2.12.

The web pages created with *Cinderella* can be included on CD-ROM or other media or can be downloaded via the web. The applications range from interactive examples created by teachers for their students and distributed on floppy disks up to whole educational websites devoted to geometry that cover all topics, starting with elementary geometry and ending at the research level. Fun examples or geometry puzzles are other areas where

Cinderella can be used for improved interactive presentation. Even the demo version of *Cinderella* that was first published in February 1999 has been used by several people for this [12, 73, 83], and a list of exemplary websites is maintained on the *Cinderella* website [46].

2.2.9 Interactive Exercises

The next level of interactive geometry is reached by combining the powerful theorem checking engine of *Cinderella* with the web export. The result are interactive geometry exercises with automatic solution checking done by the computer.

Suppose that in a sequence of geometry lessons you have taught how to do basic constructions like angular bisector or midpoint using ruler and compass only, and now you want to check whether the students can transfer these constructions to other situations. You design an assignment that combines some of the basic constructions. This approach has several drawbacks: It is very much work for you as a teacher to check all the different solutions the students offer. Good students might come up with "better" constructions as the one you had in mind, and you have to find out whether it is really a valid solution or whether it fails for some situations. Inexperienced students might come up with no solution at all, because they got stuck after the first few steps.

The interactive exercises created with *Cinderella* (Fig. 2.13) attack these problems by offering an automatic solution checking based on the built-in theorem checking, and an automatic hinting mechanism that guides students to the next step while still not restricting them to a particular construction sequences. This flexibility ensures the greatest freedom possible for the students while still helping them not to get lost.

It is not easy to design a good exercise, but since the exercises can be accessed using the Internet it is only a matter of month to create a database of high-quality exercises in a joint effort of teachers nation- or world-wide.

2.3 Availability

We started the project not only for our own purposes, but we always had in mind that the software should be available to a wide public. We chose not to give it away as an open source software for free, but to have it published like a book. This goal of a commercial-grade software was challenging and introduced much more work than an academic software project usually brings with it. We chose that way of distributing the software for several reasons. The most important reason of all is, that we would never have finished a version of *Cinderella* that is suitable for distribution if we did not sign a contract with somebody that forced us to do it. Another issue is that we could not afford to advertise the software, or to do the whole support once it is used by too many people.

The software is now available from Springer-Verlag [71], and comes with a 143 pp. manual. Another edition of the software will be available from HEUREKA-Klett Soft-

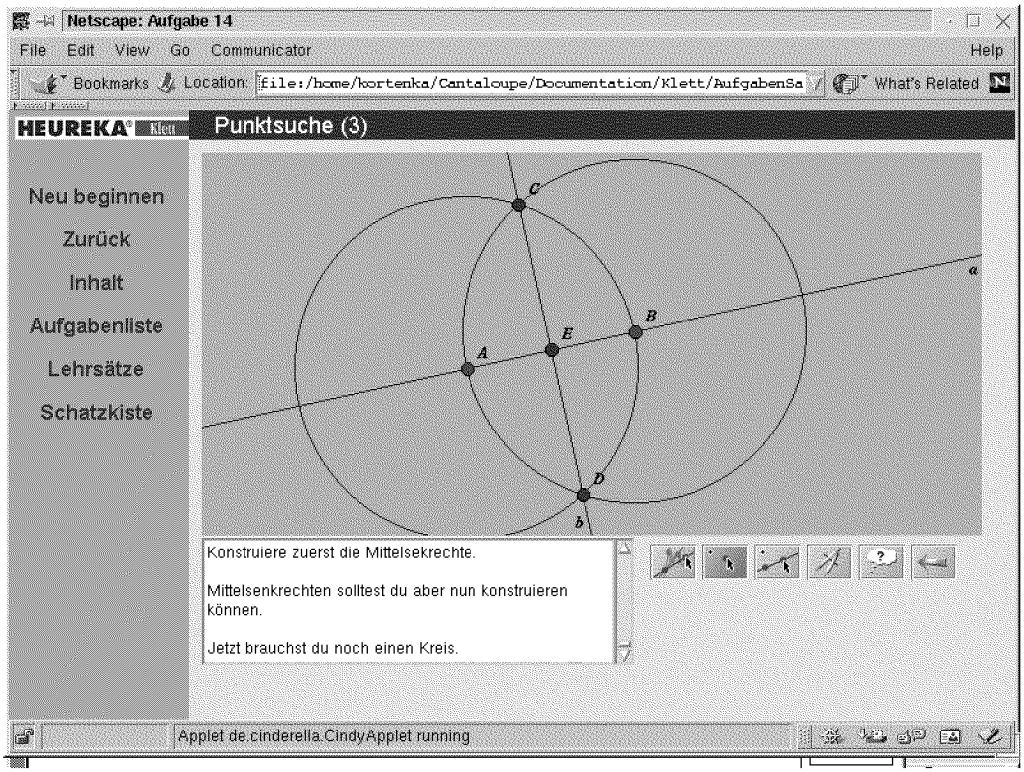


Figure 2.13: An example of an interactive Exercises, also taken from the German school edition of *Cinderella* [70]. On the lower left you can see the (German) hints and comments that *Cinderella* gave during the last construction steps, on the lower right you see the available tools for this exercise: Move, add a point, add a line, compass, hint, undo and restart.

wareverlag, Stuttgart. This edition focuses on the "after-school market" and comes bundled with self-learning material. This edition was a lot of extra work, since we had to fix several problems that are special to in-school usage.

Actually, both editions are based on the same source code, and we are still convinced that the mathematics both for a school version and a university version must be the same. The difference is only in the configuration of the user interface and the construction tools that are available.

The source code of *Cinderella* is not public for legal reasons, parts of it may be available on request for scientific purposes from Jürgen Richter-Gebert or myself.

Chapter 3

Site Map

This chapter gives you a rough outline of this thesis. Depending on your background and your interest you might choose different chapters to read, and you can make your choice using the short abstracts below.

This thesis is divided into three main parts, mathematics, computer science and education. The first part builds the theoretical foundation of Dynamic Geometry, the computer science part discuss the various problems and their solutions that arose when we tried to implement the mathematical theory. The third part covers several pedagogical issues that came up while and after we implemented *Cinderella*.

3.1 Mathematics

At first sight it is not clear where the mathematical difficulties lie in Dynamic Geometry. Most people think that high school mathematics should suffice to implement a Dynamic Geometry system that will be used in high school. One of the main goals of the mathematics part is not only to solve problems, but to demonstrate that there are problems.

3.1.1 A Framework for Dynamic Geometry

Conceptually probably the most important chapter, but for most people only of marginal interest. It defines the notion of a relational instruction set, a formalized way of describing construction steps that can be ambiguous, and introduces geometric straight-line programs, that use these relational instruction sets for describing complete constructions.

3.1.2 Projective Dynamic Geometry

As a warm-up we look at a particularly nice world of geometry. Using only points and lines in the projective plane, we look at the properties of this kind of Dynamic Geometry. It turns out that it is very well behaved – continuous, actually polynomial –, and we can do randomized theorem proving.

3.1.3 Circles and Conics

Heavily contrasting the previous chapter which presented a nice, well-behaved world of geometry, this chapter introduces all the problems that arise when we want to extend from points and lines to circles or even arbitrary conics. It will be shown that we cannot expect a system that is continuous and determined at the same time. And it will be shown that orientations cannot be used as an approach to continuity.

3.1.4 Complex Tracing

Here the main result is presented: Complex tracing, a technique that takes the history of a configuration into account. The main trick is that we carry out all calculations in complex space, and there we have all the powerful tools from complex analysis that we need to guarantee not only continuity, but even analytic behavior, as well as plenty of space to take detours around "bad" situations. In a last section we will scratch the topic of the computational complexity of complex tracing: Although complex tracing can be implemented as described in the previous chapter, it is not clear whether there exist better implementations with the same properties. Under continuity assumptions we can ask for the equivalence of or reachability between two instances of a construction. Can we decide this problem without having to trace?

3.2 Computer Science

This part covers some of the implementation specific details of *Cinderella*. It is serving as a proof of concept, showing that complex tracing can be used in geometry software.

3.2.1 Java-based Software

This chapter tells the history of the *Cinderella* project and explains why we decided to use Java as implementation language for the software. Some of the special issues that arise with platform independent software are discussed, and our approaches to solve these issues are presented.

3.2.2 Efficient Datastructures for Dynamic Geometry

A more flexible organization of construction data than the usual approach to implement Dynamic Geometry software in object-oriented languages is presented and its advantages over traditional concepts are pointed out. Although targeted at the mathematical results of the first part, these data structures are useful also for classical, non-tracing geometry software. Next it is shown how the theoretical results can be transformed into an algorithm for complex tracing. Some of the heuristics that make complex tracing and automatic

theorem checking applicable are explained, as well as the various additional uses of the automatic theorem checking engine for improved stability of the software.

3.3 Education

This thesis covers also some topics of mathematics and geometry education. This is not meant as a pedagogically stringent coverage of the topic, but as a starting point for in-depth educational research. The link between profound mathematics and better teaching shall be established.

3.3.1 Creativity in Math Education

A mathematically correct system can be used much better as a tool to raise creativity than a system which creates confusing or false output. Some of the possible stimulating uses are introduced and discussed, under the consideration of new ways of teaching using geometry software.

3.3.2 Geometry Education and the Internet

For most teachers the Internet integration of *Cinderella* is the most important new feature. In this chapter we will show some of the scenarios that are possible now and will be possible in the future using the capabilities of the network in conjunction with Internet-enabled software.

3.3.3 Future Developments

Many questions are still open and some new questions have been raised by this thesis. The last part covers these questions and intends to exhibit some of the next challenges – in mathematics, education, computer science and other areas – that are to be taken.

Chapter 4

A Framework for Dynamic Geometry

In this chapter we will fix a framework for Dynamic Geometry. It will be a rather general setup that enables us to speak not only about 2D/3D/Euclidean/elementary geometry "as we know it," but also about other structures that only look like geometric construction sequences.

4.1 Straight-Line Programs

A common and convenient way to describe polynomials without explicitly referring to their coefficients are *straight-line programs* (SLPs). They were introduced as an algebraic formulation for computations, and they have been proved useful as a measure for the complexity of polynomials [57]. Straight-line programs are in a way a sequence of elementary calculations (addition, multiplication, subtraction and – not necessarily – division) on input variables and intermediate results. With this tool you can encode functions (to be precise, rational functions if you allow division, or polynomials otherwise) in a very condensed way. It is even possible to describe some polynomials with doubly exponential degree with a linear number of operations (see Sec. 9.3.3 for a geometric translation of this). The book of Bürgisser et. al. [6] is a good source of other results with respect to straight-line programs and complexity.

As a formal reference, let us partially recall the definition of a straight-line program as presented in [6]:

Definition 4.1 (Straight-Line Program) *Let $\mathbb{K}$ be a field and let A be a $\mathbb{K}$-algebra. Let $\Omega = \mathbb{K}^c \cup \mathbb{K} \cup \{+, -, *, /\}$ be the set of* instructions*, where $\mathbb{K}^c$ denotes the set of 0-ary instructions λ^c that produce a constant $\lambda \in \mathbb{K}$, $\mathbb{K}$ denotes the set of unary instructions λ that are identified with the scalar multiplication by λ, and $\{+, -, *, /\}$ are the basic binary instructions for addition, subtraction, multiplication and division.*

Let n be a positive integer and $a \in A^n$. The pair $(A; a)$ is called an input of length n*. The arity of an operation $\omega \in \Omega$ is denoted by* $\mathrm{ar}(\omega)$.

1. (Syntax of a straight-line program) *A* straight-line program Γ *over* $\mathbb{K}$ *expecting inputs of length n is a sequence* $(\Gamma_1,\ldots,\Gamma_r)$ *of* instructions

$$\Gamma_i = (\omega_i; u_{i1},\ldots,u_{i\,\mathrm{ar}(\omega_i)}),$$

where $\omega_i \in \Omega$ *and the* $u_{i\ell}$ *are integers satisfying* $-n < u_{i\ell} < i$.

2. (Semantics of a straight-line program) *Let* $\Gamma = (\Gamma_1,\ldots,\Gamma_r)$ *be a straight-line program expecting inputs of length n,* Γ_i *as in 1.* Γ *is said to be* executable *on* $(A;a)$ *or* executable in A on input a *with* result sequence $b = (b_{-n+1},\ldots,b_r) \in A^{n+r}$, *if* $b_i = a_{n+i}$ *for* $-n+1 \le i \le 0$ *and* $b_i = \omega_i(b_{u_i1},\ldots,b_{u_i\,\mathrm{ar}(\omega_i)})$ *for* $1 \le i \le r$. $(A;b)$ *is called the* output *corresponding to the input* $(A;a)$.

3. *A straight-line program is* division-free *if it does not use divisions.*

Instead of repeating all the results about straight-line program we will present a new concept which will be shown to include straight-line programs.

4.2 Relational Instruction Sets

Clearly, although we can describe some geometric constructions coordinate-wise by SLPs, as shown in Ch. 5, we cannot expect straight-line programs to be able to describe even elementary constructions like circle-line-intersections (since these require – at least – square roots). A more flexible setup are *Geometric Straight-Line Programs (GSP)*. Since we do not want to restrict ourselves to a certain set of primitive operations (like addition, multiplication, division in the case of ordinary SLPs) or an underlying algebra A we first introduce the concept of a *relational instruction set (RIS)*.

Informally, a relational instruction set describes objects (like points, lines, conics) and possible constructions using these objects (like "point on line" or "angular bisector"). Instead of giving an algorithm or formulas for the constructions only a relation is specified that enables us to check for a certain input and output whether it is a valid construction.

Definition 4.2 (Relational Instruction Set) *A* relational instruction set *is a pair* (O,Ω) *of* objects O *and* primitive operations *(or* primitives*)* Ω *with the following properties:*
$O = (O_1,\ldots,O_k)$ *is a family of sets* O_i. *These sets partition the objects into classes of the same* type.
The primitive operations Ω *are relations*

$$\omega_i \subset (O_{x_1} \times \cdots \times O_{x_{s_i}}) \times O_{x_{s_i+1}}$$

with input size $\mathrm{ar}(\omega_i) = s_i$. *If we want to emphasize the input size we write* $\omega_i^{s_i}$. *An element of the set* $O_{x_1} \times \cdots \times O_{x_{s_i}}$ *is called* input *and an element of* $O_{x_{s_i+1}}$ *is called* output *of* ω_i.

Example 4.3 (Projective Geometry) *For a projective plane $\mathbb{P} = (P,L)$ of points P and lines L let*

$$\begin{aligned} \mathsf{Join} = \omega_1 &:= \{(p_1,p_2,l) \text{ s.t. } l \text{ is the line through} \\ &\quad p_1 \text{ and } p_2 \text{ and } p_1 \neq p_2\} \subset (P\times P)\times L \quad \text{and} \\ \mathsf{Meet} = \omega_2 &:= \{(l_1,l_2,p) \text{ s.t. } p \text{ is the point on} \\ &\quad l_1 \text{ and } l_2 \text{ and } l_1 \neq l_2\} \subset (L\times L)\times P \quad . \end{aligned}$$

Then $((O_1 = P, O_2 = L),(\omega_1,\omega_2))$ is a relational instruction set describing meet and join in Projective Geometry. The objects can have one of two different types, either they are a point or a line. Both primitives have input size 2. Furthermore, both primitives are determined, this means, for a given input there is at most one possible output (in projective planes two distinct lines meet in exactly one point, and two distinct points are connected by exactly one line). We will discuss determined primitives in more detail later.

Example 4.4 (Straight-line programs) *For a field $\mathbb{K}$ and a $\mathbb{K}$-Algebra A define a relational instruction set with only one object type, the objects $O_1 = A$ are the elements of the algebra. The primitives describe the arithmetic operations in the field.*

$$\begin{aligned} +^2 &:= \{(a,b,c) \text{ s.t. } a+b=c\} && \text{(addition)} \\ *^2 &:= \{(a,b,c) \text{ s.t. } ab=c\} && \text{(multiplication)} \\ -^2 &:= \{(a,b,c) \text{ s.t. } a=b+c\} && \text{(subtraction)} \\ /^2 &:= \{(a,b,c) \text{ s.t. } a=bc \text{ and } b\neq 0_{\mathbb{K}}\} && \text{(division)} \\ C^0_\lambda &:= \{\lambda\} \quad \lambda\in\mathbb{K} && \text{(constants)} \\ S^1_\lambda &:= \{(\lambda a,a) \text{ s.t. } a\in A\} \quad \lambda\in\mathbb{K} && \text{(scalar multiplication)} \end{aligned}$$

Observe that we are able to describe subtraction and division without referring to the basic operations in the algebra. Actually, the whole RIS does not depend on A being an algebra. But if A is indeed an algebra, then the primitives are again determined.
We can even add other primitives like

$$\sqrt{\cdot}^1 := \{(a,b) \text{ s.t. } a = bb\} \qquad \text{(square root)}$$

This definition makes sense even if the algebra is not an algebraically closed field. In general algebras there is not for every input a an output b such that $(a,b)\in\sqrt{\cdot}^1$.

Relational instruction sets provide a way to describe constructions without requiring the ability to know a way, or even the mere existence of a way to carry out the necessary calculations. Observe that this is not only important in cases where there exists no "solution" for a certain input of a primitive, but also when there is more than one: Take the square root primitive over the field of rational numbers $\mathbb{Q}$. For most inputs you cannot find a square root within $\mathbb{Q}$. But for some, say 4, there are two square roots: -2 and 2.

We do not have to specify which one we mean, as we had to do if we tried to define a square root *function* $\sqrt{\cdot}$. This is crucial throughout this thesis: We have to find a method that deals with ambiguities that occur in geometric constructions. At this point, we do not yet present a method to deal with the ambiguities, but at least we can describe them.

Example 4.5 (Ruler and Compass Constructions) *In the projective plane $\mathbb{P} = (P,L)$ of points P and lines L let C be all circles (with respect to Euclidean measurements). The primitives of a RIS (O,Ω) with $O = (P,L,C)$ that can handle intersections of circles and lines or circles and circles are given by:*

$$\begin{aligned}
\mathsf{Join} = \omega_1 &:= \{(p_1,p_2,l) \text{ s.t. } l \text{ is the line through } p_1 \text{ and } p_2 \text{ and } p_1 \neq p_2\} \subset (P\times P)\times L \quad , \\
\mathsf{Meet} = \omega_2 &:= \{(l_1,l_2,p) \text{ s.t. } p \text{ is the point on } l_1 \text{ and } l_2 \text{ and } l_1 \neq l_2\} \subset (L\times L)\times P \quad , \\
\mathsf{Circle} = \omega_3 &:= \{(p_1,p_2,c) \text{ s.t. } c \text{ is a circle with center } p_1 \text{ and } p_2 \neq p_1 \text{ lies on the circle}\} \subset (P\times P)\times C \quad , \\
\mathsf{CLInt} = \omega_4 &:= \{(c,l,p) \text{ s.t. } p \text{ is a point of intersection of circle } c \text{ and line } l\} \subset (C\times L)\times P \quad \text{, and} \\
\mathsf{CCInt} = \omega_5 &:= \{(c_1,c_2,p) \text{ s.t. } p \text{ is a point of intersection of the circles } c_1 \text{ and } c_2\} \subset (C\times C)\times P \quad .
\end{aligned}$$

Again, we do not care about ambiguities, vanishing intersections or any problems that might occur if we had to write down explicit functions to describe the primitives.

4.3 Geometric Straight-Line Programs

Let us point out that the relational instruction sets are in a way more useful than a constraint-based approach to geometric constructions: They reflect the constructiveness of the geometric structures we want to study. By defining geometric straight-line programs as below we can be sure that there will be no cycles in the dependency graph of the elements in a construction.

Definition 4.6 (Geometric Straight-Line Program) *A* geometric straight-line program *or* GSP *on a relational instruction set (O,Ω) is defined by a triple (X,R,Γ). $X = (X_1,\ldots,X_n)$ are called* input variables. *$R = (R_0,\ldots,R_{m-1})$ are called* output variables *or* intermediate results *and $\Gamma = (\Gamma_0,\ldots,\Gamma_{m-1})$ are called* statements. *Every statement Γ_i is a primitive operation ω_{j_i} of input size s_{j_i} and s_{j_i}* pointers $u_1^{(i)}\ldots u_{j_i}^{(i)} \in [-n,\ldots,i-2] \subset \mathbb{Z}$. *The* length *of the GSP is m.*

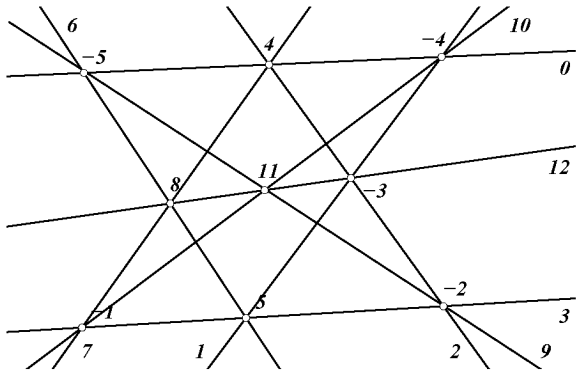
Remark 4.7 *We will omit the reference to the RIS whenever it is immediate from the context.*

Example 4.8 (A Projective Geometry program) *Consider the straight-line program with input variables $X = (X_1, \ldots, X_5)$, intermediate results $R = (R_0, \ldots, R_{12})$ and the statements shown in the table below.*

You may read the table like this: The pointers in the primitives point to the input or output variables, if a pointer i is less than zero, then it points to $X_{|i|}$, else to R_i. The pointers denote the input of the primitive, the output is the output variable on the same line.

Index	*Var/Result*	*Statement*
-5	X_5	–
-4	X_4	–
-3	X_3	–
-2	X_2	–
-1	X_1	–
0	R_0	Join -5,-4
1	R_1	Join -4,-3
2	R_2	Join -3,-2
3	R_3	Join -2,-1
4	R_4	Meet 0,2
5	R_5	Meet 1,3
6	R_6	Join -5,5
7	R_7	Join -1,4
8	R_8	Meet 6,7
9	R_9	Join -5,-2
10	R_{10}	Join -1,-4
11	R_{11}	Meet 9,10
12	R_{12}	Join 8,-3

The straight-line program on the left could be interpreted like in this construction of Pappos' theorem. The input variables are five points, the meet and join primitives prescribe lines and new points.



Observe, however, that we did not yet define any semantics of straight-line programs yet. The picture on the right is only an illustration of what we would like to be able to describe.

Notation 4.9 *As a shorthand we refer to X_i by the notation R_{-i}.*

We will formalize the interpretation of a geometric straight-line program by the notion of an *instance*. An instance of a straight-line program is an assignment of objects to the variables that is consistent with the primitives. In the case of our example above it was sufficient to assign distinct points to the variables X_1 to X_5, and the intermediate results R_i, in particular their types, were determined. This only worked because the primitives and pointers matched. Let us formalize this as a *well-defined* GSP.

Definition 4.10 (well-defined) *A geometric straight-line program (X, R, Γ) is* well-defined *if there is an assignment of types to the input and output variables such that for every statement $\Gamma_i = (\omega_j; u_1^{(i)}, \ldots, u_{s_j}^{(i)})$ and every pointer $p \in u_1^{(i)}, \ldots, u_{s_j}^{(i)}$ the type of the variable the*

pointer references and the type of the corresponding set of objects in the input of ω_j are the same, and the output type of the primitive in the k-th statement equals the type of R_k.

Remark 4.11 *It is obvious that the assignment of types is unique, so it is justified to call this assignment the* type assignment *of the GSP.*

This technical definition just ensures that a the statements of a straight-line program work on objects of the correct type. Now we will fill the variables with life – we will look for an assignment of objects to the variables that respects not only the type assignment, but also the primitives. The notion of an *instance* is similar to the definition of *executable* with ordinary straight-line programs.

Definition 4.12 (Instance) *An* instance *of a geometric straight-line program (X,R,Γ) at $X=\tilde{X}$ is an assignment of objects $\tilde{X}=\tilde{X}_1,\ldots,\tilde{X}_n$ and $\tilde{R}=\tilde{R}_0,\ldots,\tilde{R}_{m-1}$) to the variables $X=(X_1,\ldots,X_n)$ and $R=(R_0,\ldots,R_{m-1}$ such that*

1. *The object types of $\tilde{X}$ and $\tilde{R}$ match the type assignment of the GSP, and*

2. *All primitives are* satisfied, *that is, for every statement $\Gamma_i=(\omega_j;u_1^{(i)},\ldots,u_{s_j}^{(i)})$ the relation*

$$(\tilde{R}_{u_1^{(i)}},\ldots,\tilde{R}_{u_{s_j}^{(i)}},\tilde{R}_j)\in\omega_j$$

is true (we extend Notation 4.9 to the tilde-version of X and R).

Remark 4.13 *Although geometric straight-line programs are very similar to straight-line programs as in Def. 4.1 there are some conceptual differences.*

- *The $\mathbb{K}$-Algebra is part of the input of a straight-line program. For GSPs the relational instruction set is fixed in advance.*

- *The result sequence of a straight-line program is determined by the the input, if it exists. GSPs need not be determined, in fact, there might be several instances with the same input variables but different intermediate results.*

- *What is known as instructions is called statement, just to avoid confusion with the instructions of the RIS.*

Let us define some basic properties that geometric straight-line programs can have. First we rule out syntactically correct, but semantically programs that make no sense.

Definition 4.14 (Consistency) *A geometric straight-line program is* consistent *if there exists an instance of it,* inconsistent *otherwise.*

From now on most GSPs under consideration will be consistent. Nevertheless you should take care: To decide whether a certain GSP is consistent is not trivial at all, it is at least NP hard (see Sec. 7.5 for more information about complexity issues).

The next definition fixes what we already mentioned informally when discussing the Projective Geometry RIS.

Definition 4.15 (Determinism) *A geometric straight-line program is* determined *if any assignment of the input variables can be completed to at most one instance. A relational instruction set (O,Ω) is* determined *if every straight-line program on (O,Ω) is determined.*

For some relational instruction sets it is easy to determine whether they are determined: If every primitive ω_i has for every input exactly one "output," then a straight-line program cannot "branch," it is determined.

Definition 4.16 (Determined Primitive) *A primitive*

$$\omega_i \subset O_{i1} \times \cdots \times O_{i\,\mathrm{ar}(\omega_i)} \times O_{i(\mathrm{ar}(\omega_i)+1)} =: O$$

is determined, *if for each $a \in O_{i1} \times \cdots \times O_{i\,\mathrm{ar}(\omega_i)}$ there exists at most one object $b \in O_{i(\mathrm{ar}(\omega_i)+1)}$ such that $(a,b) \in O$.*

Theorem 4.17 (Determined GSPs) *If every primitive of a RIS (O,Ω) is determined, so is every geometric straight-line program over (O,Ω), and thus the RIS itself. The converse is also true.*

Proof 4.18 *Easy induction on the length of a GSP. A GSP of length 1 is clearly determined, since the only possible instance is the element (a,b) of the primitive that matches the assignment of the input variables.*

Let us assume that the theorem holds for any GSP of length r, and let (X,R,Γ) be a GSP of length $r+1$. The GSP $(X,R,(\Gamma_1,\ldots,\Gamma_r))$ is determined. If there is no instance for the shortened GSP then there is no instance for the original GSP and we are done. Let $(\tilde{X},\tilde{R})$ be the unique instance of the shortened GSP otherwise. Then there is at most one element $(a,b) \in \omega_{r+1}$ where a is the assignment of the input of the last primitive that is determined by the instance, and $(\tilde{X},\tilde{R},b)$ is the only instance possible of (X,R,Γ).

The equivalence statement follows from the trivial GSP of length 1 that uses only a statement with a non-determined primitive. □

Example 4.19 (Projective Geometry) *The RIS for Projective Geometry as introduced in Ex. 4.3 is determined, since the two primitives* Meet *and* Join *are determined. Observe that we excluded coincident lines or points as input for the primitives: Without that restriction the primitives could be undetermined.*

Example 4.20 (Straight-Line Programs with Square Roots) *The RIS for straight-line programs (Ex. 4.4) with additional $\sqrt{\cdot}^1$-primitive is not determined, since the $\sqrt{\cdot}^1$-primitive is undetermined. The number of instances can be exponentially large with respect to the length of the GSP. Let $\mathbb{K} = \mathbb{C}$ and $A = \mathbb{K}$, and consider the SLP on the left:*

Index	*Var/Result*	*Statement*	
-1	X_1	–	
0	R_0	$\sqrt{\cdot}^1$	*-1*
1	R_1	$\sqrt{\cdot}^1$	*0*
2	R_2	$\sqrt{\cdot}^1$	*1*
3	R_3	$\sqrt{\cdot}^1$	*2*
$\vdots$	$\vdots$	$\vdots$	
r	R_r	$\sqrt{\cdot}^1$	*r-1*

For $\tilde{X}_1 = 1$ there are 2^{r+1} instances of this GSP. You can easily find them all by looking at the last output variable: It must be assigned a value z that satisfies $z^{2^{r+1}} = 1$, i.e. it is a 2^{r+1}-th root of unity. Any of these will determine the variables preceding R_r, so we have a 1-1 mapping of the roots of unity to the instances of the GSP with input $\{1\}$.

This explosion of the number of instances even for only slightly undetermined primitives will be a major obstacle in the implementation of a continuous Dynamic Geometry system, see Ch. 6 and Ch. 9.

Chapter 5

Projective Dynamic Geometry

As a first introduction to what one might expect from a theory that handles Dynamic Geometry we look at "projective Dynamic Geometry (PDG)." The fundamental objects are points and lines in the real (or complex) projective plane.

In this chapter we will see that it is very easy to do geometry on a computer [67], as long as only lines and points (or, in higher dimensions, affine subspaces) are involved. After choosing the right coordinatization (see Sec. 5.1.1) it is almost trivial to work with incidence geometry, and the results (on screen) are indeed what everybody would expect.

Unfortunately, this raises the bar for larger (more functionality/objects) systems: The user expects much more than what is easily achievable, and it is not a trivial task to explain why the computer does not "behave nicely." We will discuss these problems in chapter 6, let us first concentrate on the easy part.

5.1 Points and Lines

From now on we will work in the real or complex projective plane $\mathbb{R}P^2$ resp. $\mathbb{C}P^2$. When the basic field is not important we will refer to it as $\mathbb{K}$.

The *points* in $\mathbb{K}P^2$ are the 1-dimensional linear subspaces of $\mathbb{R}^3$, the *lines* are the 2-dimensional linear subspaces of $\mathbb{R}^3$ (the subspaces of co-dimension 1). Since every 1-dimensional subspace $U \subset V$, for arbitrary $\mathbb{R}$-vector spaces V, can be written as

$$U = \{\lambda x, \lambda \in \mathbb{R}\}$$

with $x \in V \setminus \{0\}$ we can identify antipodal point-pairs on the unit sphere S_2 with the points in the real projective plane. For lines we can work with the orthogonal complement of the subspace, which is 1-dimensional, and we can identify antipodal point-pairs with these complements.

5.1.1 Homogeneous Coordinates

The most common coordinatization of points uses two coordinates, and lines are usually represented by a linear equation in x and y. It dates back to the 19th century that a better way of coordinatization of points and lines has been found. In his article "Ueber ein neues Coordinatensystem" [63] Julius Plücker describes an approach to Projective Geometry that uses three points in the projective plane as reference points for a coordinate system. He finds that using three coordinates for points (instead of the usual coordinate system) has the enormous advantage of ending up with homogeneous equations for curves – all monomials in the polynomial describing the curve have the same total degree. This make calculations very easy and straight forward. As Plücker says [63]:

> "Ich habe bei den folgenden Entwicklungen nur die Absicht gehabt, an Beispielen zu zeigen, dass die neue Methode einerseits zum Beweise vorgelegter einzelner Sätze und zur Darstellung allgemeiner Theorien sich sehr geschmeidig zeigt, und dass sie andrerseits Resultate *finden* lehrt, wenn man sie aus allgemeinen analytischen Gesichtspunkten betrachtet."

which means that he wants to show how smooth the new method (of homogeneous coordinates) can be used to prove theorems and actually helps in the process of finding new results.

Another interesting point is that Plücker remarks that the biggest advantage of the new coordinatization may be its application to the (at that time) new developments in mechanics, but he writes that he could not consider this in this article. He could not foresee that his work would be even more valuable for the field of computer graphics and visualization.

Let us describe homogeneous coordinates from a modern, 20th century computer scientists, point of view.

We represent points by three coordinates (x,y,z), not all being zero. This point $p := (x,y,z)$ in $\mathbb{R}^3$ defines a unique linear subspace of $\mathbb{R}^3$ that contains the origin and p. Since all $\lambda(x,y,z)$, $\lambda \in \mathbb{R} \setminus \{0\}$ represent the same subspace, i.e. the same projective point, we identify scalar multiples: $(2,2,4)$ and $(1,1,2)$ are the same point. We choose row vectors or column vectors for the coordinates, whichever is more convenient in a particular case. If it is not clear from the context, we assume that points are represented as column vectors, and lines are represented as row vectors.

We do the same with lines: By $(a,b,c) \in \mathbb{R}^3$ we represent the linear subspace that is the orthogonal complement of the two-dimensional linear subspace that corresponds to the line.

Embedding of the Euclidean Plane

What do we get by this representation? First of all, we have a nice embedding of $\mathbb{R}^2$, the Euclidean plane, in this coordinatization of $\mathbb{R}P^2$. All points of the Euclidean plane can be

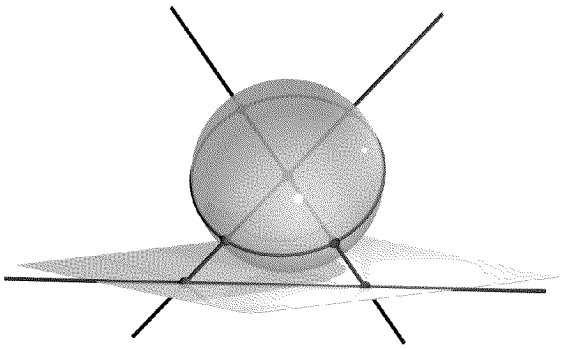


Figure 5.1: The projection of points on the Euclidean plane to the sphere. This picture explains how the Euclidean plane is embedded into $\mathbb{R}P^2$. Points at infinity can be found on the equator of the sphere.

written as $(x,y,1)$, the *homogenization* of $(x,y) \in \mathbb{R}^2$. Clearly, all points that have a third coordinate that differs from zero can be identified with a point of the Euclidean plane, since $(x,y,\lambda) = (\frac{x}{\lambda}, \frac{y}{\lambda}, 1)$.

The remaining points of $\mathbb{R}P^2$, those having 0 as the third coordinate, are the points on the "line at infinity," a one-dimensional projective space isomorphic to $\mathbb{R}P^1$. By choosing another normalization of the triples (x,y,z) we have another nice picture that puts the line at infinity into our reach: Let $|(x,y,z)|$ denote the Euclidean norm $\sqrt{x^2+y^2+z^2}$ of (x,y,z). Then for $\nu := |(x,y,z)|$ the vectors $\pm(\frac{x}{\nu}, \frac{y}{\nu}, \frac{z}{\nu})$ are antipodal points on unit sphere S^2. Every point that belongs to the Euclidean plane located at $z = 1$ is mapped on the union of the open upper and lower hemispheres of S^2, the points that are on the line at infinity are mapped to the equator $E := S^2 \cap \{(x,y,z) \,|\, z = 0\}$.

Easy tests

How can we check whether a point is coincident with a line? In the projective plane the two linear subspaces defined by the line and the point must have a non-trivial intersection. This is the case when the two 1-dimensional subspaces, the linear subspace of the point and the orthogonal complement subspace of the line, are orthogonal. This can be easily checked in our coordinatization: No matter which representation of the projective elements we take, the scalar product of the two must be zero, since the two coordinatizations of the point and the line have a zero scalar product if and only if the subspaces spanned by the coordinatizations are orthogonal.

Other relations on points and lines are collinearity resp. concurrency. Three points are collinear if and only if there is a line that meets all three of them. This happens if and only if the three subspaces of the points are linearly dependent, which is exactly true when the determinant of the three coordinatizations is 0. Observe that this criterion is independent of the special coordinatization, since the determinant is multilinear, and non-zero scalar multiples can not change the result from zero to non-zero or vice-versa.

Three lines are concurrent if and only if all three meet in one point. In that case the

three two-dimensional linear subspaces share a common one-dimensional linear subspace, and all normal vectors lie in the (two-dimensional) orthogonal complement. This is the same condition as above: Three lines are concurrent if and only if the determinant of the three coordinatizations is 0.

Basic Operations: Meet and Join

We can not only check whether a line meets a point, but we can also calculate explicitly the unique line joining two different points or the unique point of intersection of two lines.

The three-dimensional vector or cross product calculates a vector orthogonal to the two factors:

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \times \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} := \left(\begin{vmatrix} y_1 & y_2 \\ z_1 & z_2 \end{vmatrix}, \begin{vmatrix} z_1 & z_2 \\ x_1 & x_2 \end{vmatrix}, \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} \right)^T$$

It is easy to check that for any two vectors v and w the scalar products $(v \times w)^T v$ and $(v \times w)^T w$ are equal to zero, since $(a \times b)^T c = \det(a, b, c)$.

So if we use the coordinatizations of two points and calculate their cross product, we get a coordinatization of a linear subspace orthogonal to both subspaces induced by the points. If we interpret this as the orthogonal complement of a two-dimensional linear subspace, then this subspace, representing a line, contains the first two subspaces: It is the line joining the two points. On the other hand, if we start with the orthogonal subspaces of lines we will end up with a linear subspace that represents the intersection point of the two lines.

Remark 5.1 *When we try to calculate the intersection of ℓ and m where $\ell = m$, we will get $(0,0,0)$ as a result – an invalid coordinate triple. The same holds for the join of points p and q, if $p = q$.*

Duality/Polarity

As we have seen above, the concepts of points and lines are not very different. There is a natural duality between them, which is reflected by the symmetry of the homogeneous coordinates. Here is a translation table:

$$\begin{aligned} \text{point} &\Leftrightarrow \text{line} \\ \text{meet} &\Leftrightarrow \text{join} \\ \text{collinear} &\Leftrightarrow \text{concurrent} \\ \text{point on line} &\Leftrightarrow \text{line through point} \end{aligned}$$

Any theorem or other statement about points and lines in projective space can be translated – using this table – into another, equivalent statement. This *duality* or *polarity* (the second notion seems to be more popular to people working in polytope theory) is very useful: Often a proof of a geometric theorem is much nicer or might be easier to find in the polar setup.

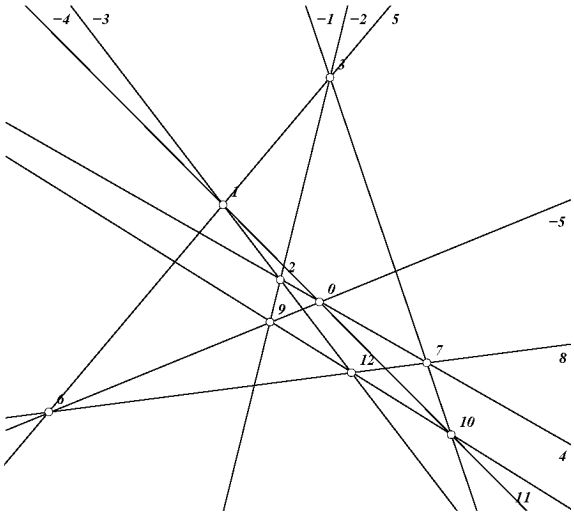


Figure 5.2: A polar version of Pappos' Theorem. The labelling of points and lines is the same (only polarized) as in example 4.8. Observe that the theorem is exactly the same: Pappos is self-polar.

Cinderella offers a polar view of the Euclidean plane, see Ch. 2.

5.1.2 GSP formulation of point/line constructions

We already presented a relational instruction set that formalized points, lines, meet and join in Projective Geometry, without using coordinates. It would be easy to include the other predicates – point on line, line through point, collinear and concurrent – that were introduced in the last section. For the collinearity and concurrency primitives a new object type that mimicks a boolean variable, consisting of only two objects, `TRUE` and `FALSE` could be introduced.

Definition 5.2 (Abstract RIS for points and lines) *The abstract RIS for points and lines is the RIS as introduced in example 4.3, augmented by new objects* $B = \{\texttt{TRUE}, \texttt{FALSE}\}$ *and new primitives*

$$
\begin{aligned}
\mathsf{POL}^2 &= \{(\ell, a) \text{ s.t. } a \text{ lies on } \ell\} \subset L \times P \\
\mathsf{LTP}^2 &= \{(a, \ell) \text{ s.t. } \ell \text{ meets } a\} \subset P \times L \\
\mathsf{Coll}^3 &= \{(a,b,c,\texttt{TRUE}) \text{ s.t. } a,b,c \text{ are collinear}\} \\
&\cup \{(a,b,c,\texttt{FALSE}) \text{ s.t. } a,b,c \text{ are not collinear}\} \subset P \times P \times P \times B, \quad \text{and} \\
\mathsf{Conc}^3 &= \{(\ell,m,n,\texttt{TRUE}) \text{ s.t. } \ell,m,n \text{ are concurrent}\} \\
&\cup \{(\ell,m,n,\texttt{FALSE}) \text{ s.t. } \ell,m,n \text{ are not concurrent}\} \subset L \times L \times L \times B.
\end{aligned}
$$

Although this extended RIS could describe the projective plane, its points and lines and the basic operations, it is a little bit too abstract to be useful. Instead, we will define a RIS that is based on homogeneous coordinates.

Definition 5.3 (Homogeneous RIS for points and lines) *The objects of this RIS are the vectors in $\mathbb{K}^3$ and $\mathbb{K}$-scalars. We do not identify scalar multiples of these vectors, and we do not exclude the zero vector. So we have $O = (O_1, O_2)$ with $O_1 := \mathbb{K}^3$ and $O_2 := \mathbb{K}$.*

We have only three primitives, that correspond to the cross product, the scalar product and determinants:

Cross Product:	$\mathsf{Cross}^3 = \{(x,y,z)\ s.t.\ z = x \times y\}$	$\subset (O_1)^3$
Scalar Product:	$\mathsf{Scal}^2 = \{(x,y,z)\ s.t.\ z = x^T y\}$	$\subset (O_1)^2 \times O_2$
Determinant:	$\mathsf{Det}^4 = \{(x,y,z,r)\ s.t.\ r = \det(x,y,z)\}$	$\subset (O_1)^3 \times O_2$

Its immediate from their definitions that these primitives are determined, so the same is true for every straight-line program on the homogeneous RIS. This matches our intuition: There is only one instance of a construction with points and lines if you fix the input parameters. We will investigate this and other properties in Sec. 5.2.

By moving away from the abstract definitions of points and lines to homogeneous coordinates we lost the distinction between points and lines. If we would like to fix that we could do so by introducing two disjoint sets $O_1 = \mathbb{K}^3$ and $O_1' = \mathbb{K}^3$ that represent points resp. lines. The cross product had to be split in a meet and a join primitive, both working on the correct sets. The scalar product had to be changed to be valid for one point and one line only, and the determinant should ensure that only objects of the same type are used. But since we would not get any additional insight by these formalizations, we decided to work with the same objects for points and lines. This is a little bit like ignoring the difference between row and column vectors. This makes a wider range of constructions possible: By using a point as a line we can change from a line to its polar point and vice versa. Although we can actually construct more with this extra operation, we do not get more computational power from an algebraic point of view, as we will see in Sec. 5.1.5.

Another change from the abstract RIS is that we allowed the zero vector to be used as homogeneous coordinates. This zero-vector does not represent a projective point or line, but we can assign it to an "invalid point" or "invalid line." So when does this invalid point appear? Let us assume that we did not feed it as an input point. Then the only way to get an invalid point is to calculate the cross product of two vectors that are scalar multiples of each other. This means, we are trying to carry out a join or meet operation on two equal lines or equal points, which we did not allow on the abstract RIS.

On the homogeneous RIS we do not restrict ourselves like that. We allow operations that create invalid objects, and we allow further operations on these invalids.

What happens if we feed an invalid point into one of the three predicates? For the cross product we will just end up with the zero vector, i.e. the invalid object, and the scalar product and the determinant will evaluate to zero as soon as an invalid element is used as input (see 5.1). This means that the point on line and line through point tests and the collinearity and concurrency conditions are always true for invalid points or lines. This will be useful later, when we want to prove theorems by random instances – when a construction degenerates, this will not count as an counter-example for a theorem.

Remark 5.4 *The determinant primitive is redundant, since we can replace it by a combined cross and scalar product as in* $\det(x,y,z) = (x \times y)^T z$.

Geometrically: Three points are collinear if and only if the line through two of them meets the third point, and three lines are collinear if and only if the intersection of two of them lies on the third line. So we can assume that a GSP on a homogeneous RIS does not use determinants, if we need it:

Definition 5.5 (Determinant-free GSP) *A GSP on a homogeneous RIS is* determinant-free *if it does not contain a* Det *statement.*

5.1.3 An SLP formulation

The above did not really need the concept of a geometric straight-line program. In fact, we will show now that the homogeneous RIS is compatible with (or equivalent to) ordinary straight-line programs.

The first observation is that we can compile every GSP on a homogeneous RIS to an SLP over $\mathbb{K}$, that is in a certain way equivalent to the GSP. In the following transformation, as in all the transformations in this thesis, we will use pointers that are pairs of indices, and we assume a lexicographic order on these indices.

Transformation 5.6 (Transformation of an GSP on a homogeneous RIS to an SLP) *Let* (X,R,Γ) *be a GSP on the homgeneous RIS* $((O_1 = \mathbb{K}^3, O_2 = \mathbb{K}), \Omega = (\mathsf{Cross}, \mathsf{Scal}, \mathsf{Det}))$. *We replace every variable* X_i *or* R_i *of type* O_1 *with three variables* $X_{i,j}$ *or* $R_{i,j}$. *By remark 5.4 we can assume that* Γ *is determinant-free. The remaining statements of* Γ *are transformed into several SLP instructions as in this table:*

GSP statement				$\leftrightarrow$	*SLP code*			
k:	Cross	i	j	$\leftrightarrow$	$k,-5$:	$*$	$i,2$	$j,3$
					$k,-4$:	$*$	$i,3$	$j,2$
					$k,-3$:	$*$	$i,3$	$j,1$
					$k,-2$:	$*$	$i,1$	$j,3$
					$k,-1$:	$*$	$i,1$	$j,2$
					$k,0$:	$*$	$i,2$	$j,1$
					$k,1$:	$-$	$i,-5$	$i,-4$
					$k,2$:	$-$	$i,-3$	$i,-2$
					$k,3$:	$-$	$i,-1$	$i,-0$
k:	Scal	i	j	$\leftrightarrow$	$k,-3$:	$*$	$i,1$	$j,1$
					$k,-2$:	$*$	$i,2$	$j,2$
					$k,-1$:	$*$	$i,3$	$j,3$
					$k,0$:	$+$	$k,-3$	$k,-2$
					k:	$+$	$k,0$	$k,-1$

After renumbering the variables in the transformed GSP we end up with an SLP that calculates each variable individually. The input of the SLP is the split input of the original GSP, the output of the SLP must be re-blocked to get the output of the GSP.

Remark 5.7 *We could have transformed the GSP into an SLP over the $\mathbb{K}$-algebra $A = \mathbb{K}^3$ that has the cross product as multiplication. But we did not want to rely on the special multiplication, since we can fall back to the field $\mathbb{K}$, and in the next section we will do the reverse transformation, which is not possible for all algebras A.*

5.1.4 Homogeneous RIS GSPs and division-free SLPs are equivalent

Transformation 5.6 showed that any GSP on a RIS may be expressed as an division-free SLP. Now we will show the converse: Every division-free SLP over a field $A = \mathbb{K}$ can be emulated by an appropriate homogeneous RIS GSP.

The construction is easy and builds on the fact that we can do multiplications, additions and substractions geometrically using von-Staudt constructions (due to Karl Georg Christian von Staudt, see [80, 81]). Here are three formulas that we will need for the transformation. We can emulate multiplication, addition and substraction in $\mathbb{K}$ by repeated cross products in $\mathbb{K}^3$ using some additional constant vectors:

Addition:

$$\begin{aligned}
&\left[\left[\left[\begin{pmatrix}x\\0\\1\end{pmatrix}\times\begin{pmatrix}0\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\left[\left[\begin{pmatrix}y\\0\\1\end{pmatrix}\times\begin{pmatrix}0\\1\\0\end{pmatrix}\right]\times\begin{pmatrix}0\\-1\\1\end{pmatrix}\right]\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&\quad=\left[\left[\begin{pmatrix}-1\\-x\\x\end{pmatrix}\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\left[\begin{pmatrix}-1\\0\\y\end{pmatrix}\times\begin{pmatrix}0\\-1\\1\end{pmatrix}\right]\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&\quad=\left[\begin{pmatrix}-x\\1\\0\end{pmatrix}\times\begin{pmatrix}y\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&\quad=\begin{pmatrix}1\\x\\-(x+y)\end{pmatrix}\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&\quad=\begin{pmatrix}x+y\\0\\1\end{pmatrix}
\end{aligned}\tag{5.1}$$

Subtraction:

$$\begin{aligned}
&\left[\left[\left[\left[\left[\begin{pmatrix}y\\0\\1\end{pmatrix}\times\begin{pmatrix}0\\1\\0\end{pmatrix}\right]\times\begin{pmatrix}0\\-1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\left[\left[\begin{pmatrix}x\\0\\1\end{pmatrix}\times\begin{pmatrix}0\\1\\0\end{pmatrix}\right]\times\begin{pmatrix}0\\-1\\1\end{pmatrix}\right]\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&\quad=\left[\left[\left[\begin{pmatrix}y\\1\\1\end{pmatrix}\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}x\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&\quad=\left[\left[\begin{pmatrix}1\\-y\\0\end{pmatrix}\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}x\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&\quad=\left[\begin{pmatrix}y\\1\\0\end{pmatrix}\times\begin{pmatrix}x\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&\quad=\begin{pmatrix}1\\-y\\-(x-y)\end{pmatrix}\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&\quad=\begin{pmatrix}x-y\\0\\1\end{pmatrix}
\end{aligned}\tag{5.2}$$

Multiplication:

$$\begin{aligned}
&\left[\left[\left[\left[\left[\left[\begin{pmatrix}y\\0\\1\end{pmatrix}\times\begin{pmatrix}0\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}1\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}1\\0\\0\end{pmatrix}\right]\times\begin{pmatrix}x\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&=\left[\left[\left[\left[\left[\left[\begin{pmatrix}-1\\-y\\y\end{pmatrix}\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}1\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}1\\0\\0\end{pmatrix}\right]\times\begin{pmatrix}x\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&=\left[\left[\left[\left[\left[\begin{pmatrix}-y\\-1\\0\end{pmatrix}\times\begin{pmatrix}1\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}1\\0\\0\end{pmatrix}\right]\times\begin{pmatrix}x\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&=\left[\left[\left[\left[\begin{pmatrix}-1\\-y\\1\end{pmatrix}\times\begin{pmatrix}1\\0\\0\end{pmatrix}\right]\times\begin{pmatrix}x\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&=\left[\left[\left[\begin{pmatrix}0\\1\\y\end{pmatrix}\times\begin{pmatrix}x\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&=\left[\left[\begin{pmatrix}1\\xy\\-x\end{pmatrix}\times\begin{pmatrix}0\\0\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&=\left[\begin{pmatrix}xy\\-1\\0\end{pmatrix}\times\begin{pmatrix}0\\1\\1\end{pmatrix}\right]\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&=\begin{pmatrix}-1\\-xy\\xy\end{pmatrix}\times\begin{pmatrix}0\\1\\0\end{pmatrix}\\
&=\begin{pmatrix}xy\\0\\1\end{pmatrix}
\end{aligned}\tag{5.3}$$

With these formulas in mind, we are ready for

Transformation 5.8 (Transformation from SLP to homogeneous RIS GSP) *For $x\in\mathbb{K}$ let $\tilde{x}\in\mathbb{K}^3$ be the vector $\tilde{x}=\begin{pmatrix}x\\0\\1\end{pmatrix}$. Given an SLP (X,R,Γ) over $\mathbb{K}$ with input $X_1,\ldots,X_n$ and length m we write down a GSP, and we give transformations for the input of the SLP to the input of the GSP. Finally, we identify variables where we can read off the values of the variables of the original SLP.*

The input of the GSP will be $\tilde{X}=(\tilde{X}_1,\ldots,\tilde{X}_n,C_1,\ldots,C_6)$, all of type O_1. An input vector $(x_1,\ldots,x_n)$ of the SLP is mapped to

$$\left(\begin{pmatrix}1\\0\\1\end{pmatrix},\begin{pmatrix}0\\1\\1\end{pmatrix},\begin{pmatrix}0\\0\\1\end{pmatrix},\begin{pmatrix}0\\1\\0\end{pmatrix},\begin{pmatrix}1\\0\\0\end{pmatrix},\begin{pmatrix}0\\-1\\1\end{pmatrix},\tilde{x}_1,\ldots,\tilde{x}_n\right)$$

The instructions of Γ are transformed as in this table. As an abuse of notation C_i stands for the pointer $-(n-6+i)$ that points to C_i:

SLP instruction				$\leftrightarrow$	*GSP code*			
k:	$+$	i	j	$\leftrightarrow$	$k,-5$:	Cross	$i,0$	C_2
					$k,-4$:	Cross	$k,-5$	C_3
					$k,-3$:	Cross	$j,0$	C_4
					$k,-2$:	Cross	$k,-3$	C_6
					$k,-1$:	Cross	$k,-4$	$k,-2$
					$k,0$:	Cross	$k,-1$	C_4
					k:	Scal	$k,0$	C_5
k:	$-$	i	j	$\leftrightarrow$	$k,-7$:	Cross	$j,0$	C_4
					$k,-6$:	Cross	$k,-7$	C_6
					$k,-5$:	Cross	$k,-6$	C_3
					$k,-4$:	Cross	$k,-5$	C_3
					$k,-3$:	Cross	$i,0$	C_4
					$k,-2$:	Cross	$k,-3$	C_6
					$k,-1$:	Cross	$k,-4$	$k,-2$
					$k,0$:	Cross	$k,-1$	C_4
					k:	Scal	$k,0$	C_5
k:	$*$	i	j	$\leftrightarrow$	$k,-7$:	Cross	$i,0$	C_2
					$k,-6$:	Cross	$k,-7$	C_3
					$k,-5$:	Cross	$k,-6$	C_1
					$k,-4$:	Cross	$k,-5$	C_5
					$k,-3$:	Cross	$k,-4$	$j,0$
					$k,-2$:	Cross	$k,-3$	C_3
					$k,-1$:	Cross	$k,-2$	C_2
					$k,0$:	Cross	$k,-1$	C_4
					k:	Scal	$k,0$	C_5

In the case where i is negative $i,0$ refers to i. In the new program $k,0$ holds the tilde-version of k. So we can re-use the results of the cross-product computations, and we also have access to a variable which contains exactly the same value as the corresponding variable in the original SLP.

The correctness of the transformation follows from equations 5.1, 5.2 and 5.3.

How did we come up with the cross product construction? The basic building blocks used are the von-Staudt constructions for multiplication and addition/subtraction (von Staudt used these constructions to get rid of coordinates in Projective Geometry, see also [17]). We can read off the formulas directly from these figures, keeping in mind that

- the line at infinity has coordinates $(0,0,1)$ and parallel lines meet at infinity,
- the y-axis has coordinates $(1,0,0)$,

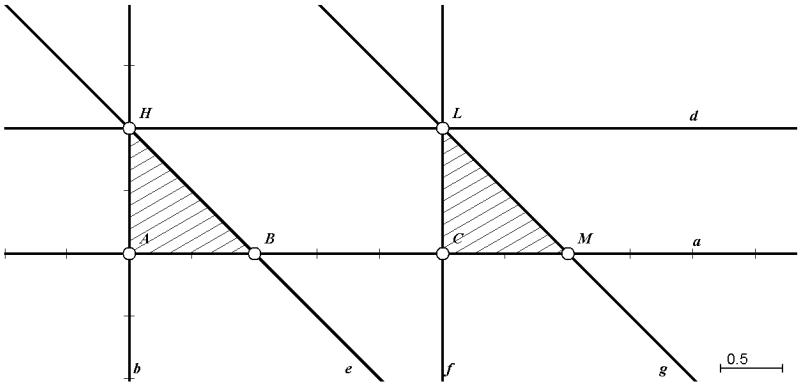


Figure 5.3: Von-Staudt Addition: Adding the x-coordinates of two points by constructing two congruent triangles.

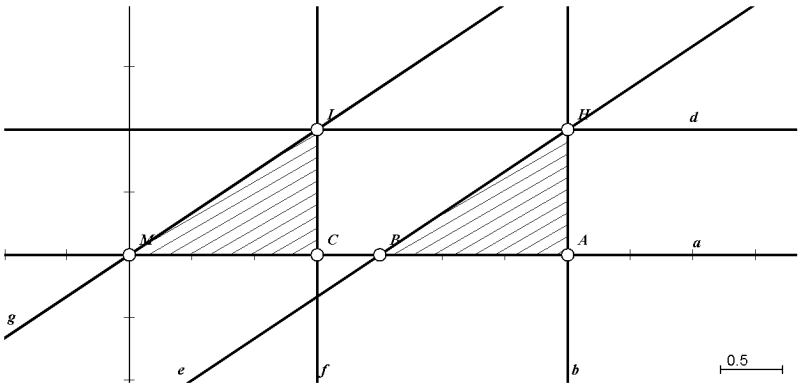


Figure 5.4: Von-Staudt Subtraction: Subtracting the x-coordinates of two points by constructing two congruent, but mirrored, triangles.

- the x-axis has coordinates $(0,1,0)$,
- the $x=1$-line has coordinates $(0,-1,1)$,
- the origin has coordinates $(0,0,1)$,
- the point $(1|0)$ has coordinates $(1,0,1)$, and
- the point $(0|1)$ has coordinates $(0,1,1)$.

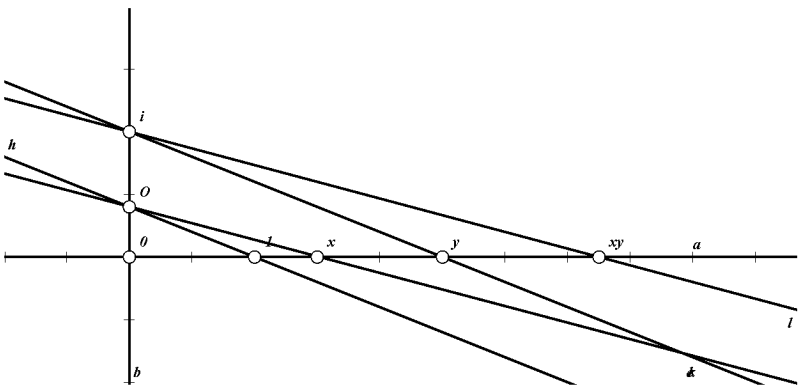


Figure 5.5: Von-Staudt Multiplication: Multiplying the x-coordinates of two points by transferring two ratios. If you reverse the construction you can also do divisions.

5.1.5 Abstract Point/Line-RIS GSPs and SLPs

Just for completeness we want to mention a way to avoid dealing with coordinates and still showing the "equivalence" of point/line-constructions and straight-line programs over a field $\mathbb{K}$. The necessary trick would be to add a cross ratio primitive to the abstract RIS for point/line-constructions. This cross ratio primitive would map four points A, B, C and D that lie on a line to the cross ratio $(AB|CD)$, a projective invariant that is very useful whenever you want to do measurements in projective space, see also section 5.4. The cross ratio is defined by taking the signed distances on the line between the points and dividing them:

$$CR(AB|CD) := \frac{|AC||BD|}{|AD||BC|}$$

You can also get the same cross ratio by using homogeneous coordinates on the line (two-dimensional) and taking determinants (lowercase letters denote the homogeneous coordinates of a point on a line):

$$CR(AB|CD) := \frac{\det(ac)\det(bd)}{\det(ad)\det(bc)}$$

Now we can use the cross ratio to extract the variables from the points on the x-axis. We only have to define a point "0," a point "1" and a point ∞ at infinity (observe that these points do not necessarily lie where you expect them, but can be placed arbitrarily), and the cross ratio $(0X|1\infty)$ is the "value" of point X.

We do not give the transformations explicitly here, it should be sufficient to know that you could do them also for abstract constructions. They are based on the same von-Staudt constructions as the homogeneous GSP to SLP transformation. It is just a matter of taste whether you want to work coordinate-free until the very last moment, or whether you introduce coordinates as early as possible since you need them anyway.

5.2 Determinism, Conservatism, Continuity

An implementation of a Dynamic Geometry system that supports points and lines in projective space is easy. Working with such a system raises certain expectations for Dynamic Geometry systems, which at the first look seem to be very reasonable.

The first apparent property is *determinism*: For any given input, there is at most one instance of a construction. We already proved this using the fact that all instructions in the homogeneous relational instruction set are determined. But remember that this is a special property – we will see that the RIS for a reasonable dynamic geometry system that can handle circles is not determined. Nevertheless, most other implemented Dynamic Geometry software shows deterministic behavior, and it is in fact what people expect.

The second property is a direct consequence of determinism, and for determined systems it is so natural that you probably will not notice that this is a distinguished behavior.

The point/line-constructions are *conservative*, meaning that when you move objects and then undo all your moves by reversing them, then you will have the same instance of the construction. If there is only one instance, like in a determined system, you will always have the same instance for the same set of input parameters, but in an indetermined system you might still expect conservatism. The reason why we explicitely mention conservatism is that *Cinderella* looks like being non-conservative (macroscopically), but actually it is conservative (microscopically). That is, every move is translated into a series of small moves, of which each is conservative. The twist comes in because a move in one direction is translated into another series of moves as the move in the opposite direction.

The third, and, as you will see, most challenging, property is *continuity*. We did not define a topology on our objects yet, but its clear what is meant by continuous movements: We do not want to have large "jumps" of elements for small changes in the input parameters (here we do not consider a point moving through infinity and coming back from the other side of the plane as a large jump, have a look at the spherical projection!).

These three observations are at the core of the problems arising in Dynamic Geometry. Of course, a user who starts to work with points and lines and experiencing the well-behaving world of Projective Geometry, expects the system to show the same behavior when other objects, for example circles, are introduced. See the next chapter for some examples, and let us concentrate on the reasons for the niceness of Projective Geometry.

We have shown in the last section that the homogeneous RIS GSPs can be written as ordinary straight-line programs over a field $\mathbb{K}$. This means that *all variables are given by multivariate polynomials in the input coordinates*, coordinate-wise. So here are the quick proofs for our observations:

Proof 5.9 (Point/Line-constructions are determined) *Since every coordinate of every point and line is a polynomial in the coordinates of the input points and lines, as is every variable calculated by scalar products (and determinants), there is only one instance for every GSP with given input, which we can find by evaluating the polynomials.* □

Proof 5.10 (Point/Line-constructions are conservative) *Since there is only one instance, we cannot end up in another instance after moving around.* □

For the last proof we have to fix a notion of continuity. We do not define a topology for $\mathbb{K}P^2 \cup \{(0,0,0)\}$, but if we did we ended up with the same continuity as the one below:

Definition 5.11 *A GSP on a homogeneous RIS is continuous if all coordinate functions are continuous functions in the input coordinates.*

Of course, this definition was made to make the following proof trivial:

Proof 5.12 (Point/Line-constructions are continuous) *All coordinate functions are polynomials in the input coordinates, so they are continuous.* □

You might wonder why we dwell on trivialities here. The reason is that these trivialities immediately become non-trivial in the next section, and it needs a lot of work to find out why they are suddenly that hard. For now you can just accept that everything is easy in the world of points and lines, but you should not draw the conclusion that it will stay like that.

5.3 Randomized Proving

In this section we will describe a way to prove geometric theorems automatically without using symbolic methods like Gröbner bases [77] or binomial proofs [66]. This method is very powerful, it is not restricted to *special* classes of point/line-incidence theorems, but it works for *any* point/line-incidence theorem. See also Sec. 5.3.5 for related results in that context.

The method uses the fact that the coordinate functions in homogeneous RIS GSPs are multivariate polynomials, and for these we can use a lot of known theory.

5.3.1 Testing Polynomials

Assume that you are given for a field $\mathbb{K}$ a polynomial $p \in \mathbb{K}[X]$, and you would like to determine whether it is the zero polynomial, that is, $p \equiv 0$. Clearly it depends on the representation of p how difficult this is. If you know all the non-zero coefficients c_i and

$$p(x) = \sum_{i=0}^{d} c_i x^n$$

then it is a trivial task: *All* coefficients must be zero for p being the zero polynomial.

But if the polynomial is described by a "black box," some device where you can insert a value for x and get the evaluation $p(x)$ of p at x then it is not as easy as above. You need more information about the polynomial to be able to tell anything. One particular useful piece of information is the maximum degree d of p, since the fundamental theorem of algebra (using the Euclidean algorithm) tells us that there are at most d distinct roots of p (see Thm. 4.4 in[56]), unless $p \equiv 0$. So if we are lucky and find more than d roots we have a certificate for the polynomial being identically zero.

This simple technique can be used to derive a Monte-Carlo method for checking zeroness of polynomials:

Theorem 5.13 (Zero testing for polynomials) *Let $p \in \mathbb{K}[X]$ be a univariate polynomial with degree $\leq d$. Choose a finite subset $S \subset \mathbb{K}$, and pick r uniformly at random from S. Then*

$$\mathbf{Pr}[p(r) = 0 \,|\, p(X) \not\equiv 0] \leq \frac{d}{|S|} \quad .$$

Proof 5.14 *There are at most d roots, which we divide by the number of possible values for r.* □

The probability that we pick a root "by accident," given that the p is not the zero polynomial, can be made very low. If the set S is chosen very large, we can make the probability of picking a root as low as any $\varepsilon > 0$, so in the limit case we can be sure just by one test. By repeating the zero test independently we can also easily lower the probability of a wrong answer.

The black boxes we will look at are division-free straight-line programs. These are not really black, since we can read the instructions, so we know a lot more than only the total degree. But it is not clear whether this additional knowledge can be of any use. Together with the connection of projective Dynamic Geometry to straight-line programs we can then apply the results to geometric constructions.

Since division-free straight-line programs model multivariate polynomials – the number of variables is the size of the input of the SLP – we have to extend Thm. 5.13 to the multivariate case.

5.3.2 The Schwartz-Zippel Theorem

Although the number of roots of a multivariate polynomial is usually infinite, consider $p(x,y) = xy$, the bound given by the following theorem is as good as in the univariate case. The trick is that we restrict ourselves on a subset of the input that is the cartesian product of subsets of the field. We could not choose just any subset of $\mathbb{K}^n$.

Theorem 5.15 (Schwartz-Zippel Theorem) *Let $Q(x_1,\ldots,x_n) \in \mathbb{K}[x_1,\ldots,x_n]$ be a multivariate polynomial of total degree d. Fix any finite subset $S \subset \mathbb{K}$, and let $r_1,\ldots,r_n$ be chosen independently and uniformly at random from S. Then*

$$\mathbf{Pr}[Q(r_1,\ldots,r_n) = 0 \,|\, Q(x_1,\ldots,x_n) \not\equiv 0] \leq \frac{d}{|S|} \quad .$$

Proof 5.16 *For an easy proof using induction on the number of variables see [60].*

If we know the maximal probability of finding a zero under the assumption that the polynomial is not the zero polynomial, then we can also prove that a polynomial is the zero polynomial by choosing a set S that is large enough for the probability to be below 1 and evaluating the polynomial for all $(r_1,\ldots,r_n) \in S^n$. This gives the following corollary about *test sets*.

Corollary 5.17 (Test Sets) *Let $Q(x_1,\ldots,x_n) \in \mathbb{K}[x_1,\ldots,x_n]$ be a multivariate polynomial of total degree less or equal d. Fix any finite subset $S \subset \mathbb{K}$ with $|S| > d$. If $Q(r_1,\ldots,r_n) = 0$ for all $(r_1,\ldots,r_n) \in S^n$ then $Q(x_1,\ldots,x_n) \equiv 0$.*

Proof 5.18 *The probability for Q evaluating to 0 in S^n under the assumption that $Q \not\equiv 0$ is less than $\frac{d}{|S|} < 1$ by Thm. 5.15. The evaluations prove that the probability is equal to 1, so the additional assumption of the Theorem cannot be true.* □

This corollary can be used to create a deterministic algorithm for zero checking of polynomials. If we have a bound for the total degree of a polynomial then we just can evaluate the polynomial over a large enough subset of $\mathbb{K}$ and we will either find a counter-example that shows that the polynomial is not equal to zero, or we will end up with enough examples that prove the zero identity.

5.3.3 The Test-Set Lemma

In some cases the total degree exceeds the single degree in each variable drastically. If we know the single degrees we can reduce the test set size as shown below in a lemma that appeared in [93].

Lemma 5.19 (Test Set Lemma) *Let $Q(x_1,\ldots,x_n) \in \mathbb{K}[x_1,\ldots,x_n]$ be a multivariate polynomial of degree in x_i less or equal d_i. Fix finite subsets $S_i \subset \mathbb{K}$ with $|S_i| > d_i$. If $Q(r_1,\ldots,r_n) = 0$ for all $(r_1,\ldots,r_n) \in S_1 \times \cdots \times S_n$ then $Q(x_1,\ldots,x_n) \equiv 0$.*

Proof 5.20 *We prove the lemma by induction on the number of variables n. For $n = 1$ the lemma is equivalent to Cor. 5.17, so let us assume that it is true for $n = j$ and prove it for $n = j+1$.*

Write Q as a univariate polynomial $\tilde{Q}$ in $x := x_{j+1}$. The coefficients c_i of $\tilde{Q}$ are multivariate polynomials in $x_1,\ldots,x_j$ with the induced degree bounds d_i. The degree of $\tilde{Q}$ is $d := d_{j+1}$.

Fix $(\hat{r}_1,\ldots,\hat{r}_j) \in S_1 \times \cdots \times S_j$, and consider the univariate polynomial $\hat{Q}(x) := Q(\hat{r}_1,\ldots,\hat{r}_j)$. This $\hat{Q}$ has degree at most d, and its coefficients are exactly the coefficients c_i evaluated at $(\hat{r}_1,\ldots,\hat{r}_j)$. Since $\hat{Q}(r) = 0$ for all $r \in S_{j+1}$ we know more than d roots of $\hat{Q}$, so $\hat{Q} \equiv 0$.

But we can choose any element of $S_1 \times \cdots \times S_j$ as $(\hat{r}_1,\ldots,\hat{r}_j)$, so we know by induction that all $c_i \equiv 0$, and we can conclude that $Q = \tilde{Q} \equiv 0$. □

Lemma 5.19 does not only imply Cor. 5.17, but it is much more powerful. Suppose you are given a polynomial in ten variables with degree two in each variable. If you are unlucky, you have to check 21^{10} instances when you rely on the total degree. When you use the refined version, you can do the same with 3^{10} instances, a factor of 282.475.249.

5.3.4 Automatic Theorem Proving for Constructive Point/Line-Incidence Theorems

Let us now mix what we know about homogeneous coordinates and zero testing of polynomials. We will get an automatic theorem prover for incidence theorems on points and lines in the projective plane.

For every homogeneous RIS GSP we will write down another GSP on another GIS that gives an estimate for the degree in each input variable of the polynomials describing the intermediate results of the GSP. With this estimate we can apply the test-set Lemma 5.19.

Let us collect all the necessary notations and definitions for this.

Notation 5.21 (Multidegree of a polynomial) *For a multivariate polynomial* $p \in \mathbb{K}[X_1,\ldots,X_n], p \neq 0$ *let* $\deg(p) = (d_1,\ldots,d_n)$ *denote the degree in each variable.*

Definition 5.22 (Partial order on multidegrees) *A multidegree* $(c_1,\ldots,c_n)$ *is less or equal to* $(d_1,\ldots,d_n)$ *if* $c_i \leq d_i$ *for all* $i \in \{1,\ldots,n\}$. *In that case we write* $(c_1,\ldots,c_n) \preceq (d_1,\ldots,d_n)$.

Definition 5.23 (Sum of multidegrees) *For two multidegrees* $(c_1,\ldots,c_n)$ *and* $(d_1,\ldots,d_n)$ *their* sum $(c_1,\ldots,c_n)+(d_1,\ldots,d_n)$ *is defined by*

$$(c_1+d_1,\ldots,c_n+d_n).$$

Definition 5.24 (Maximum of multidegrees) *For two multidegrees* $(c_1,\ldots,c_n)$ *and* $(d_1,\ldots,d_n)$ *the maximum multidegree* $\max\{(c_1,\ldots,c_n),(d_1,\ldots,d_n)\}$ *is the multidegree*

$$(\max\{c_1,d_1\},\ldots,\max\{c_n,d_n\}).$$

A few basic observations help us in finding the upper bounds for the multidegrees of the coordinate polynomials. When we multiply polynomials, the degrees will be added, when we add polynomials, we must take the maximum degree of each to receive the multidegree.

Lemma 5.25 (Bounds on the multidegree) *Let* $p,q \in \mathbb{K}[X_1,\ldots,X_n]$ *be two polynomials with* $\deg(p) \preceq (c_1,\ldots,c_n)$ *and* $\deg(q) \preceq (d_1,\ldots,d_n)$. *Then*

1. $\deg(pq) \preceq (c_1+d_1,\ldots,c_n+d_n)$,
2. $\deg(p+q) \preceq \max\{\deg(p),\deg(q)\}$,
3. $\deg(p-q) \preceq \max\{\deg(p),\deg(q)\}$.

Proof 5.26 *The degree bounds on p and q show that we can write*

$$p(x_1,\ldots,x_n) = \sum_{j_1=0}^{c_1} \cdots \sum_{j_n=0}^{c_n} \alpha_{j_1,\ldots,j_n} x_1^{j_1} \cdot \cdots \cdot x_n^{j_n} \quad \textit{and} \tag{5.4}$$

$$q(x_1,\ldots,x_n) = \sum_{k_1=0}^{d_1} \cdots \sum_{k_n=0}^{d_n} \beta_{k_1,\ldots,k_n} x_1^{k_1} \cdot \cdots \cdot x_n^{k_n}. \tag{5.5}$$

1. *Multiplying equations 5.4 and 5.5 gives*

$$p(x_1,\ldots,x_n)q(x_1,\ldots,x_n) = \sum_{j_1=0}^{c_1}\cdots\sum_{j_n=0}^{c_n}\sum_{k_1=0}^{d_1}\cdots\sum_{k_n=0}^{d_n}\alpha_{j_1\ldots j_n}\beta_{k_1\ldots k_n}x_1^{j_1+k_1}\cdot\ldots\cdot x_n^{j_n+k_n}.$$

   The bound on the multidegree follows immediately from this representation.

2. *The sum of p and q is given by*

$$p(x_1,\ldots,x_n)+q(x_1,\ldots,x_n) = \sum_{j_1=0}^{\max\{c_1,d_1\}}\cdots\sum_{j_n=0}^{\max\{c_n,d_n\}}(\alpha_{j_1\ldots j_n}+\beta_{j_1\ldots j_n})x_1^{j_1}\cdot\ldots\cdot x_n^{j_n}.$$

   The bound on the multidegree follows immediately from this representation.

3. *We can apply the same formula as for the addition multidegree bound since* $\deg(-q) = \deg(q)$ *for all multivariate polynomials q.*

□

Notation 5.27 (Multidegree of a vector) *For a vector* $(p_1,p_2,p_3)^T$ *of multivariate polynomials we introduce the shortcut* $\deg((p_1,p_2,p_3)^T)$ *for the vector of multidegrees* $(\deg(p_1),\deg(p_2),\deg(p_3))^T$

Lemma 5.28 (Multidegree bounds for determinants of polynomials) *Let* $p_1,p_2,p_3,q_1,q_2,q_3,r_1,r_2,r_3 \in \mathbb{K}[X_1,\ldots,X_n]$ *be multivariate polynomials with* $\deg(p_i) \preceq (c_{i1},\ldots,c_{in})$, $\deg(q_i) \preceq (d_{i1},\ldots,d_{in})$ *and* $\deg(r_i) \preceq (e_{i1},\ldots,e_{in})$. *Then we can give the following upper bounds for operations on these polynomials.*

1. *The multidegree of a* 2×2*-determinant is bounded by*

$$\begin{aligned}\deg\begin{vmatrix} p_1 & q_1 \\ p_2 & q_2 \end{vmatrix} &= \deg(p_1q_2-p_2q_1)\\ &\preceq \max\{\deg(p_1q_2),\deg(p_2q_1)\}\\ &\preceq \max\{\deg(p_1)+\deg(q_2),\deg(p_2)+\deg(q_1)\}\end{aligned}$$

2. *The multidegree of a each coordinate of a cross product of two vectors of multivari-*

ate polynomials is bounded by

$$\deg\left(\begin{pmatrix}p_1\\p_2\\p_3\end{pmatrix}\times\begin{pmatrix}q_1\\q_2\\q_3\end{pmatrix}\right) \preceq \deg\begin{pmatrix}\begin{vmatrix}p_2 & q_2\\ p_3 & q_3\end{vmatrix}\\ -\begin{vmatrix}p_1 & q_1\\ p_3 & q_3\end{vmatrix}\\ \begin{vmatrix}p_1 & q_1\\ p_2 & q_2\end{vmatrix}\end{pmatrix}$$

$$= \begin{pmatrix}\max\{\deg(p_2)+\deg(q_3),\\ \deg(p_3)+\deg(q_2)\}\\ \max\{\deg(p_3)+\deg(q_1),\\ \deg(p_1)+\deg(q_3)\}\\ \max\{\deg(p_1)+\deg(q_2),\\ \deg(p_2)+\deg(q_1)\}\end{pmatrix}$$

3. *The multidegree of the scalar product of two vectors is bounded by*

$$\deg\left((p_1,p_2,p_3)\begin{pmatrix}q_1\\q_2\\q_3\end{pmatrix}\right) \preceq \begin{array}{c}\max\{\max\{\deg(p_1)+\deg(q_1),\\ \deg(p_2)+\deg(q_2)\},\\ \deg(p_3)+\deg(q_3)\}\end{array}$$

4. *The degree of a* 3×3*-determinant of polynomials is bounded by*

$$\deg\begin{vmatrix}p_1 & q_1 & r_1\\ p_2 & q_2 & r_2\\ p_3 & q_3 & r_3\end{vmatrix} \preceq \deg\left(\left(\begin{pmatrix}p_1\\p_2\\p_3\end{pmatrix}\times\begin{pmatrix}q_1\\q_2\\q_3\end{pmatrix}\right)^T\begin{pmatrix}r_1\\r_2\\r_3\end{pmatrix}\right)$$

$$\preceq \max\{$$
$$\max\{\deg(p_2)+\deg(q_3)+deg(r_1),$$
$$\deg(p_3)+\deg(q_2)+deg(r_1)\},$$
$$\max\{\deg(p_3)+\deg(q_1)+deg(r_2),$$
$$\deg(p_1)+\deg(q_3)+deg(r_2)\},$$
$$\max\{\deg(p_1)+\deg(q_2)+deg(r_3),$$
$$\deg(p_2)+\deg(q_1)+deg(r_3)\}\}$$

Proof 5.29 *All the bounds are immediate from the preceding lemmas.* □

Now we are set for the transformation of the homogeneous RIS GSP to an GSP over the *bounding RIS*, which can carry out maximum operations and additions on multidegrees.

Definition 5.30 (Bounding RIS) *A* bounding RIS *is a relational instruction set whose objects O_1 are multidegree vectors $(d_1,\ldots,d_n) \in \mathbb{N}_0^n$ with two primitives,* MAX^2 *and* $+^2$*, where*

$$\begin{aligned} \mathsf{MAX}^2 &= \{(a,b,c)\ s.t.\ c = \max\{a,b\}\} \subset O_1^3 \\ +^2 &= \{(a,b,c)\ s.t.\ c = a+b\} \subset O_1^3 \end{aligned}$$

The transformation from the original GSP on the homogeneous RIS to the GSP on the bounding RIS uses the formulas from Lemma 5.28.

Transformation 5.31 (Homogeneous RIS GSP to Bounding RIS GSP) *Let (X,R,Γ) be a GSP on a homogeneous RIS, $X = (X_1,\ldots,X_n)$, $\Gamma = (\Gamma_1,\ldots,\Gamma_m)$. The new GSP on the bounding RIS will be $(\tilde{X},\tilde{R},\tilde{\Gamma})$.*

The input $\tilde{X}$ of the new GSP is constructed by taking for every variable X_i of type O_1 (the homogeneous coordinate vectors) three variables $\tilde{X}_{i1},\tilde{X}_{i2}$ and $\tilde{X}_{i3}$, and for every variable X_j of type O_2 (the scalars) one variable $\tilde{X}_j$. The input variables will be multidegrees with the same number of elements as the size of the input of the new GSP.

The statements of the bounding RIS GSP are obtained by translating every statement of the original GSP conforming to this table, the translation for the Det *primitive is obtained by first translating to a determinant-free GSP and then using the* Cross *and* Scal *translations:*

Homogeneous RIS GSP	$\leftrightarrow$	*Bounded RIS GSP code*			
k: Cross i j	$\leftrightarrow$	$k,-5$:	$+$	$i,2$	$j,3$
		$k,-4$:	$+$	$i,3$	$j,2$
		$k,-3$:	$+$	$i,3$	$j,1$
		$k,-2$:	$+$	$i,1$	$j,3$
		$k,-1$:	$+$	$i,1$	$j,2$
		$k,0$:	$+$	$i,2$	$j,1$
		$k,1$:	Max	$k,-5$	$k,-4$
		$k,2$:	Max	$k,-3$	$k,-2$
		$k,3$:	Max	$k,-1$	$k,0$
k: Scal i j	$\leftrightarrow$	$k,-3$:	$+$	$i,1$	$j,1$
		$k,-2$:	$+$	$i,2$	$j,2$
		$k,-1$:	$+$	$i,3$	$j,3$
		$k,0$:	Max	$k,-3$	$k,-2$
		k:	Max	$k,0$	$k,-1$

What can we do with this transformation? It is meant to have an easy way to get a bound on the degree in each variable of every intermediate result of a GSP. We know the multidegree of each input element, it is one for the variable corresponding to the

coordinate of the input, and zero for all other variables. If we feed this into the transformed GSP, we will get the best possible trivial upper bound for the degree of each variable of the polynomial described by the original GSP.

Definition 5.32 (Standard Input of a Bounding RIS) *For a bounding RIS on multidegrees $(d_1,\ldots,d_n)$ of length n the standard input is*

$$\begin{aligned} X_1 &= (1,0,0,0,\ldots,0) \\ X_2 &= (0,1,0,0,\ldots,0) \\ X_3 &= (0,0,1,0,\ldots,0) \\ &\vdots \quad\quad \vdots \\ X_n &= (0,0,0,\ldots,0,1) \end{aligned}$$

The transformation to bounding RIS GSPs is a universal tool that we can use to prove constructive incidence theorems in Projective Geometry involving points and lines. Informally spoken, a constructive incidence theorem is a theorem of the form "In a projective point/line construction given by following construction steps this point/line-incidence has to occur." The exact definition of constructive incidence theorem is based on homogeneous RIS GSPs:

Definition 5.33 (Constructive Incidence Statement) *A* constructive incidence statement *is a GSP (X,R,Γ) with input size n and length m on a homogeneous RIS with the following properties:*

1. *All input variables are of type O_1, i.e. they stand for homogeneous vectors.*
2. *All intermediate results are of type O_1, except for R_{m-1}, which is of type O_2. The variable R_{m-1} is called* conclusion.

The conclusion of a constructive incidence statement is, since it is of the scalar type, created by a Scal or Det primitive, all other statements must use the Cross primitive. So the conclusion could be stated in human language as "if we construct like that, then in the end this incidence will always be true" or "if we construct like that, then in the end these three points/lines will always be collinear/concurrent." This "always be" is coded by asking for the polynomial of the last intermediate result R_{m-1}:

Definition 5.34 (Truth of a Constructive Incidence Statement) *A constructive incidence statement of length m is* true *if all instances of it have $R_{m-1}=0$, i.e. the polynomial encoded by R_{m-1} is the zero polynomial, and it is* false *if there exists one instance with $R_{m-1}\neq 0$. In that case we speak of a* constructive incidence theorem.

Of course, there are statements that are always true because they degenerate before evaluating the conclusion, one input of the last statement is always the zero vector.

Definition 5.35 (Degenerate Constructive Incidence Theorem) *A constructive incidence theorem is* degenerate *if one input of the last statement is the zero vector for all instances of the constructive incidence statement.*

Degenerate constructive incidence theorems are always true, since the scalar product and the determinant evaluate to zero when one of the inputs is the zero vector.

With all this equipment we can give a simple algorithm for theorem proving:

Algorithm 5.36 (Theorem Proving for Constructive Incidence Theorems)
Input: A constructive incidence statement (X,R,Γ). of length m
Output: Either `TRUE` if the constructive incidence statement is true, or a counter-example.

1. Transform (X,R,Γ) to the GSP $(\tilde{X},\tilde{R},\tilde{\Gamma})$ on a bounding RIS as in transformation 5.31.
2. Evaluate $(\tilde{X},\tilde{R},\tilde{\Gamma})$ with the standard input for bounding RIS GSPs to get the multidegree of R_{m-1}.
3. Choose a test-set using the multidegree according to Lemma 5.19.
4. Feed all elements of the test-set into the original GSP. If the evaluation of R_{m-1} is non-zero for at least one element of the test-set, return it as a counter-example for the statement. Else return `TRUE`.

The correctness of the algorithm follows immediately from the test-set Lemma 5.19 and the bounds on the multidegrees for cross and scalar products of vectors of polynomials as in Lemma 5.28.

Example 5.37 (Pappos' theorem) *We will prove Pappos' theorem using algorithm 5.36. First of all we have to find a construction sequence that encodes Pappos' theorem as a constructive incidence theorem. We formulate example 4.8 as an homogeneous RIS GSP and add a conclusion, and then we transform it and evaluate it on the standard input. See table 5.1 for the construction and the bound on the multidegree.*

Next we only have to evaluate 4^{15} *(roughly a billion) examples of Pappos' theorem and when we do not find a counter-example this will be a proof of the theorem.*

The problem of this method of proving theorems is that the degree bounds can increase very quickly, actually it can happen that they double in each construction step. So although we have efficient methods of zero-testing polynomials, we have to check an enormous amount of examples to have a valid proof.

Here the power of randomized proving can be exploited in its full strength: The bound on the multidegree can be used trivially as a bound on the total degree by adding all single degrees (in the case of example 5.37 the total degree will be less than or equal to $15 \cdot 3 = 45$. So using the Schwartz-Zippel theorem 5.15 we know that if we pick one random example from a set of size $(2^{32})^{15}$ and this evaluates to zero, then the probability that Pappos' theorem is false is less than $\frac{45}{2^{32}}$, only slightly more than 0.000001%.

Index	Var/Result	Statement
-5	X_5	–
-4	X_4	–
-3	X_3	–
-2	X_2	–
-1	X_1	–
0	R_0	Cross -5,-4
1	R_1	Cross -4,-3
2	R_2	Cross -3,-2
3	R_3	Cross -2,-1
4	R_4	Cross 0,2
5	R_5	Cross 1,3
6	R_6	Cross -5,5
7	R_7	Cross -1,4
8	R_8	Cross 6,7
9	R_9	Cross -5,-2
10	R_{10}	Cross -1,-4
11	R_{11}	Cross 9,10
12	R_{12}	Cross 8,-3
13	R_{13}	Scal 11,12

Index	Multidegree
-5.1	(0,0,0,0,0,0,0,0,0,0,1,0,0)
-5.2	(0,0,0,0,0,0,0,0,0,0,0,1,0)
-5.3	(0,0,0,0,0,0,0,0,0,0,0,0,1)
-4.1	(0,0,0,0,0,0,0,0,1,0,0,0,0)
-4.2	(0,0,0,0,0,0,0,0,0,1,0,0,0)
-4.3	(0,0,0,0,0,0,0,0,0,0,1,0,0)
-3.1	(0,0,0,0,0,0,1,0,0,0,0,0,0)
-3.2	(0,0,0,0,0,0,0,1,0,0,0,0,0)
-3.3	(0,0,0,0,0,0,0,0,1,0,0,0,0)
-2.1	(0,0,0,1,0,0,0,0,0,0,0,0,0)
-2.2	(0,0,0,0,1,0,0,0,0,0,0,0,0)
-2.3	(0,0,0,0,0,1,0,0,0,0,0,0,0)
-1.1	(1,0,0,0,0,0,0,0,0,0,0,0,0)
-1.2	(0,1,0,0,0,0,0,0,0,0,0,0,0)
-1.3	(0,0,1,0,0,0,0,0,0,0,0,0,0)
0.1	(0,0,0,0,0,0,0,0,0,1,1,0,1,1)
0.2	(0,0,0,0,0,0,0,0,0,1,0,1,1,0,1)
0.3	(0,0,0,0,0,0,0,0,0,1,1,0,1,1,0)
1.1	(0,0,0,0,0,0,0,0,1,1,0,1,1,0,0,0)
1.2	(0,0,0,0,0,0,0,1,0,1,1,0,1,0,0,0)
1.3	(0,0,0,0,0,0,0,1,1,0,1,1,0,0,0,0)
2.1	(0,0,0,0,1,1,0,1,1,0,0,0,0,0,0)
2.2	(0,0,0,1,0,1,1,0,1,0,0,0,0,0,0)
2.3	(0,0,0,1,1,0,1,1,0,0,0,0,0,0,0)
3.1	(0,1,1,0,1,1,0,0,0,0,0,0,0,0,0)
3.2	(1,0,1,1,0,1,0,0,0,0,0,0,0,0,0)
3.3	(1,1,0,1,1,0,0,0,0,0,0,0,0,0,0)
4.1	(0,0,0,1,1,1,1,1,1,1,1,1,1,1,1)
4.2	(0,0,0,1,1,1,1,1,1,1,1,1,1,1,1)
4.3	(0,0,0,1,1,1,1,1,1,1,1,1,1,1,1)
5.1	(1,1,1,1,1,1,1,1,1,1,1,0,0,0,0)
5.2	(1,1,1,1,1,1,1,1,1,1,1,1,0,0,0)
5.3	(1,1,1,1,1,1,1,1,1,1,1,1,0,0,0)
6.1	(1,1,1,1,1,1,1,1,1,1,1,1,0,1,1)
6.2	(1,1,1,1,1,1,1,1,1,1,1,1,1,0,1)
6.3	(1,1,1,1,1,1,1,1,1,1,1,1,1,1,0)
7.1	(0,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
7.2	(1,0,1,1,1,1,1,1,1,1,1,1,1,1,1)
7.3	(1,1,0,1,1,1,1,1,1,1,1,1,1,1,1)
8.1	(2,2,2,2,2,2,2,2,2,2,2,2,2,2,2)
8.2	(2,2,2,2,2,2,2,2,2,2,2,2,2,2,2)
8.3	(2,2,2,2,2,2,2,2,2,2,2,2,2,2,2)
9.1	(0,0,0,0,1,1,0,0,0,0,0,0,1,1,1)
9.2	(0,0,0,1,0,1,0,0,0,0,0,1,0,1,1)
9.3	(0,0,0,1,1,0,0,0,0,0,0,1,1,1,0)
10.1	(0,1,1,0,0,0,0,0,0,1,1,0,0,0,0)
10.2	(1,0,1,0,0,0,0,0,1,0,1,0,0,0,0)
10.3	(1,1,0,0,0,0,0,0,1,1,0,0,0,0,0)
11.1	(1,1,1,1,1,1,0,0,1,1,1,1,1,1,1)
11.2	(1,1,1,1,1,1,0,0,1,1,1,1,1,1,1)
11.3	(1,1,1,1,1,1,0,0,1,1,1,1,1,1,1)
12.1	(2,2,2,2,2,2,3,3,2,2,2,2,2,2,2)
12.2	(2,2,2,2,2,2,3,2,3,2,2,2,2,2,2)
12.3	(2,2,2,2,2,2,3,3,2,2,2,2,2,2,2)
13	(3,3,3,3,3,3,3,3,3,3,3,3,3,3,3)

Table 5.1: On the left you can read off the construction steps for Pappos' Theorem, on the right the evaluated multidegree bound is shown for every intermediate result.

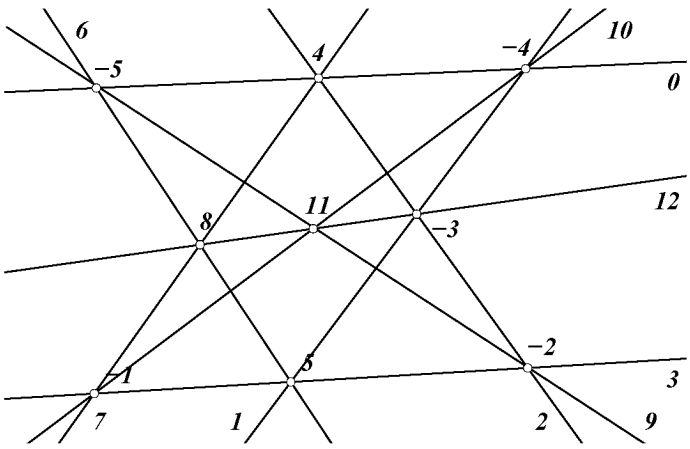


Figure 5.6: The configuration corresponding to table 5.1

In [74] Schwartz discusses how the probability can be kept low even if all calculations are carried out in the integers modulo some random prime, which solves all the numerical problems that can arise while evaluating a homogeneous geometric straight-line program. However, we will not discuss this here.

We would like to end this section with two remarks concerning this method of proving incidence theorems.

Remark 5.38 *There are non-constructive incidence theorems.*

This is an obstacle when you want to generally apply the theorem proving algorithm for incidence theorems. There is no algorithm that constructs construction sequences for theorems given in hypotheses/conclusion-form automatically, and there cannot be a general one, since there are theorems with no corresponding construction sequence.

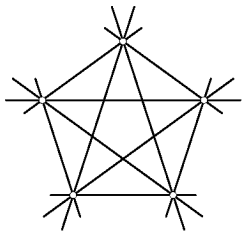


Figure 5.7: A regular pentagon and its diagonals.

Here is one example: If you construct the five intersections of the lines that are given by each edge and a non-adjacent diagonal in a pentagon, then if four of these are on a line, then will be the fifth. This is true for a regular pentagon, where these intersections lie on the line at infinity, see Fig. 5.7, and thus it is true for every projective transform of the regular pentagon. But whenever we have an arbitrary pentagon, which is not the projection of a regular one, then there will be no four of these points constructed above that lie on a line. So the assumption of the theorem is only valid for projections of regular pentagons. Since these need non-rational coordinates (in any projection!) the theorem is non-constructive.

Even worse: There are theorems that are not constructible even if you admit other tools like a compass! These appear in configurations that cannot be constructed since the placement of some points involves solving polynomial equations of degree higher than 4.

Another issue is the efficiency of the proving algorithm. As mentioned above the degree bound cannot guarantee polynomiality of the proving algorithm. If we need a polynomial time algorithm (polynomial in the size of the construction), we must use the randomized variant.

5.3.5 Related Results

The idea of using zero-testing of polynomials in geometric theorem proving is not new, it is known as the method of "proving by examples" or "parallel numerical verification" and was developed by Mike Deng, Lu Yang, Jingzhong Zhang and their co-workers [10, 90, 91, 93] based on work of Jiawei Hong [28, 29]. Hong gave criteria to determine whether a particular example (evaluation) is a proof for a polynomial being the zero polynomial, in fact, if the value you plug into the polynomial is large enough – where "large enough" is the critical part – one evaluation is sufficient. The parallel method concentrates on evaluating several examples instead of only one. The adjective "parallel" comes from the

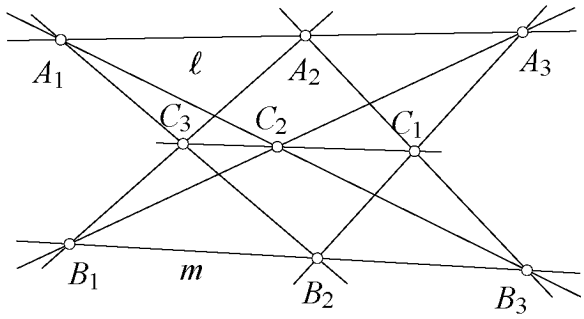


Figure 5.8: Pappos' theorem in general position: A_1, A_2 and A_3 are pairwise distinct, as are B_1, B_2 and B_3. We want to prove that the C_i are collinear.

fact that you can carry out the necessary evaluations of polynomials in parallel, since they are independent. Nevertheless, so far there is no implementation known to us that actually parallelizes the computations.

The main difference of the former approaches is that the step of finding the polynomials corresponding to the geometric theorem is not included. Instead the authors rely on other ways to find a polynomial system that has to be checked, e.g. Wu's method (after a suitable coordinatization) that was introduced in [89]. So a set of *hypotheses* is created and a *conclusion*, and a first obstacle is to create one polynomial equation by elimination techniques. In our situation we can use the given construction sequence to create the polynomials automatically. However, proving by examples in general is more powerful, since we can only create the construction sequences automatically as long as only points and lines in projective space are involved. See the next chapter for the reasons and the way out of this dilemma.

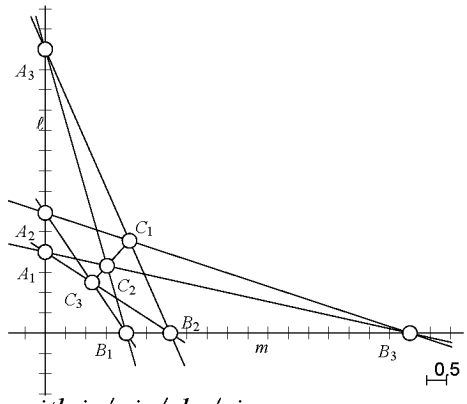
We would like to present one particularly nice example of the "proving by examples" method that shows how the method can be applied even if we do not use a computer, and how we can actually avoid any evaluations by simple arguments. This is not an example for automatic theorem proving, but it is an example how the methods of automatic theorem proving can be enhanced when we do not work with general bounds for the degrees but with concrete configurations. This proof was presented in [93].

Example 5.39 (Pappos' Theorem using the parallel numerical method) *Let ℓ, m be two distinct lines. Let A_1, A_2, A_3 be three points on ℓ and B_1, B_2, B_3 be three points on m. The three intersections of A_iB_j and A_jB_i, $i \neq j \in \{1,2,3\}$, are denoted by C_3, C_2 and C_1 as in Fig. 5.8, and we want to show that they are collinear.*

After a projective transformation we can assume that ℓ is the x-axis and m is the y-axis, and no point lies on the line at infinity. So the dehomogenized coordinates of the A_i are $(u_i, 0)$, and the coordinates of the B_i are $(0, v_i)$. By (x_i, y_i) we denoted the coordinates of the C_i.

The hypotheses are the equations that describe the position of the C_i with respect to the A_i and B_i, so we have

$$H_1: \quad v_2x_3 + u_1y_3 - u_1v_2 = 0,$$
$$H_2: \quad v_1x_3 + u_2y_3 - u_2v_1 = 0,$$
$$H_3: \quad v_3x_2 + u_1y_2 - u_1v_3 = 0,$$
$$H_4: \quad v_1x_2 + u_3y_2 - u_3v_1 = 0,$$
$$H_5: \quad v_3x_1 + u_2y_1 - u_2v_3 = 0, \text{ and}$$
$$H_6: \quad v_2x_1 + u_3y_1 - u_3v_2 = 0,$$



that are created by the collinearities $A_iB_jC_k$ with $i \neq j \neq k \neq i$.
The conclusion is described analogously by

$$C_1: \quad \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0.$$

Now we try to estimate the degree of the conclusion polynomial. Using pairs of hypotheses we can describe the dependent variables by fractions

$$x_k = \frac{L_{ij}}{N_{ij}}, \quad y_k = \frac{M_{ij}}{N_{ij}}$$

where L_{ij}, M_{ij}, N_{ij} are polynomials in u_i, u_j, v_i, v_j of a degree less or equal to 1 in each variable. So the conclusion can be written as, after eliminating the common denominators,

$$\Phi = \begin{vmatrix} L_{23} & M_{23} & N_{23} \\ L_{13} & M_{13} & N_{13} \\ L_{12} & M_{12} & N_{12} \end{vmatrix} = 0,$$

where the degree of Φ in each variable is less than 3. So we can choose $S := \{0,1,2\}^6$ as a test set for the theorem (using Lemma 5.19).

But, if $A_i = 0$ or $B_i = 0$, then the theorem is obviously true, so we only have to check it for $S' := \{1,2\}^6$. But now two of the A_i and two of the B_i have to coincide, so two of the C_i are coincident, too, which means that the theorem is trivially true for all test cases. □

5.4 Measurements

Most people think that Projective Geometry and measurements are two different worlds that cannot be merged. Interestingly this was noted already by Felix Klein in his "Vorlesungen über nicht-euklidische Geometrie" [42], and unfortunately it is still true today:

> " ... weil sich die Geometer an den Gedanken gewöhnt hatten, daß Metrik und projektive Geometrie in keiner Beziehung zueinander ständen."

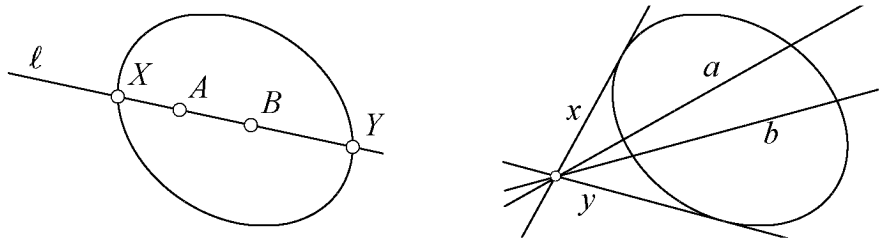


Figure 5.9: Measuring distances and angles in Cayley-Klein geometries

The way out of the non-existing dilemma is completely formalized by *Cayley-Klein geometries*. We will not discuss this wonderful part of geometry here in full detail, but for a better understanding we just want to give a rough sketch of how measurements can be expressed in Projective Geometry.

The first thing you need is a dual pair of conics that play the role of infinity: Two conics C and D are said to be a dual pair if

$$\mu C = D^* \qquad \text{and} \qquad \lambda D = C^*$$

for some μ and λ. Here C^* is the adjoint of a conic. If the matrix of C is invertible, then $C^* = \det(C)C^{-1}$, but the formal definition also works for matrices that are not invertible:

$$C^* = \big(\det(C_{ij})\big)_{ij}$$

Here C_{ij} is the matrix C missing row i and column j. In a dual pair (C,D) we call C the *fundamental conic* and D is its *dual*.

A dual pair (C,D) has the property that every point that lies on C is tangent to D if we interpret it as a line, and vice-versa. This extends the point/line duality to conics and its duals. We need this property in order to measure angles, see below, because we have to calculate tangents to the fundamental conic.

All measurements are now carried out with respect to that fundamental pair of conics, and the key operation here is the cross ratio of four points or four lines, as introduced in section 5.1.5.

You might wonder why the cross ratio is a suitable way to do measurements, since it is a projective invariant, which is definitely not the case for, say, Euclidean distances. But here the trick is that we are not measuring the distance between four objects, but only two objects, and the missing two ones for the cross ratio will be calculated from the first two ones and the fundamental conic. Thus the transformations that are invariant with respect to distances or angles are exactly those transformations that map the fundamental conic and its dual to itself.

Here is the recipe for distances:

- Fix a fundamental pair of conics (C,D).

- For two points A and B take the join $\ell = A \vee B$ and intersect it with the fundamental conic C. Call the intersections X and Y.
- Calculate the cross ratio $CR(AB|XY)$, which is possible since all four points lie in a 1-dimensional subspace (on a line).
- Take the logarithm of the cross ratio and (if you like) multiply it with some cosmetic constant c_{dist}. Call the result "distance."

And the analogous (polar) procedure for angles:

- Fix the same fundamental pair of conics as for distance measurements.
- For two lines a and b take the meet $M = a \wedge b$ and let the tangents to the fundamental conic through M be x and y.
- Calculate the cross ratio $CR(ab|xy)$, which is possible since all four lines meet in a point.
- Take the logarithm of the cross ratio and (if you like) multiply it with some cosmetic constant c_{ang}. Call the result "angle."

These simple formulas

$$\text{dist}(AB) = c_{dist} \ln(CR(AB|XY)) \text{ and } \angle ab = c_{ang} \ln(CR(ab|xy))$$

unify the measurements in Euclidean, hyperbolic, elliptic, relativistic and some other types of geometry. You can classify the geometries via the projective equivalence of fundamental conics and dual fundamental conics.

More material on Cayley-Klein geometries in Dynamic Geometry software can be found in [87] and [49].

Chapter 6

Circles and Conics

The last chapter presented the easy part of Projective Geometry on a computer. The implementation of a software that is based on Ch. 5 is straight-forward. Since we could fall back on mathematics of the last century we can expect that within the last one hundred years enough research has been carried out to cover also what is needed to add circles and conics to the setup of dynamic projective geometry. It turns out (in the next chapter) that there actually is enough theory to make also conics and circles available, but we will have to rediscover it.

6.1 Extending point/line-constructions

We will now extend the homogeneous coordinates approach for points and lines to cover also algebraic curves of degree 2, conic sections.

6.1.1 Representation of Conics

Definition 6.1 (Conics in homogeneous coordinates) *A* conic *in the projective plane* $\mathbb{R}P^2$ *is given by a homogeneous equation of degree 2:*

$$C := \{(x,y,z) \; s.t. \; ax^2 + bx^2 + cx^2 + exy + fxz + gyz = 0\} \tag{6.1}$$

Another common representation of a conic can be given by a 3×3-matrix A:

$$C := \{(x,y,z) \text{ s.t. } (x,y,z)A(x,y,z) = 0\} \tag{6.2}$$

Clearly this representation is not unique, even if we identify scalar multiples of A, but with the additional restriction on A to be symmetric we can read off the coefficients of Eq. 6.1 from A.

6.1.2 Basic operations

We will now define a set of basic operations that we definitely expect from a Dynamic Geometry system. Although this set is very small, all problems will already arise within this limited framework. For a much more detailed discussion of the possible operations on points, lines and conics see [87]. Another good source will be [4], which is partly based on [68].

We will explicitly describe how the necessary calculations for the basic operations can be carried out in homogeneous coordinates.

Conic by five points

For interactive manipulations the definition of a conic by its matrix is not very suitable. It is much more preferable to be able to visually manipulate a conic via a direct connection of a point on screen to the entries of the matrix.

Five points define a unique conic, if no four of them are on a common line. So we introduce a primitive Conic that uses five points as input and outputs the conic defined by these five points, and the zero matrix in the degenerate case.

The conic equation can be calculated using the five equations that are given by the point/conic incidences. A general conic can be written as

$$a \cdot x^2 + b \cdot y^2 + c \cdot z^2 + d \cdot xy + e \cdot xz + f \cdot yz = 0. \tag{6.3}$$

Using the known coordinates (x_i, y_i, z_i) of the points on the conic, $i = 1, \ldots, 5$, assuming that (x, y, z) describes another point on the conic, and regarding the coefficients $a, \ldots, f$ as unknowns we get a *linear* equation system

$$\begin{aligned}
x_1^2 \cdot a + y_1^2 \cdot b + z_1^2 \cdot c + x_1 y_1 \cdot d + x_1 z_1 \cdot e + y_1 z_1 \cdot f &= 0 \\
x_2^2 \cdot a + y_2^2 \cdot b + z_2^2 \cdot c + x_2 y_2 \cdot d + x_2 z_2 \cdot e + y_2 z_2 \cdot f &= 0 \\
x_3^2 \cdot a + y_3^2 \cdot b + z_3^2 \cdot c + x_3 y_3 \cdot d + x_3 z_3 \cdot e + y_3 z_3 \cdot f &= 0 \\
x_4^2 \cdot a + y_4^2 \cdot b + z_4^2 \cdot c + x_4 y_4 \cdot d + x_4 z_4 \cdot e + y_4 z_4 \cdot f &= 0 \\
x_5^2 \cdot a + y_5^2 \cdot b + z_5^2 \cdot c + x_5 y_5 \cdot d + x_5 z_5 \cdot e + y_5 z_5 \cdot f &= 0 \\
x^2 \cdot a + y^2 \cdot b + z^2 \cdot c + x\ y\ \cdot d + x\ z\ \cdot e + y\ z\ \cdot f &= 0
\end{aligned}$$

This equation system has a non-trivial solution, so for the determinant of the coefficients

$$\begin{vmatrix}
x_1^2 & y_1^2 & z_1^2 & x_1 y_1 & x_1 z_1 & y_1 z_1 \\
x_2^2 & y_2^2 & z_2^2 & x_2 y_2 & x_2 z_2 & y_2 z_2 \\
x_3^2 & y_3^2 & z_3^2 & x_3 y_3 & x_3 z_3 & y_3 z_3 \\
x_4^2 & y_4^2 & z_4^2 & x_4 y_4 & x_4 z_4 & y_4 z_4 \\
x_5^2 & y_5^2 & z_5^2 & x_5 y_5 & x_5 z_5 & y_5 z_5 \\
x^2 & y^2 & z^2 & xy & xz & yz
\end{vmatrix} = 0 \tag{6.4}$$

must hold. Equation 6.4 in turn gives a formula for the coefficients $a,\ldots,f$ in terms of the (x_i,y_i,z_i): If we expand the determinant along the last row, we see immediately that $a,\ldots,f$ are given by the 5×5-subdeterminants of the 5×6-matrix.

This formula is computationally expensive, so we would like mention a better way to find the conic using the "Plücker μ." The linear combination $C=\lambda A+\mu B$ of two conics A and B that meet in four different points is another conic that also meets these four points. We can adjust the two parameters λ and μ such that a given fifth point is also met by C. So in order to find the conic that meets five points we just have to find two conics that meet in four of the points. Since the product of the equations of two lines is the equation for the degenerate conic consisting of these lines, we can just proceed like this:

- Choose two pairs of points, say (p_1,p_2) and (p_3,p_4).
- Calculate the degenerate conics $(p_1\times p_2)\cdot(p_3\times p_4)^T$ and $(p_1\times p_3)\cdot(p_2\times p_4)^T$. These meet exactly in $p_1,\ldots,p_4$ (if the points are in general position).
- Adjust the linear combination $C=\lambda A+\mu B$ such that C meets p_5, too. You can choose $\lambda=p_5^tBp_5$ and $\mu=-p_5^tAp_5$.

Euclidean Circle by three points

The general equation defining a circle with center (x_0,y_0),

$$(x-x_0)^2+(y-y_0)^2-r^2=0,$$

can be written with homogeneous coordinates as

$$x^2+y^2-x_0xz-y_0yz+(x_0^2+y_0^2+r^2)z^2=0.$$

This shows immediately that for the complex homogeneous coordinates $I:=(i,1,0)$ and $J:=(i,-1,0)$ the equation is satisfied. This shows that all circles share two common complex points I and J. See the last chapter for historical comments on the introduction of I and J into projective geometry.

Since a circle is a special conic and a circle is defined by three points, we can immediately define a basic operation for circles using the Conic primitive and the two constant points I and J. So for all $p_1,p_2,p_3\subset P^3$ we have

$$\mathsf{Circle}(p_1,p_2,p_3)=\mathsf{Conic}(I,J,p_1,p_2,p_3).$$

Circle by center and point

Another way to describe a circle is by its center and radius. Of course, this depends on the measurements, but here we give a formula that works for any measurement, using the fundamental conic F of the corresponding Cayley-Klein geometry.

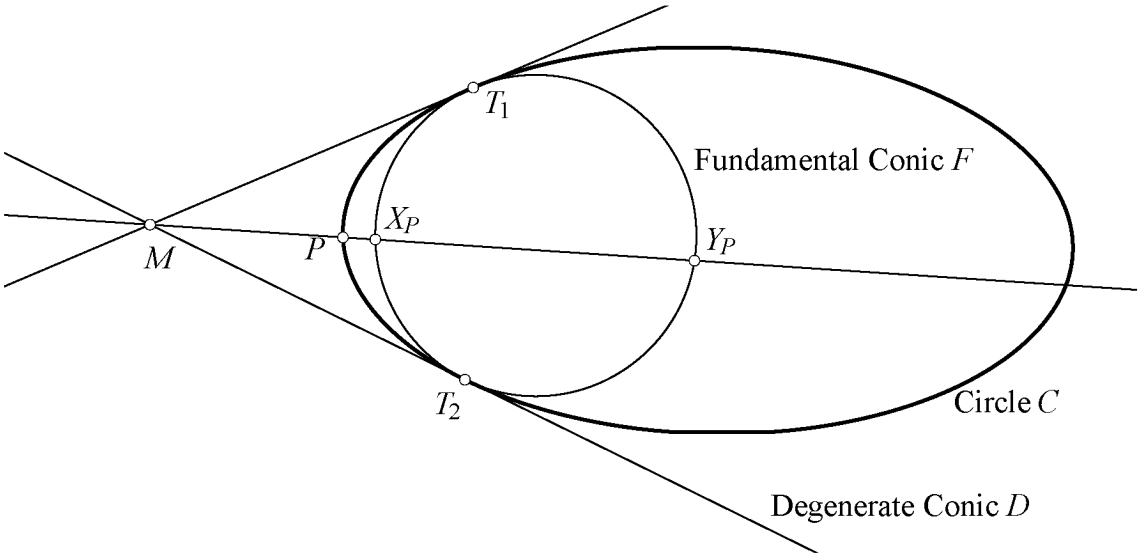


Figure 6.1: A circle as linear combination of a degenerate circle with radius 0 and the fundamental conic.

Let D be the degenerate conic that is given by the two tangent lines to F through M. This conic is a circle with radius 0 – for any point Q on D and intersections X and Y of the line through M and Q with the fundamental conic the cross ratio $CR(QM|XY)$ is 1, since $X = Y$.

We claim that $C := \lambda F + \mu D$ is a circle around M, i.e. all points on C have the same distance to M. Let P be an arbitrary point on C, and let X_P and Y_P be the two points of intersection of the line through P and M with F. The distance of P to M is given by

$$\text{dist}(MP) = c_d \ln(CR(MP|X_P Y_P)) \tag{6.5}$$

The unique projective transformation that maps the two tangent points T_1 and T_2 of D and F as well as M to themselves and that maps P to a point Q is denoted by A_Q.

Define F' as the set of points given by applying A_Q to X_P for all Q on C:

$$F' := \{x \text{ s.t. } x = A_Q X_p \text{ for some } Q \in C\} \tag{6.6}$$

An easy calculation shows that F' is given by a quadratic form, it is a conic. Moreover, it is a conic that is tangent to the two lines of D and that meets X_p, so it must be the fundamental conic F.

This shows that Cayley-Klein-measurements are invariant under A_Q for $Q \in C$, which means that all points on C have the same distance from M as P.

We have seen that it is sufficient to find the right linear combination of F and D to find the circle around M through P using the same adjustment step as above in Sec. 6.1.2. The only part that is still missing is a way to find the degenerate tangent conic, but we will see how to do that in the next section.

Intersection of a conic and a line

Intersecting a conic and a line is the polar operation to constructing the tangent lines to a conic. It is sufficient to describe one of these operations, and we will use the second one because it is a little bit easier to visualize.

We are looking for the two lines ℓ_1 and ℓ_2 through a point p that are tangent to a given conic C. We will use a two-step procedure for the calculation. First we calculate a degenerate conic D that consists of the two tangent lines, and then we try to "factorize" the conic D into the two lines.

A useful tool is the cross operator that creates a matrix out of a vector that simulates the cross product:

Definition 6.2 *Let $p = (x,y,z)^T \in \mathbb{K}^3$ be a vector. The cross operator L^p of p is a matrix in $\mathbb{K}^{3\times 3}$ which is defined by*

$$L^p = \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} \tag{6.7}$$

A cross operator always yields an anti-symmetric matrix, in fact, any anti-symmetric 3×3-matrix can be written as a cross operator of a suitable vector.

Lemma 6.3 *Let $p = (x,y,z)^T$ be a vector. Then*

$$L^p \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \end{pmatrix} \tag{6.8}$$

Proof 6.4 *Eq. 6.8 follows immediately by using Eq. 6.7 from the definition of the cross operator.* □

Now the matrix of D, the degenerate conic of two lines through p that are tangent to C is given by

$$D = (L^p)^T C (L^p)\,. \tag{6.9}$$

This follows again immediately from the definition of the cross operator. A point x lies on D (or, in the polar sense, a line x is tangent to D) if

$$x^T D x = 0\,. \tag{6.10}$$

Plugging Eq. 6.9 into Eq. 6.10 gives

$$0 = x^T D x = x^T (L^p)^T C (L^p) x = (L^p x)^T C (L^p) x = (p \times x)^T C (p \times x) = 0\,, \tag{6.11}$$

so a point x lies on D if the line through x and p is tangent to C.

The next step is to find two vectors ℓ_1 and ℓ_2 such that the matrix $\ell_1 \ell_2^T$ and D describe the same conic. As a first observation we see that a conic is invariant under addition of a cross operator to its matrix.

Lemma 6.5 *For all $q \in \mathbb{K}^3$ and matrices $A \in \mathbb{K}^{3\times 3}$ the conic described by A and the conic described by A plus the cross operator of q are the same:*

$$\{p \text{ s.t. } p^T(A+L^q)p = 0\} = \{p \text{ s.t. } p^T Ap = 0\} \tag{6.12}$$

Proof 6.6

$$\begin{aligned} p^T(A+L^q)p &= p^T Ap + p^T L^q p \\ &= p^T Ap + (p \times q)p \\ &= p^T Ap + \det(p,q,p) \\ &= p^T Ap \end{aligned}$$

We will now use Lemma 6.5 to find a matrix D' that describes the same conic as D, but is of the form $\ell_1 \ell_2^T$. We know that D' has rank 1 – all columns (or rows) are just multiples of each other. This condition can be expressed in terms of the subdeterminants of D': Every 2×2 subdeterminant must vanish.

Let $D = \begin{pmatrix} a & d & e \\ d & b & f \\ e & f & c \end{pmatrix}$. Choosing the right subdeterminants gives nice formulas for the coefficients (l,m,n) for the cross operator:

$$0 = \begin{vmatrix} b & f-n \\ f+n & c \end{vmatrix} = bc - f^2 + n^2 \Leftrightarrow l^2 = -\begin{vmatrix} b & f \\ f & c \end{vmatrix} \tag{6.13}$$

and similarly

$$m^2 = -\begin{vmatrix} a & e \\ e & c \end{vmatrix} \text{ and } n^2 = -\begin{vmatrix} a & d \\ d & b \end{vmatrix}. \tag{6.14}$$

We have to take care that we choose the right signs for the square roots when we calculate (l,m,n), but we can do it such that no other subdeterminant equation is violated. Now D' is obtained by adding the cross operator of (l,m,n) to D.

Further investigation reveals that (l,m,n) is a scalar multiple of p:

$$\begin{pmatrix} l \\ m \\ n \end{pmatrix} = \pm\sqrt{\alpha}p$$

where

$$\begin{aligned} \alpha &= z^2ba + 2zbxe - bx^2c + 2yzfa - 2yfxe - y^2ca + z^2d^2 \\ &\quad -2zdye - 2zdxf + 2dxyc + y^2e^2 + x^2f^2 \end{aligned} \tag{6.15}$$

This fixes the necessary sign decisions, we just have to use the same signs as in p.

Here is another way to get α. Using Eq. 6.9 and our knowledge about (l,m,n) we can write

$$(L^p)^T C L^p \pm \sqrt{\alpha} L^p = ((L^p)^T C \pm \sqrt{\alpha}\mathbf{1}) L^p \tag{6.16}$$

In order to get a rank-1 matrix from Eq. 6.16 $\sqrt{\alpha}$ must be a eigenvalue of $(L^p)^T C$, because else the determinant of $(L^p)^T C \pm \sqrt{\alpha}\mathbf{1}$ does not vanish and the product of this matrix with L^p will have the same rank as L^p, which is 2.

As a last step we have to choose a row ℓ_1 and a column ℓ_2 of D' that are not equal to zero (we get a zero row or zero column if one entry of ℓ_2 resp. ℓ_1 is zero, but there is always a good choice, since otherwise the rank of the matrix would be 0, not 1). Since $\ell_1 \ell_2^T$ is a scalar multiple of D', these are the tangents to C through p.

Intersection conic/conic

As a final example for the Plücker-μ technique we show how to find the four intersections of two conics.

The important observation is that all linear combinations of two conics will pass through the common intersections of these two conics. By adjusting the parameters of the linear combination we can find degenerate conics consisting of two lines that pass through the intersection points. Splitting these conics into two lines reduces the problem either to the previously solved conic/line-intersection or, if we split two of them, to simple line/line-intersections.

Fig. 6.2 shows the situation for two conics that intersect in four real valued points. The six lines are a generalization of the radical axis of two circles, see also Sec. 2.2.2.

The adjustment of the parameters is computationally easy. Let $D(\lambda) = C_1 + \lambda C_2$ be a linear combination of the two matrices defining the two conics. The necessary condition for $D(\lambda)$ to become degenerate is

$$\det(D(\lambda)) = 0 \tag{6.17}$$

which is a polynomial equation of degree 3. So there are three – possibly complex – solutions for this equation, which correspond to the three different degenerate conics through four points.

In the case where the coordinates of the conics (the entries of the defining matrices) are real numbers, there will be at least one real solution to equation 6.17. Fig. 6.3 shows this solution for two ellipses that intersect in two points only.

Compound operations

The basic operations so far are fairly complete. Most other important constructions can be built from these basic building blocks. We will try to follow this principle of a small and easy to handle set of operations. This is not only useful to unclutter the theory, but

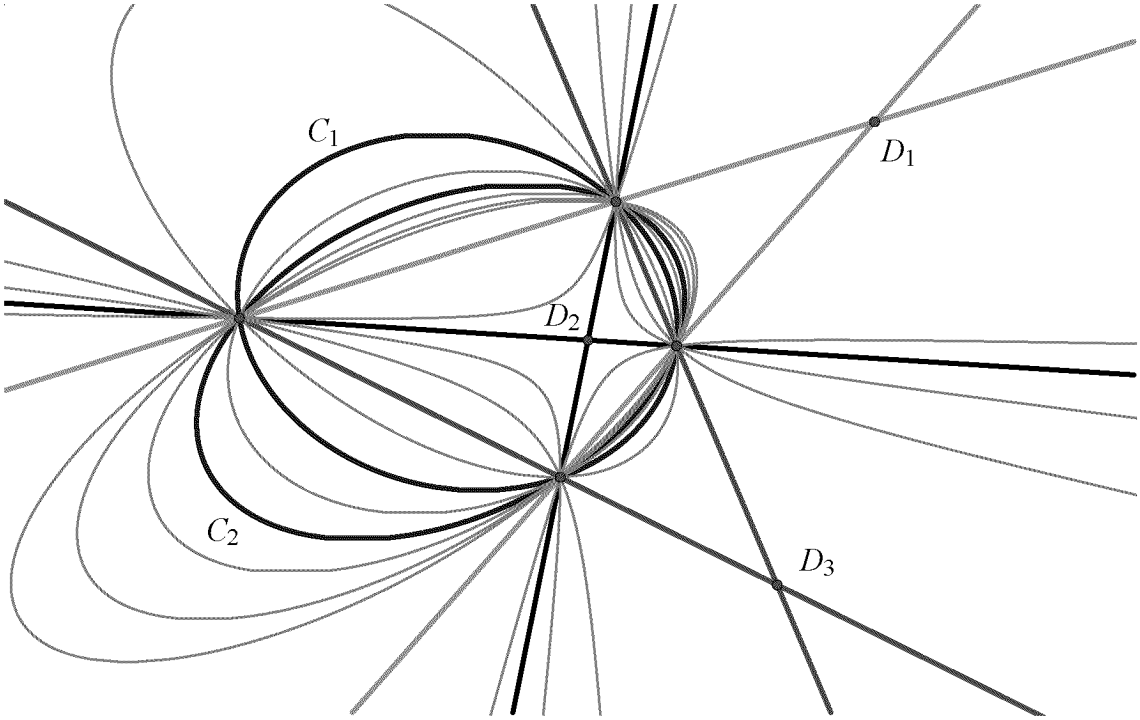


Figure 6.2: All linear combinations of the two conics C_1 and C_2 pass through their intersection points. In this family there are three degenerate conics D_1, D_2 and D_3 that consist of two lines each.

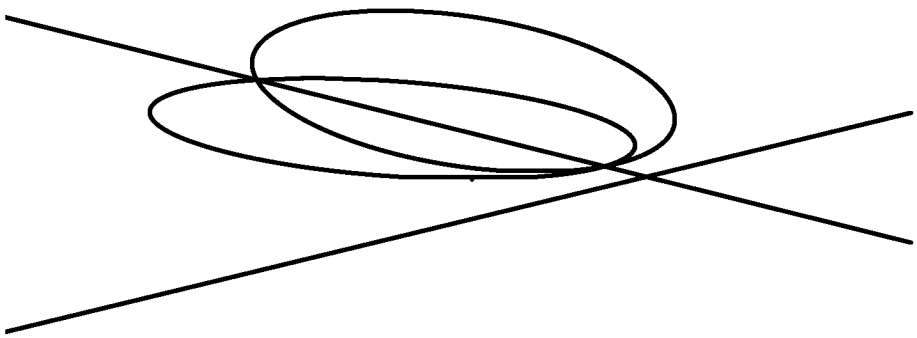


Figure 6.3: Two conics and their real generalized radical axes.

also from an implementational point of view: Using macros that re-use easily verifiable operations greatly decreases the number of implementation bugs. Sometimes this induces a small performance penalty, but we prefer the overall stability and reliability of such a system. See the implementation section for more details on this issue.

However, one construction which we will use as an example later should be mentioned. We can construct the (Euclidean) angular bisectors of two lines using the following construction: Construct a circle through the intersection of the two lines, and choose for each line one of the intersection points of it with the circle. Construct two circles using these two points, taking one of it as center and the other on the boundary and vice-versa. The two intersections of the last two circles lie on one of the angular bisectors, as does the intersection of the first two lines. This completes the construction.

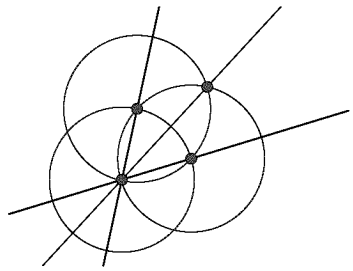


Figure 6.4: Angular bisector

Note that we did not specify how to get a certain angular bisector. It will be one of the key statements of this chapter that there is no generic construction for only one of the two bisectors, you will always get both or none.

Another side remark: For two identical lines this construction does not work, since we cannot find the center of the first circle. This can easily be fixed by changing the construction from "angular bisector of two lines" to "angular bisector of two lines that meet in a point," where the point is specified separately. See Sec. 9.3.1 for more information on how this is implemented in *Cinderella*.

6.1.3 GSP formulations

Similarly to Sec. 5.1.2 we will now define a homogeneous relational instruction set for the basic operations on points, lines and conics extending Def. 5.3. We will omit an analogon to the abstract RIS as given in 5.2 – the formalism would be too much just for having a tool for informal descriptions. Instead we will just refer to the basic operations of the last section and assume some reasonable RIS.

Definition 6.7 (Homogeneous RIS for points, lines and conics) *The* homogeneous RIS for points, lines and conics *is the homogeneous RIS for points and lines as in Def. 5.3 augmented by the following objects and operations:*

The new object type O_3 is given by the set of all matrices $M \in \mathbb{K}^{3\times 3}$. We do not exclude the zero matrix, and we do not identify scalar multiples.

The new primitives correspond to the necessary calculations for the basic operations as in Sec. 6.1.2.

$$
\begin{aligned}
\mathsf{DegConic}^2 &= \{(x,y,M)\ s.t.\ M = xy^t\} && \subset (O_1)^2 \times O_3 \\
\mathsf{Eval}^2 &= \{(x,M,s)\ s.t.\ s = x^tMx\} && \subset O_1 \times O_3 \times O_2 \\
\mathsf{LinComb}^4 &= \{(A,B,\lambda,\mu,C)\ s.t.\ C = \lambda A + \mu B\} && \subset (O_3)^2 \times (O_2)^2 \times O_3 \\
\mathsf{CrossOp}^1 &= \{(p,M)\ s.t.\ M = L^p\} && \subset O_1 \times O_3 \\
\mathsf{MatrixEval}^2 &= \{(A,B,C)\ s.t.\ C = B^tAB\} && \subset (O_3)^3 \\
\mathsf{Split}^1 &= \{(D,p)\ s.t.\ x^tDx = 0 \Leftrightarrow x^t(pq^t)x = 0, q \in \mathbb{K}^3 \\
&\qquad \textit{for all } x \in \mathbb{K}^3\} && \subset (O_1)^2 \times O_3 \\
\mathsf{Radical}^2 &= \{(C_1,C_2,D)\ s.t.\ D = \mu C_1 + \lambda C_2, \\
&\qquad \det(D) = 0\} \subset (O_3)^3
\end{aligned}
$$

This RIS is not determined, since the Split and Radical primitives are not determined. In the next section we will see that this is not caused by an awkward definition but that it is a mathematical necessity.

We also note that Split and Radical are not given explicitly but implicitly. We could have forked the necessary decisions into two respective three different solutions, thus avoiding the scalar multiplicity of the results. Anyway, it does not make a real difference (theoretically, not in the implementation), whether we have infinitely many solutions or just two or three.

6.2 Determinism vs. Continuity

The two major properties of point/line-constructions was that they are determined (or conservative) and continuous. In this section we will see that we have to give up at least one of these when we admit circles and their intersections in the constructions. The same is true if we introduce angular bisectors, actually we use the circle intersections to construct angular bisectors.

We still do not fix what we mean by "continuity," because we are not yet sure what we can expect. Instead, we rely on an intuitive notion of continuity, which will be adjusted whenever necessary.

6.2.1 Iterated angular bisectors

Consider the following construction: Two lines a and b that go through a point A and one angular bisector c of a and b through A. Furthermore assume that b is fixed and a can be rotated around A. It is obvious that the c must rotate by $\frac{\alpha}{2}$ whenever a rotates by α if we want a continuous movement. After a full turn (360 degrees or 2π) of a the angular bisector will be at its old place. So we have both a continuous and deterministic behavior.

But what happens if we use another angular bisector for a line d at angle $\frac{\alpha}{4}$ (Fig. 6.5)? This line will move with a quarter of the rotational speed of a, so after a full turn of a we end up with d rotated by 90 degrees. *For the same input (a and b) this construction has different output if we assume continuity in movements.*

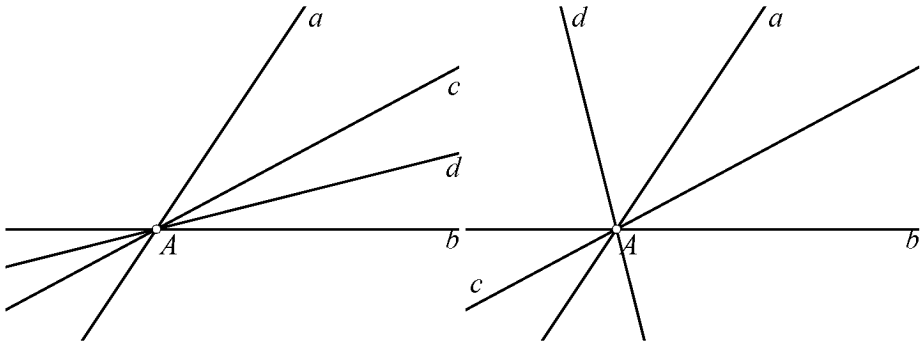


Figure 6.5: The two possible configurations of a angular quadrisector construction. Under the assumption of continuous movements both must be connected by a continuous motion of a, hence this construction is not determined.

Theorem 6.8 (Continuity destroys Determinism) *As soon as you two angular bisectors in a construction, you cannot have deterministic and continuous behavior. If a RIS for points, lines and conics admits an angular bisector construction, then under the assumption of continuity the RIS cannot be determined.*

Proof 6.9 *Use an angular quadrisector construction for a line ℓ that moves at the quarter of the rotational speed of another line a. This line ℓ will be perpendicular to its original position after a full turn of a.* □

It is important that it is not the fault of the homogeneous RIS as defined in 6.7 that it is not determined (because of the indetermined primitives Split and Radical). Any reasonable RIS, that is, any RIS that admits at least the basic constructions for points, lines and conics, will not be determined if it shows continuous behavior, if it uses the homogeneous coordinatization of objects.

6.2.2 Finite Augmentation does not help

Why did we need an angular quadrisector in the last section? Shouldn't one angular bisector be enough? In fact, yes, because we usually do not consider the direction of a line, so we only had to rotate a by a half-turn, and then c would have been perpendicular to its original position.

But most software uses orientations, and it looks like a promising way to distinguish lines, points and conics by their signs. At least for one angular bisector it seems to work, so this technique could be the solution to find a better RIS by modifying the objects in addition to the primitives.

Unfortunately, sign decisions are not the solution to the continuity problem. Here is a simple construction that fails to work: Take three circles of the same radius, one fixed and two having their center on the first one (we can create points on circles using lines

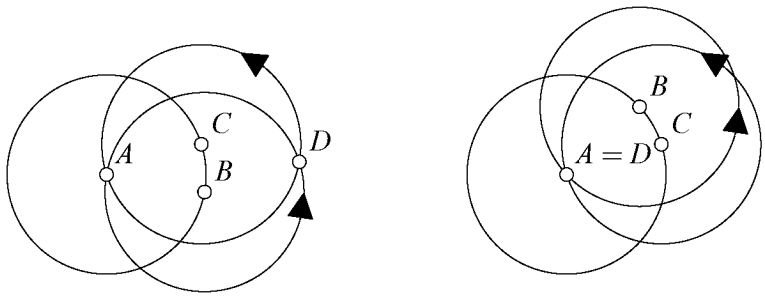


Figure 6.6: Non-continuous behavior due to sign decisions. When B moves through C, then D jumps onto A.

through the center and another free point) like in Fig. 6.6. At the intersection D a counter-clockwise turn of the circle that is centered at B will move it onto the circle around C. If we use this handedness decision all the time then D will jump onto A when B moves through C.

The reason why sign decisions make one angular bisector work, but not the next, quadrisecting, one, is revealed by closer inspection. It is true that after a 180 degree turn of a (see again Fig. 6.5) the bisector c is at its old position. But it has the wrong orientation – we are not able to forward our decision making tool through the construction. You can also see this effect in the figure: The label of c is at the opposite side in the right drawing, which is a hint for the orientation of c.

This loss of information at an angular bisector can be exploited to proof the following theorem. Iterating angular bisectors makes all "additional information approaches" fail. Trying to achieve determinism under the assumption of continuity by adding finite number of bits of information to the input elements of a construction will fail.

Theorem 6.10 (Finite Augmentation Theorem) *If we allow only a finite number of additional bits of information for the input elements of the homogeneous RIS it still cannot be determined if it is assumed to be continuous.*

Proof 6.11 *Assume that we add n bits of information to the input elements. Consider an iterated angular bisector construction of $n+1$ iterated bisectors. We must make 2^n full turns with the rotating line in order to return to the starting configuration, which means that there are 2^{n+1} different instances for the same construction sequence (see Fig. 6.7 for the 2^3 instances in the case $n = 2$). These cannot be encoded by n additional bits.* □

The contrasting difference to Thm. 6.8 of the finite augmentation Theorem is that it extends the impossibility of continuity and determinism to relational instruction sets using extended descriptions of the objects. Thm. 6.8 only addresses relational instruction sets that work on the same object types as the homogeneous RIS for points, lines and conics.

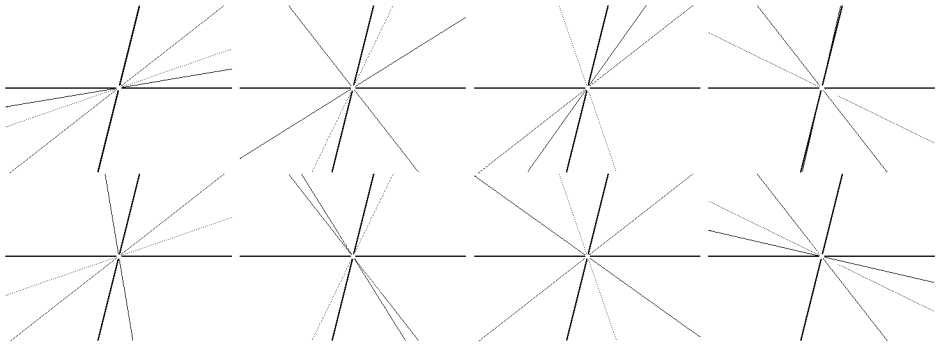


Figure 6.7: Illustration of the finite augmentation theorem. The eight (2^3) instances of a three-fold iteration of the angular bisector construction.

It is not at all clear how an augmented RIS could be used in a dynamic geometry system, because we do not know how to assign the additional bits, but Thm. 6.10 tells us that we do not need to care.

6.2.3 Reverse Augmentation

A closer inspection of the last theorem points out a possible criticism: We only used the additional bits of the input elements, and we did not try to use the additional bits of the dependent objects. If we could make use of these, then we were able to make the continuous iterated angular bisector construction determined. Exactly the number of bits that we need to encode the exponential number of different states (or instances) of the construction is available if we only allowed oriented lines.

But, as we have also seen, we are not able to propagate the necessary bits through the construction; already the orientation of the first angular bisector cannot be read off the input lines. Still there might be a way to use these additional bits "inside" the construction for a good determinism strategy, using some kind of reverse augmentation. Throughout this thesis we will assume that we only use the input elements for detecting determinism, and in fact it seems like there is no easy way to use the lost bits of dependent elements.

6.2.4 Is Continuity important?

We might ask whether it is really important in real life that a Dynamic Geometry system is continuous. It is not an artificial restriction that is introduced because it is mathematically interesting, but it is crucial for elementary tasks in a dynamic geometry system.

A popular example, which is also mentioned in [53], of the importance of continuity is shown in Fig. 6.8. A segment is mirrored at a line using a small construction to mirror each endpoint. If a Dynamic Geometry software is not able to keep track of the intersections of a circle and a line, it can happen that the construct mirror image flips over to the original image, or even worse, only one point switches to the other side.

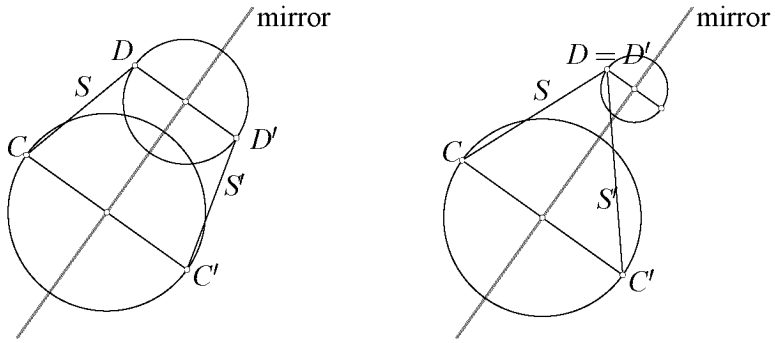


Figure 6.8: On the left you can see the original construction of a mirrored segment S, on the right you see the same construction after the point D was moved and new (wrong) decisision were made by the software.

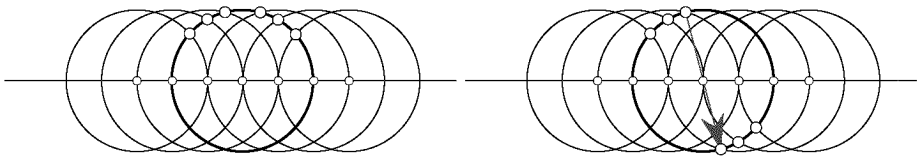


Figure 6.9: A construction that jumps in all geometry software packages except *Cinderella*, and also in CAD software like Autodesk Mechanical Desktop: A circle (several positions drawn with thin lines) moves through another circle (one position drawn with thick lines) of the same radius. On the left the behavior we would like to see, the intersection of the two circles stays above the line, on the right the jumping situation. The decision strategy of Cabri cannot work here, since the other intersection is not already given by some other, fixed point.

In the case of this mirror construction there is an immediate argument why this cannot be the right behavior: A point and its mirror image are the two intersections of a certain line and a circle, and if these two intersections are distinct – the point does not lie on the mirror line – then one should be the original point and the other one should be its image. It is easy to check this condition for some constructions, and in fact this strategy is built into Cabri Géomètre [54]. See also section 9.2.4 which describes how *Cinderella* re-uses this idea for more efficient calculations. A situation where this strategy fails is given in Fig. 6.9: Two circles of the same radius are centered on a line

Another "real world" example of problems with jumping elements is the angular bisector construction as introduced in section 6.1.2. We already mentioned that it is not clear how to describe this angular bisector construction in a way that gives the "right" angular bisector also after movements, and we saw in section 6.2.2 how we can force discontinuities by iterating angular bisector constructions in a conservative system.

Another effect we can see in many geometry softwares is that depending on the first

helper point on one of the lines we get either one or the other angular bisector, and the "jump" is when the helper point moves through the intersection of both lines.

A third reason for continuity are geometric theorems: A well-known theorem for triangles states that the three angular bisectors meet in a point – which is true if you choose the right angular bisectors, but fails badly if you take the wrong ones, see Fig, 7.4 on page 111. What we want to avoid is that by moving a construction we can come from a "theorem is true"-situation to a "theorem is false"-situation.

This would not only be counter-intuitive, but also destroys any meaningful notion of a geometric theorem. This is a big problem when we want to do randomized theorem proving as in Sec. 5.3. How can we get a random example of a theorem (or even wore, many random examples), if we cannot describe the theorem for arbitrary parameters? We have exactly one instance, the one we constructed, and we do not have a way to create any other instance. This is far away from a "random" instance.

It is not immediate that continuous movements guarantee that theorems are true forever (although we will prove that later), but it is much more likely than in a system which shows random behavior.

6.2.5 Algorithm continuity

All examples above have in common that the jumping situations occur at degenerate positions – for example, two circles become equal, and their intersection is undefined, or two points become equal and their connecting line is undefined.

So maybe it is just an isolated effect that occurs at these singular situations. Most people could accept this behavior because you can "see" that there is a "problem" somewhere.

We would like to point out that there *are* situations where we have really unmotivated discontinuities. The standard example are again angular bisectors of two lines that share a common given point, which are always defined, regardless of the two lines being equal or not. These can be used for the iterated bisector construction of Thm. 6.10, which will always create a jump in determined geometry systems. In a way this is a second order effect – we do not see the first discontinuity (the sign change of the first angular bisector), but we see the change when a construction uses this sign (as the intersection of the circle and a line does in most geometry software).

Another example are operations like "conic/conic-intersection," which hide the complexity of the calculation from the user. Here it is not possible to see from the outside where the internal singularities are. Try intersecting two conics in a Dynamic Geometry software – other than *Cinderella* – that can handle conics, and watch how the four intersections are interchanged all the time while moving one of the conics. This highly undesirable behavior is not understandable without knowing the internal implementation of the intersection algorithm.

Related to this is the behavior of *macros*. A macro is a part of a construction sequence that can be reused for a construction, and it is defined by its input and output elements. The

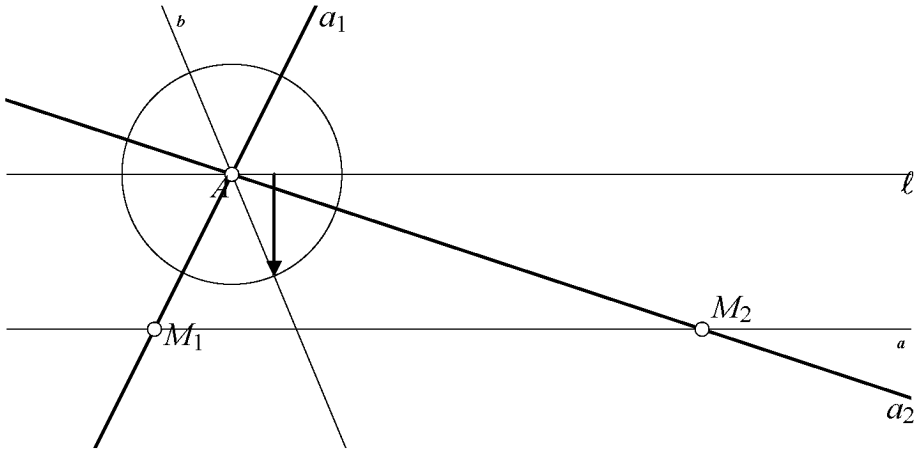


Figure 6.10: A construction having only two degenerate positions, when $A = M_1$ or $A = M_2$. The oriented distance (marked by an arrow) can be plotted in a graph as in Fig. 6.11.

corresponding intermediate construction steps and intermediate elements will be hidden if you apply a macro, making it look like a new tool. Most geometry software systems support macros in one or the other way.

Imagine that you create a macro using two lines and a point as input and that constructs the angular bisector using three circles as intermediate elements. If you apply this macro, it will create one of the angular bisectors (most times you will not be able to predict which one). You will not see the circle construction that drives this angular bisector, but of course you will experience all the discontinuities that are caused by it.

Our point is that even if you are aware of the singularity problems in constructions, you will always come into situations where you will not immediately "see" what causes the discontinuity. This means that the software will show unpredictable and unintuitive behavior.

6.2.6 Surfaces

Let us have a look at an interesting construction that shows the incompatibility of determinism and continuity once more. The construction will have only two singular positions, and in all other cases the dependent elements will have real valued coordinates.

In Fig. 6.10 we see one (movable) point A with two lines a_1 and a_2 connecting A with M_1 and M_2. There are two instances of the angular bisector b of a_1 and a_2 through A. Intersect one of it with a circle of fixed radius around A, and measure the distance of the point of intersection from a line ℓ that is parallel to the connecting line a of M_1 and M_2 and goes through A.

This distance in relation to the position of A is shown in the graphs in Fig. 6.11. The left plot corresponds to the situation in a determined system like Cabri: For every position of A there is only one distance $\mathrm{dist}(B, \ell)$. The bottom picture shows all possible instances,

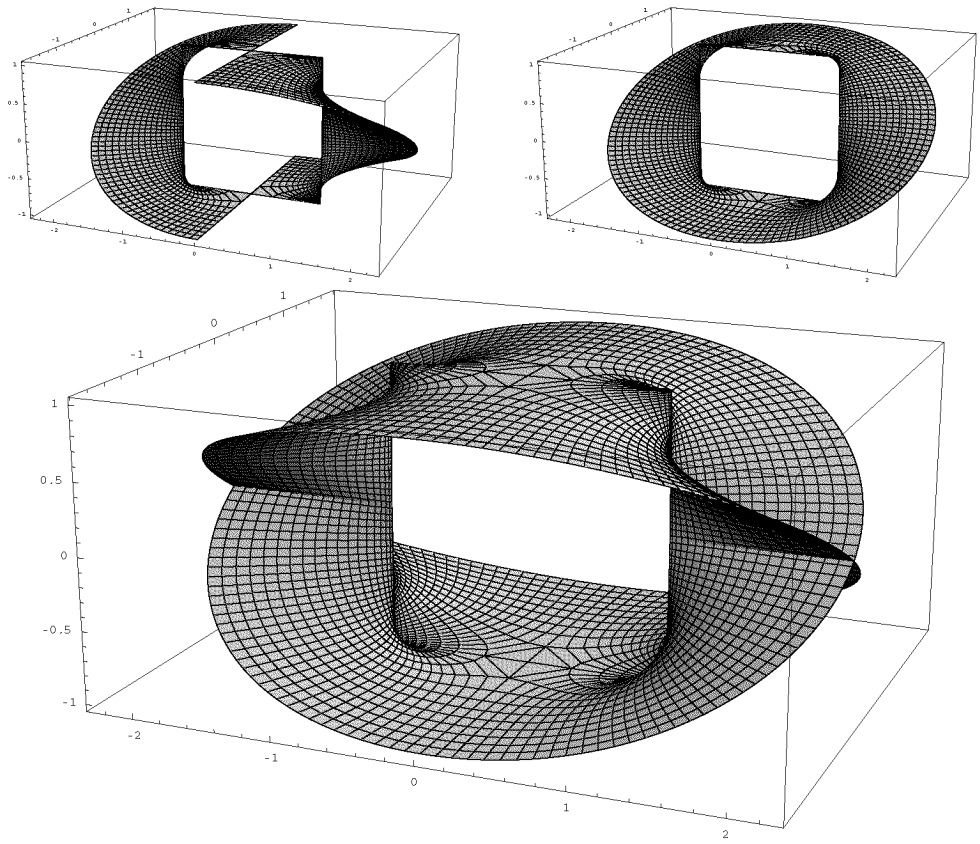


Figure 6.11: The two small graphs show incomplete surfaces generated by the construction shown in Fig. 6.10 in determined geometry software. The big graph shows the complete surface that must be available in a continuous system.

i.e. up to two instances per (x,y)-position.

Try to remove parts of the plot on the right to make it determined while still having a smooth surface. This is not possible; if you start at any position you will have to add more and more to make the surface smooth until you end up with all the points. On the left you can actually *see* where the jump in a determined geometry software occurs (the plot shows exactly the behavior of Cabri): If you move point A over the segment between M_1 and M_2, the distance will "flip," which is caused by changing to the other angular bisector instance. Other software might show other behavior, but there will always be a "jump."

This example suggests two approaches. The first one would be to try to minimize the occurance of jumping situations by (implicitly) selecting the right instances for every point. The second one would be to allow indeterminism to make continuous moves possible.

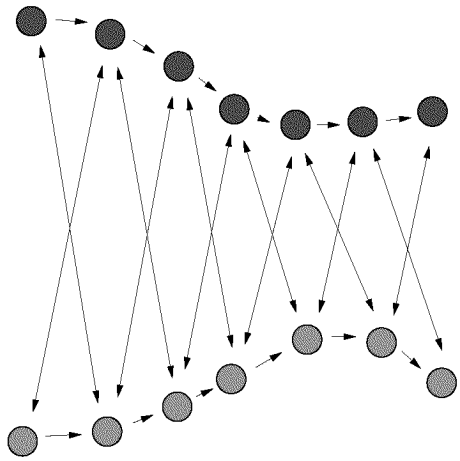


Figure 6.12: The assignment of the new to the old intersections is easy if the sum of the distances "across" is significantly greater than the sum of the other two distances.

6.3 Tracing

Now that we know that the search for the "right," determined, algorithms will not be successful, we will use an approach that looks very unsophisticated at first, but it will turn out to be the solution to the continuity problem later.

What do we try? We have an instance of a construction, we change a parameter and we have to choose the right one out of a set of new instances. First of all, we can make the global decision by deciding each undetermined construction step at a time. So we have to be able to assign the two new intersections of a conic and a line (or two circles, or the four new intersections of two conics) to the old ones. We will do this using a "near-to" decision: Under the assumption of continuity the new points should be close to the old points after a small movement.

By measuring the pairwise distances we might be able to find the right assignment (Fig. 6.12). If not, for example, because the points moved to fast or are too close to each other, we will need another, intermediate instance of the construction. We do not know how to get this at the moment, but let us assume that it is possible.

With this vague strategy in mind, let us investigate the problems of this method.

6.3.1 Complex Elements

We have seen already that complex coordinates are useful in projective geometry, even if we want to handle constructions within the reals. For example, the circle operation for Euclidean circles used the conic construction together with two complex-valued points. The whole theory of measurements in Cayley-Klein geometry could not work without complex numbers.

Another reason for working within the complex projective plane are intersections of conics and lines: As we have to solve a quadratic equation for the points of intersection we can end up with two, one or no solution, corresponding to true intersection, tangent line and disjoint objects. This would be a problem for the near-to tracing algorithm: How

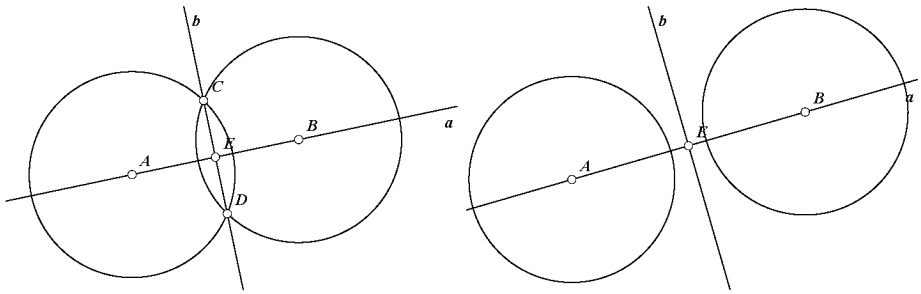


Figure 6.13: The construction on the left does work even if the distance of A and B is greater than the radii of the circles. The line b is still a real line, and E is still the midpoint of A and B.

can we find the best match if there is nothing or not enough to match? If we solve all polynomial equations over the complex numbers, we will always have the same number of solutions (counted with multiplicity).

If we can do near-to decisions also with complex elements then we can still trace even if intersections become complex. It is not easy to find a numerically stable distance measure in $\mathbb{C}P^2$ that can be used for tracing, but we do not care yet.

A nice side-effect of using the complex numbers as base field for geometric constructions is that theorems that are true for the reals or constructions that work within the reals work even if intermediate elements become complex. In the next chapter we will prove this, here we will just give two examples.

Example 6.12 (Midpoint construction) *The midpoint between two points A and B is the intersection of the line connecting A and B and the perpendicular bisector of the segment AB. The latter can be found by intersecting two circles with equal radii, one around A and the other around B, and connecting the two intersections by a line. If the two circles are too small to intersect in real points, we can use the complex intersections and get the same result, since both intersections are complex conjugates and their join is again a real line. See Fig. 6.13.*

Example 6.13 (Radical intersection Theorem) *The three radical axes of three circles meet in a point, even if the circles do not intersect. This observation is based on the same conjugates argument as in the midpoint example. See Fig. 6.14.*

A very important remark must be made at this point: As we have already seen, we cannot use signs as a decision tool. We have just found another reason, namely that there is no useful way to define orientations for elements having complex coordinates. Actually, this is what Winroth has to admit in his thesis [87]: A close inspection of his work reveals that he first claims that continuity can be achieved using sign decisions, but later restricts his – incomplete – proof to the case where no complex elements or degenerate situations are involved. The proof itself is based on the wrong assumption that there is a continuous function for the "orientation vector" of all computed elements.

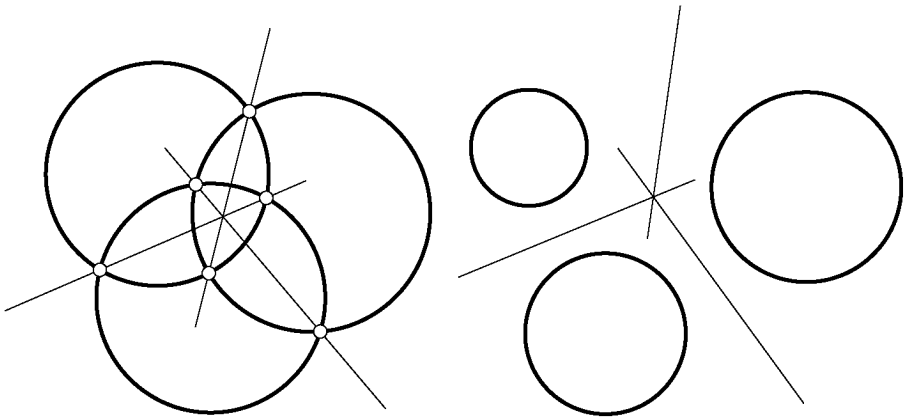


Figure 6.14: The radical axes of three circles meet in a point, even if the circles do not intersect. The construction of the radical axis using the two intersections of two circles gives always a real line.

His conclusion is to resort to "continuous tracking," which he does not explain in detail, but which is probably the same concept as the "tracing" explained above. We mention Winroth's results here explicitly because his thesis seems to provide solutions to many of the problems that are also addressed in this thesis, but actually some of his claims are at least misleading. While you read his text you believe that he proves continuity, but in the end you see that he actually proves it only for a very restricted set of special cases. Nevertheless, his work contains many good and interesting approaches to Dynamic Geometry (although many comparisions of his software to *Cinderella* are based on a very old – even at the time he made them – version of *Cinderella* and thus not valid anymore), and we hope that his system "pdb" (projective drawing board) will be available some day.

6.3.2 Singularities

So far we did not analyze one particular case where the "near-to" decision tracing strategy will fail: Whenever two points are very close or even at the same place we will not be able to distinguish them. This happens for example in the case where a line intersecting a circle is moved into a tangent situation, as illustrated in Fig. 6.15.

After we moved through a singularity, we will not be able to assign the two new paths of the points to the old ones. It is also not possible to distinguish the two points while we are in the degenerate – tangent – situation, but we do not have to distinguish these points, since in that position the two points have the same coordinates.

Winroth in his Ph.D. thesis [87] comes to the following conclusion:

> Of course, no unique correspondence can be defined between the roots before and after this singularity, not even in theory.

This conclusion is, although reasonable at first sight, completely wrong, as we will see in

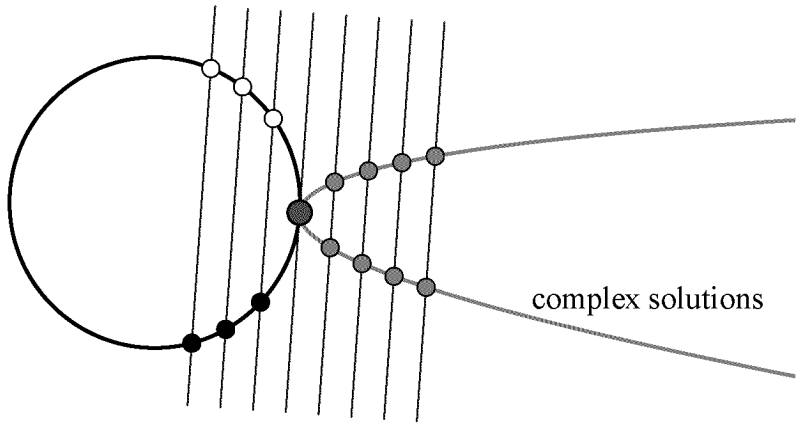


Figure 6.15: Two intersections of a line and a circle crash into a singularity: While we can assign the two new intersections to the old ones both before and after the tangent case – in real and complex space –, we do not have a way to decide how the points before and after the singularity should be assigned to each other.

the next chapter. Poncelet had a better intuition when he formulated what Kepler and others conjectured before (cited following [17], who cites from [64], boldface emphasizing was done by the author):

> Let us consider an arbitrary figure, in a general position which in a certain sense is indeterminate among all positions it can assume without violating the laws, the conditions, the bonds that exist between the different parts of the system; let us suppose according to these data one has found one or more relations or properties, which may be metric or descriptive, belonging to the figure, by way of ordinary explicit reasoning, that is, the procedure which is in certain cases considered as the only rigorous one. Is it not obvious that if while preserving those data one undertakes to vary the original figure ever so slightly and subjects parts of it to an arbitrary but continuous motion – is it not obvious that the **properties and relations, found in the first system, remain valid in its successive stages,** provided that due account is attributed to the particular modifications that may arise, for example if certain magnitudes vanish or change their direction or sign, and so on, modifications that can easily be recognized *a priori* and by sure rules?

What Poncelet writes here in a very poetic and emphatic style is that we really expect that a metric or incidence property of a construction that is true for small coordinate changes of a construction should also be true in general. When we move through a singularity, we must make sure that all changes that happen to the construction later are done in a way that none of the properties that were valid before will be violated.

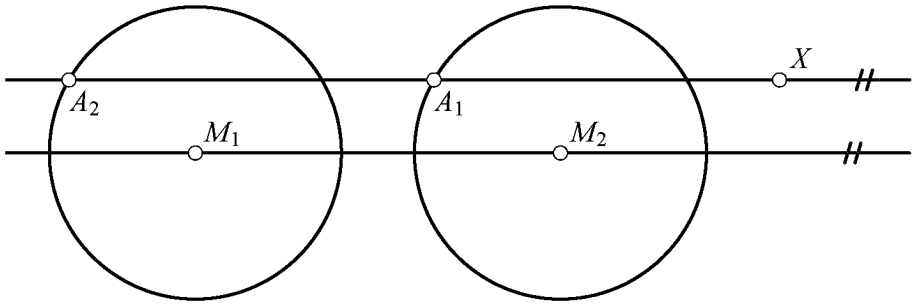


Figure 6.16: The left and the right intersection A_1 and A_2 must always be on the same side, either both on the left or both on the right half of its circle, else the principle of continuity is violated.

Let us look at the implications of the principle of continuity.

Local and global decision consistency

Consider the following construction: Two circles with the same radius that are intersected by a line parallel to the line connecting the centers of the circles. Pick a point of intersection of the line with each circle such that they are "on the same side" (Fig. 6.16).

Now, when we move the parallel line first in a way such that both circles still have two real intersections with it. We will expect that A_1 and A_2 "stay on their half of the circles." The distance between A_1 and A_2 will be constant, and equal to the distance of M_1 and M_2.

Now, if we move the parallel line through X across the tangent case the two intersections will have complex coordinates. We have to decide for both A_1 and A_2 which complex branch they take (see Fig. 6.15). There is no obvious reason to prefer one or the other branch, but in any case we have to make two consistent decisions, i.e. the distance between A_1 and A_2 should still be the distance between M_1 and M_2.

When we come back and move the parallel line into a situation where both circles have a real intersection with it, then we again have to decide for each intersection whether it should go left or right, but again we must ensure that both go to the same side, or else the principle of continuity will be violated.

We see that we have to make local decisions that are globally consistent, and this happens always when two parts of the construction go through singularities at the same time. We will see later how we can achieve global consistence while still operating locally.

Removable singularities

Another important issue are *removable singularities*: It can happen that a dependent element cannot be computed at a certain position (this means, its coordinates will all be zero), but the continuous motion before and after that situation could be completed, the singularity can be removed.

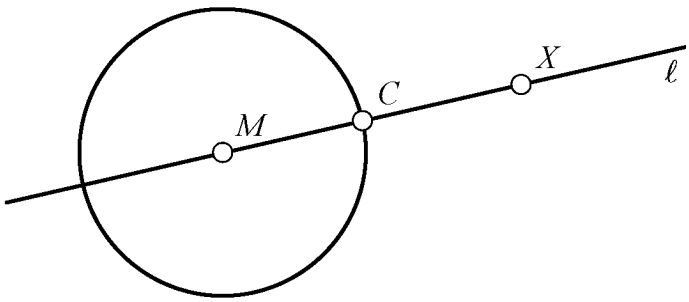


Figure 6.17: A removable singularity: If the free point X moves through the center M of the circle, the line will not be defined in that moment, and thus the intersection C with the circle will not be defined, either.

One example was presented in Fig. 6.9: If two circles with the same radius are fixed on a line and one is moved through the other, then there is a highly degenerate situation when both circles are centered around the same point. But if we consider the complete motion of one circle through the other, we can remove the singularity by glueing in the limit point, the one that is on the circle, on the correct side of the supporting line, and at the biggest distance possible from this lines.

In any case where we move through such a degenerate situation we really want the dynamic geometry software to behave as if it had automatically removed the singularity.

Here is another example which only needs a circle, the center of the circle, a line through the center and another, free point, and one intersection of the circle with the line (Fig. 6.17).

If we move X, then the two points of intersection of the circle and ℓ will always be easily distinguishable. But if we move X exactly through M, then we will have a degenerate situation where $A = X$ and ℓ is not defined. We know to which point of intersection we want C assign to after we passed the degenerate position, but how can we describe this mathematically?

6.4 Is Continuity Achievable?

Now that we know a lot of example situations that are crucial for continuity, we can ask whether this goal is achievable at all. By looking at *constructible functions* we will give some criteria that coordinate functions of geometric elements must satisfy in a system that behaves continuously.

6.4.1 Constructible Functions

The points that are constructible with ruler and compass are a well studied topic of algebra. It is a crucial part of the basis of Galois theory to show that, if we identify the

Euclidean plane with the complex numbers, the constructible numbers are a certain subfield of $\mathbb{C}$. Using the two tools starting with a finite set of starting points it is only possible to construct the smallest field containing these points and repeated field extensions of degree two of this.

For example, if we start with the two points 0 and 1 it is possible to construct $\mathbb{Q}+i\mathbb{Q}$ and all (iterated) square roots. But all constructions that are used to show this theorem neglect the decision that must be made to find the right intersection of the line and the circle created by ruler and compass (see Huckenbeck's thesis [31] where he studies the ruler/compass-constructible functions in depth, but without considering dynamics). Since it suffices to show that there is a way it is not important to give the exact choices – there is a right choice, that's all.

The situation in Dynamic Geometry is similar: The user made his choices, and later the computer has to make the corresponding consistent choices himself. Here it is not sufficient to know that there is a right choice, we have to find it.

Let us have a look at the functions that can be constructed, assuming that the Dynamic Geometry system used shows continuous behavior. Here is a way to extract such information from a construction: Similar to the von-Staudt constructions in the point/line model we use a point x on the x-axis as a *variable* x. The function's value $y = f(x)$ is another point y on the x-axis. Since we can use a parallel transport construction or perpendicular projections to "move" a function value to that x-axis this is not as restrictive as it might look like.

Definition 6.14 (Constructible function) *A function $f(x)$, $x \in [a,b] \subset \mathbb{R}$ is a* constructible function *if there there is a construction where for all positions $(x,0,1)$, $x \in [a,b]$, of a free point a dependent point F has coordinates $\lambda(f(x),0,1)$.*

Be careful: We do not use the RIS/GSP-concept for the definition of a constructible function, since we have no concept yet to avoid indeterminism.

A first trivial observation is that all constructible functions must be continuous in a continuous system:

Theorem 6.15 (Constructible functions are continuous) *Under continuity assumptions, all constructible functions are continuous.*

Proof 6.16 *If $f(x)$ is a constructible function which is not continuous at x_0, then the construction will jump if moved across x_0.* □

A large class of constructible functions are the polynomials, which in fact are continuous.

Theorem 6.17 (Polynomial constructions are constructible) *All polynomial functions $p(x)$ in one variable are constructible.*

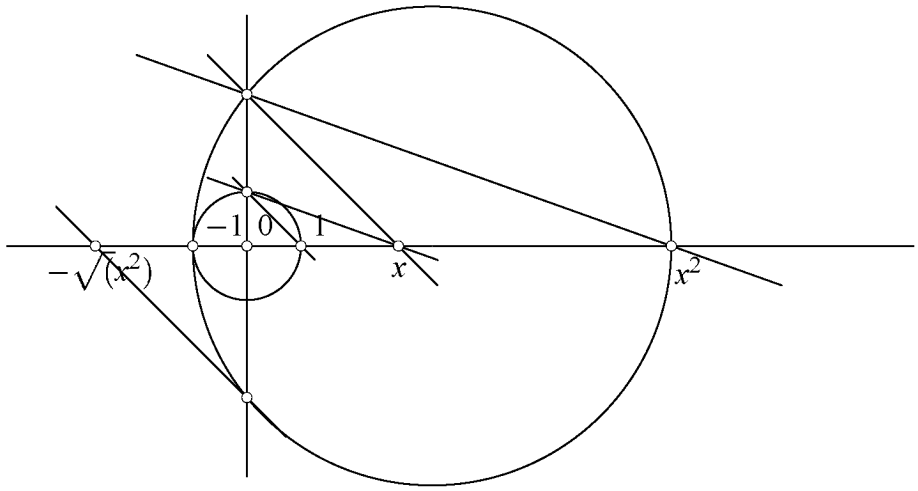


Figure 6.18: Constructing the square root of x^2.

Proof 6.18 *This is a consequence of Transf. 5.8, which shows that we can encode any straight-line program in the coordinates of a point using point/line-constructions only.* □

Are there continuous functions that are not constructible? In fact, there are. Here is an example:

Theorem 6.19 (The absolute value function is not constructible) *The absolute value function $|x|$ is not constructible in a neighborhood of 0 in a continuous Dynamic Geometry system.*

Proof 6.20 *Suppose $|x|$ is constructible. Using von-Staudt division (see Fig. 5.5) we construct a function $\frac{x}{|x|}$. This function is not defined for $x = 0$, but for every $x \neq 0$ in an neighborhood of 0, and it is -1 for all $x < 0$ and $+1$ for all $x > 0$. The point defining $\frac{x}{|x|}$ will jump at 0.* □

A surprising consequence is that there is no $+\sqrt{\cdot}$-construction in a continuous system (a function that constructs just the positive square root of a positive real number). This shows that there cannot be an explicit construction for the square root in the ruler/compass constructions without ambiguity!

Theorem 6.21 *The positive square root function $+\sqrt{x}$ for $x \geq 0$ is not constructible.*

Proof 6.22 *This is a direct consequence of Thm. 6.19 and Thm. 6.17, since else we could construct $+\sqrt{x^2} = |x|$.* □

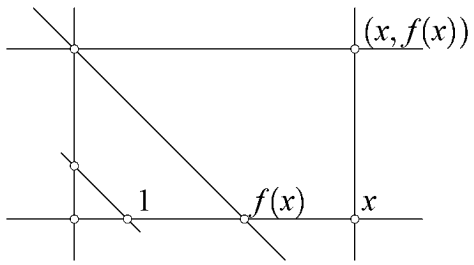


Figure 6.19: A construction that constructs a point with coordinates $(x, f(x))$ given x and $f(x)$.

We encourage you to do the construction shown in Fig. 6.18 that constructs x^2 and finds the square root of that value. With *Cinderella* the construction will not jump, and this means that the constructed point $-\sqrt{x^2}$ defines the function $-x$, and not $-|x|$.

What made the absolute value function unconstructible? It is continuous, but not differentiable at $x = 0$. A very important observation is that *any* function that is not continuously differentiable cannot be constructible in a continuous geometry system.

Theorem 6.23 (Constructible functions are differentiable) *In a continuous geometry system all constructible functions are continuously differentiable.*

The proof of this theorem is based on the following two lemmas.

Lemma 6.24 *If $f(x)$, $x \in [a,b]$ is constructible, then we can construct a point moving along $(x, f(x), 1)$.*

Proof 6.25 *Use the construction shown in Fig. 6.19.* □

Lemma 6.26 *Given a point moving along $(x, f(x))$, $x \in [a,b]$ and a $x_0 \in [a,b]$ we can construct a line that will not move continuously if $f(x)$ is not continuously differentiable at x_0.*

Proof 6.27 *We will give a construction for a line ℓ that will jump if and only if $f(x)$ is not continuously differentiable at x_0. This line is defined by the (fixed) point $A = (x_0, f(x_0))$ (we can construct this) and another point $B = (x_0 + h, f(x_0 + h))$, which uses the same construction as A, but is driven by the point x at $(x_0 + h, 0)$ (Fig. 6.20).*

The right and left limit of the slope of ℓ as x goes to x_0 will be identical to the right and left limit of $f'(x)$. Actually, we just constructed the line which is used to define the derivative of a function. □

This completes the proof of Thm. 6.23.

With some extra work we could give constructions that jump if any derivative of a constructible function is not continuous. However, with the tools presented in the next chapter this will be an easy consequence, and so we will avoid the technical details here.

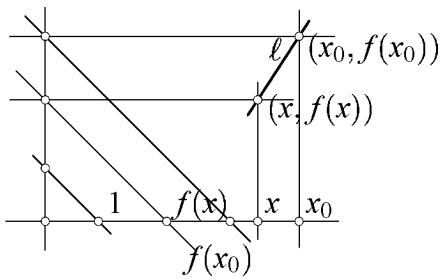


Figure 6.20: A construction that constructs a line with a slope that converges like the derivative of $f(x)$ at x_0

Chapter 7

Complex Tracing

The last chapter illustrated the fundamental problem when trying to achieve continuity in a Dynamic Geometry system: Adding construction steps can change the parameter space of a construction – as it happens when we iterate angular bisectors.

But this problem occurs and has been solved in other areas. Probably the most prominent one is the theory of Riemann surfaces, where the *definition space* of a function is extended as necessary. Like adding construction steps in Dynamic Geometry composing functions will change the definition (or parameter) space. The role of the iterated angular bisectors in Dynamic Geometry is taken by iterated square roots over $\mathbb{C}^*$ in Complex Analysis.

This chapter will show how close the relations really are, and how methods of complex analysis can be used to solve the problem of continuity in Dynamic Geometry.

7.1 A parameterization of the input space

As a first step we have to clarify what we mean by a dynamic parameter change. If we look at a geometry software we see that the input to a motion will not be a continuous path, but a series of *mouse events*, that give a sequence of new positions for a point (or another object, but we will restrict ourselves to movements of points right now).

The most natural approach is to translate this discrete information into a piecewise linear path for the moving element. In Fig. 7.1 this path is shown: We connect each pair of subsequent mouse positions by a line, and we assume that the point moves continuously on that line. The whole problem is reduced to being able to move a point from A to B along a straight line.

We will use a parameter $\lambda \in \mathbb{R}$ that runs from 0 to 1 to describe this move along a segment AB, and for all $\lambda \in [0,1]$ we assume that the moving point P will have the coordinates

$$P(\lambda) = (1-\lambda)A + \lambda B.$$

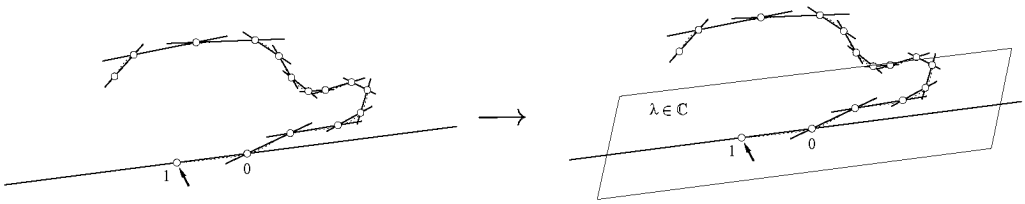


Figure 7.1: Instead of getting a continuous path we will get a series of discrete mouse events. The continuous path we will consider is just one piece of the piecewise linear parameterization of a moving element. The main idea is now to move from the left picture to the right one, to a complex parameterization of the Input: We will allow detours through complex space for the parameter λ. You can imagine this as having the mouse cursor step a little bit outside the screen to avoid a singularity.

7.2 The Main Idea

As we have seen in Sec. 6.1.2, we can carry out all basic operations for points, lines and conics using addition, multiplication, subtraction, square and third roots over the field of complex numbers. We also have a strategy of keeping track of decisions while dynamically changing parameters. If we move a point from A to B we will use intermediate points $(1-\lambda)A+\lambda B$ with $\lambda \in [0,1]$ for making the right "near-to" decisions. The only problem with this approach is that we cannot distinguish points after they passed through a singularity – there seems to be no way to prefer one assignment to the other, but yet we have to, as shown in the last chapter.

How can we handle these degenerate situations, how can we avoid the singularities? The answer is, we will just avoid them by not walking through them. We can choose another path from A to B than the segment $[0,1]$. All basic operations still work for complex numbers, so *we can choose any path* $p\colon [0,1] \to \mathbb{C}$ *from* $p(0)=0$ *to* $p(1)=1$ *in the complex plane*. We use this path as a parametrization for the movement from A to B; the intermediate points now lie at $(1-p(\lambda))A+p(\lambda)B$ with $\lambda \in [0,1]$.

7.2.1 A small Example

Before we will prove that we can always avoid degenerate positions, we will have a look at one of the examples of Sec. 6.3.2. What will happen to the singularity in Fig. 6.15 if we change from an ordinary, real movement to a complex path?

Here is the setup: Let C be the unit circle around 0 with equation $x^2+y^2-z^2=0$. Let $\ell(\lambda)$ be the line $x-\lambda z=0$. If we try to trace the two intersections of $\ell(\lambda)$ and C from 0 to 2 (using real values for λ only), we will crash into the singularity at $\lambda=1$ (Fig. 7.2).

If we change the path $t \mapsto 2t$ of λ to the path $t \mapsto 1-e^{i\pi t}$, we will avoid the singularity, as you can see in Fig. 7.3.

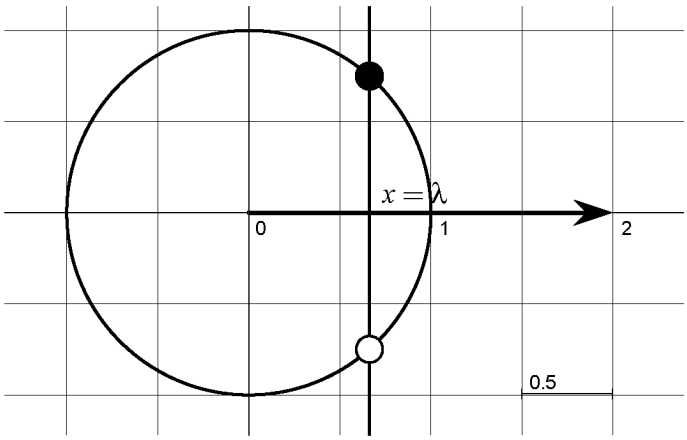


Figure 7.2: When the line moves from $x = 0$ to $x = 2$ along $x = \lambda$, $\lambda \in [0,2] \subset \mathbb{R}$ we will hit a singularity at $x = 1$.

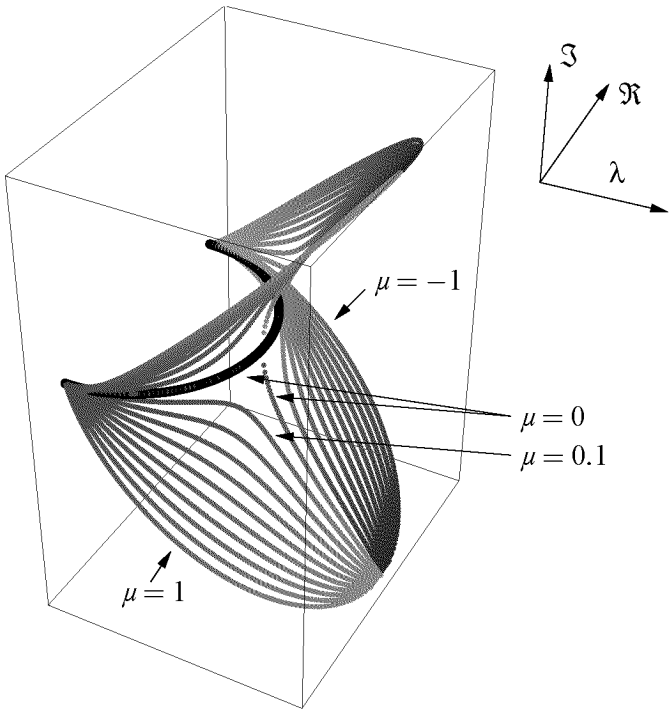


Figure 7.3: If we choose a path $\gamma_\mu(t) = 2(\sin(\pi t/2) + \mu i \cos(\pi t/2))$ for $\mu \neq 0$ we can avoid the singularity at $(1,0)$. The figure shows a 3-dimensional plot of the complex intersections of the line with the circle for different μ's.

7.3 Complex Tracing

Now we will formalize the idea of "complex tracing" presented above. As a first step we will transform a GSP on the homogeneous RIS to a GSP on a very basic RIS that encodes the λ-parameterization along a (complex) line. This RIS uses elements of $\mathbb{C}$ as its objects and can do additions, multiplications, subtractions as well as square root and third root operations as the inverse of second and third powers.

Next we will see how we can assign to any instance of a GSP on the basic RIS a Riemann surface that completely determines the instance we will reach when we move along a path. Finally, we will prove the important fact that we can always find a complex path that avoids singularities if we do not start or end in a degenerate situation.

7.3.1 Reparameterization

We have already seen that it is both useful and sufficient to work with only one complex parameter λ instead of allowing all free elements to be moved at the same time. We will further reduce the complexity of the objects involved in constructions by moving to the coordinate level. Here is a very basic RIS for calculations within the complex numbers, that is as powerful as the homogeneous RIS for points, lines and conics. It is a version of the RIS in Ex. 4.4 simulating straight-line programs with additional root operations.

Definition 7.1 (Basic Complex RIS) *The* basic complex RIS *is a RIS as in Ex. 4.4 over $A = \mathbb{C}$ with the additional operations for square and cubic roots:*

$$\sqrt{\cdot}^{1} := \{(a,b) \text{ s.t. } a = bb\} \qquad \text{(square root)}$$
$$\sqrt[3]{\cdot}^{1} := \{(a,b) \text{ s.t. } a = bbb\} \qquad \text{(cubic root)}$$

As we already did when we showed the equivalence between straight-line programs and the homogeneous RIS for points and lines, we will now transform a GSP over the homogeneous RIS for points, lines *and conics* to a GSP over the basic complex RIS.

Transformation 7.2 (Homo. RIS for points, lines and conics to basic complex RIS) *The translation of the primitive operations that work on vectors and scalars only is exactly the same as in Trans. 5.6 on page 53.*

The translation of the primitives DegConic, Eval, LinComb, CrossOp *and* MatrixEval *is also straightforward.*

The Split *primitive is translated by inserting the appropriate code to calculate α as in Eq. 6.16 and then using the $\sqrt{\cdot}$-primitive to get $\pm\sqrt{\alpha}$. This can be used to find a matrix by inserting instructions to calculate*

$$M = ((L^p)^T C \pm \sqrt{\alpha}\mathbf{1})L^p \qquad (7.2)$$

The first row of M is filtered and is available as the result of the Split *operation.*

The Radical *primitive is translated similarly. Expanding the* $\det(D) = \det(\mu C_1 + \kappa C_2) = 0$ *condition gives a polynomial equation of degree 3 for μ and κ. This can be solved algebraically using third roots and square roots. We omit the lengthy formula here, but assume that the corresponding code is inserted.*

A few remarks are necessary at this point:

- The translation of the Radical-primitive can be designed to be division-free by choosing suitable formulas for μ and λ.
- We cannot translate any instance of a homogeneous RIS GSP to an instance of the basic complex RIS. This happens because all scalar multiples of an output to the Split or Radical primitive are also valid outputs. But, it is possible to find another, equivalent instance in the sense that the dependent objects are scalar multiples of the objects in the original instance.
- By fixing the first row of M in the translation of the Split operation we might loose some information. In the case where the first row is the zero vector we usually choose another row in order to find the intersection point. It is important for us not to do this change, as it is crucial to have a fixed calculation without branches. In the next sections we will see that we will not loose too much, since the case that all three entries of a vector become zero will only happen for very few input values.

We still have to change the GSP such that it is parameterized by a single parameter λ. This is done in the obvious way: Given any GSP on a basic complex RIS and two assignments for the input variables, we can create the linear interpolation between these two by a little additional construction. The resulting GSP will have only one input variable.

Definition 7.3 (Reparameterization) *Given a GSP (X,R,Γ) on a basic complex RIS, an instance $(\tilde{X},\tilde{R})$ and a second assignment $\hat{X}$ of objects to the input, the* reparameterization of (X,R,Γ) from $\tilde{X}$ to $\hat{X}$ *is a GSP $((\lambda),R',\Gamma')$ that is derived from R and Γ by*

- *adding instructions to the beginning of the GSP that calculate $(1-\lambda)$ and all constants that correspond to the objects in $\tilde{X}$ and $\hat{X}$, and*
- *adding instructions to calculate $(1-\lambda)\tilde{X}_i+\lambda\hat{X}_i$ using the above calculations, and*
- *replacing all references to $\tilde{X}$ with references to intermediate results $(1-\lambda)\tilde{X}_i+\lambda\hat{X}_i$.*

Now we are at a mathematical level where each coordinate of the objects in the original GSP on the homogeneous RIS is described by an output variable of the reparameterized GSP. All coordinates can be obtained by repeated application of additions, multiplications, inverses of squares and cubes to constants and λ. We must take into account that the inverses of squares and cubes are not given by functions, but by relations, and this

means that we cannot just change λ to 1 and find the right coordinates of all elements by evaluating the GSP.

The only completely known instance is the one at $\lambda = 0$, since we can read it off the one we put into the reparametrization. In the Dynamic Geometry software setting this is the instance the user constructed, where he explicitly made the choices in case of ambiguities. How can we, starting at $\lambda = 0$, make our way to $\lambda = 1$?

7.3.2 Continuations and Riemann surfaces

The answer to the question in the last paragraph is "by walking from 0 to 1." In Sec. 6.3 we formulated a heuristic approach to continuity that used this method. Now we will formalize this approach. Whenever we will speak about a GSP, we mean a reparameterized GSP using the notations of Def. 7.3.

The approach presented now is in fact not really new (although we do not know a reference for a complete formal description). Felix Klein mentions in his book "Development of Mathematics in the 19th Century"' [41] that we are in the lucky position today that we do not have to *believe* Poncelet's Principle of Continuity, but we really *know* it works and even how, using analytic functions. So in a way we are rebottling old wine, it is a rediscovery of a well-known theory that so far nobody applied to Dynamic Geometry. In fact, we think that Dynamic Geometry is a very intuitive approach to Riemann surfaces, and therefore we also try not to clutter the beautiful mathematics with too much formalism.

Let us first assume that we have the maximal possible number of instances of (λ, R', Γ') with $\lambda = 0$ and $\lambda = 1$, i.e. all inverse squares and cubes are applied to a complex number different from 0 in both the start and the end position. In this case we can find a ε-neighborhood $U_\varepsilon(0)$ of 0 such that all intermediate results of the GSP can be described by analytic (in particular continuous!) functions f_i. What we are looking for is a path from 0 to 1 along which we can continue the function and find the right instance at 1 by analytic continuation of the f_i. This will give a continuous function for all coordinate functions of the original construction.

Back in the geometric world we know such a path whenever there is no singularity blocking our way from one position to the other, we just take the path along the real segment $[0,1]$ (the identity). We will prove now that it is always possible to find a – possibly complex – path from 0 to 1. For this we will use complete analytic continuations, Riemann surfaces and the identity theorem for analytic functions.

Here is the definition of the Riemann surfaces associated to the intermediate results of a GSP.

Definition 7.4 (Riemann surfaces for a GSP on a basic complex RIS) *Let $((\lambda), R, \Gamma)$ be a GSP on a basic complex RIS and let $(0, \tilde{R})$ be an instance of (λ, R, Γ) at $\lambda = 0$. Furthermore, assume that there is a region $G \subset \mathbb{C}$ containing 0 such that there is an analytic function f_i for every R_i with $(\lambda, f_1(\lambda), \ldots, f_n(\lambda))$ being an instance of $((\lambda), R, \Gamma)$ at every*

$\lambda \in G$. *Then the* Riemann surface of R_i, *denoted* $X(R_i)$, *is the Riemann surface* $X(f_i)$ *defined by* f_i.

The Riemann surfaces $X(f_i)$ define complete analytic continuations of the f_i up to isomorphism (Thm. 3.5 in [15]). This means we can associate to every variable an analytic function on its Riemann surface.

Definition 7.5 (Complete analytic continuations for basic complex RIS GSPs) *Using the notations of Def. 7.4, the complete analytic continuation* $\hat{f}_i : X(R_i) \to \mathbb{C}$ *is the unique (up to isomorphism) complete analytic continuation defined by* $X(R_i)$.

The fibers of the Riemann surface lie above all the points of $\mathbb{C}$ that define non-degenerate constructions. Non-degenerate constructions are those where we do not have to find the square or cubic root of 0, you can (and should) imagine them as all the situations where no ambiguous intersection points collapse, like in the case of a line becoming tangent to a circle.

Definition 7.6 (Singularity of a basic complex RIS GSP) *A* singularity *of a basic complex RIS GSP is a point in the complement of the set of all* λ_0 *in* $\mathbb{C}$, *where the fiber over* λ_0 *is non-empty for all* $X(R_i)$.

Let us prove that under the assumption that we are in non-degenerate (or non-singular) situations at $\lambda = 0$ and $\lambda = 1$ we will always find a path from 0 to 1. The proof uses induction on the number of instructions of the GSP, and also uses the identity theorem for Riemann surfaces, which we will briefly recall.

Theorem 7.7 (Identity Theorem for Riemann surfaces) *If two analytic functions* $\varphi : X \to Y$ *and* $\psi : X \to Y$ *are identical on a non-discrete subset* $N \subset X$, *then* $\varphi \equiv \psi$.

Proof 7.8 *For a proof see any book on complex analysis on Riemann surfaces, e.g. [15].*

Here non-discrete means having no accumulation point. We are ready to prove the main theorem of this section:

Theorem 7.9 (Complex Connectedness Theorem) *Let* $((\lambda), R, \Gamma)$ *be a GSP on a basic complex RIS satisfying the conditions of Def. 7.4. Then either the GSP is in a singular position at* $\lambda = 1$, *or we can find a path for* λ *from* 0 *to* 1 *that avoids all singularities of all* $\hat{f}_i$. *Moreover, in the later case all singularities are discrete.*

Proof 7.10 *We prove the theorem by induction on the length of* $\Gamma = (\Gamma_1, \ldots, \Gamma_r)$. *If* $r = 1$ *we are done, because the only function we have is the identity function.*

Let us assume that the theorem is true for $i < r$. *If* 1 *is a singularity, then we are done, so let us also assume that* $\hat{f}_i$ *is a holomorphic function on* $X(R_i)$ *for all* $i < r$. *If the last instruction* Γ_r *of the GSP is one of the operations addition, subtraction, multiplication, or*

constant, the resulting Riemann surface will introduce in the worst case all the singularities that were already present in the Riemann surfaces of their operands, which means that nothing has changed.

If the Γ_r is a square or cubic root, it introduces singularities at all zeros of their input. If the input is zero for all elements in the fiber of $\lambda = 1$ we are done because of the singularity at 1. If the input is different from 0 for one element z_0 in the fiber of $\lambda = 1$, it cannot be zero on a non-discrete set, because then it had to be identical to zero, a contradiction. So we are also done, because there is a path from 0 to 1 avoiding all singularities.

Remark 7.11 *The instance of the GSP we reach at 1 depends on the path taken, or better, on its homotopy class. If we walk close to the real line we will either use a path that is homotopic to direct path on the real line, if there is no singularity on the real line, or we can at least find a series of homotopic paths from 0 to 1 that converges to the path on the real line. Thus a geometry software based on complex tracing will always look like it is doing the right thing – given that the complex path does not "catch" additional singularities. However, even if it catches a singularity, we will end up at a situation that is "mathematically correct."*

Remark 7.12 *We excluded singularities by identifying the instances of a GSP where we construct the root of 0. These singularities can be canceled out in the case where the function $\hat{f}_i$ that gives the argument of the root is identical to zero. You get an equivalent non-singular GSP by just omitting the trivial root operations and shifting the rest of the program.*

Definition 7.13 (Reduced GSP) *A* reduced GSP *is a GSP where every global singularity has been canceled out.*

Finally, we are able to define what we mean by a *continuous move* of a GSP on a homogeneous RIS on points, lines and conics, using the analytic continuations just introduced.

Definition 7.14 (Continuous Move) *A* continuous move *of a configuration given by a GSP (X,R,Γ) on a homogeneous RIS on points, lines and conic is a change of the input $\tilde{X}$ to another input $\hat{X}$ with instances $(\tilde{X},\tilde{R})$ and $(\hat{X},\hat{R})$ that is induced by a complete analytic continuation of the associated reparameterized basic complex RIS GSP, i.e. the transformation of (X,R,Γ) with the instance $(\hat{X},\hat{R})$ will give an instance of the reparameterized GSP that can be reached by an analytic continuation along a continuous path.*

7.4 Automatic Theorem Checking

We will now present an approach to automatic theorem proving. We intend to use it for geometry theorem proving as we already did with points and lines in Ch. 5 using

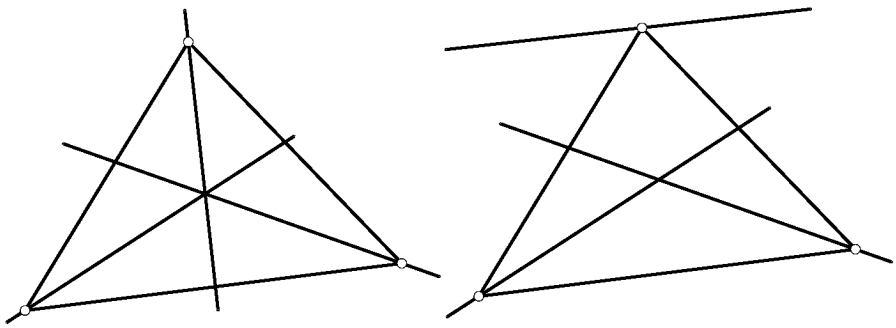


Figure 7.4: Two instances of the angular bisector theorem – one is obviously a bad choice

polynomial descriptions of theorems and the variations on the fundamental theorem of algebra.

Before we procede, we must fix what we mean by "theorem." Is "The angular bisectors in a triangle meet in a common point" a theorem? Different people might come up with different opinions on this, since there are some choices in the construction that rule about the truth of it. In Fig. 7.4 we see two instances of a triangle and its angular bisectors, one shows the theorem, the other one does not.

In a continuous geometry software we actually want theorems to be true always, if they are for the neighborhood of one instance. By moving base elements we should not be able to destroy the theorem. Let us define a geometry theorem by requiring a non-singular instance and that it is true for all instances we can reach by continuous moves without hitting singularities.

Definition 7.15 (Dynamic Geometry Statement) *A* Dynamic Geometry Statement (DGS) *is a GSP on a homogeneous RIS on points, lines and conics together with an instance* $(\tilde{X},\tilde{R})$ *, where the last instruction has a scalar ouput. It is* true *for the instance* $(\tilde{X},\tilde{R})$ *if the scalar is* 0 *at* $\tilde{X}$*, it is* false *otherwise. If the corresponding GSP is reduced, we call the DGS* reduced.

Definition 7.16 (Dynamic Geometry Theorem) *A* Dynamic Geometry Theorem *is a Dynamic Geometry Statement that is true, and for which all instances that can be reached using subsequent continuous moves are also true.*

As stated in Def. 7.14, we are allowed to make several subsequent moves but we may not "stop" at singularities. This would obviously destroy any meaningful notion of theorem, because you could move a construction into a completely degenerate position (in our angular bisector theorem example we could move all three vertices of the triangle to

the same position) and then move everything apart again. While we rest at the singularity (and change moving elements), we do not have any control over the choice of the angular bisector (it is the "invalid line" with coordinates $(0,0,0)$).

You should also keep in mind that we can arrive at different instances of a configuration by using different sequences of linear paths. Actually, making continuous moves is not even commutative – we can arrive at a different instance if we first move A to A' and then B to B' compared to moving first B to B' and then A to A'.

How can we do some kind of automatic theorem proving for geometric theorems as in Def. 7.16? An obstacle here is that we do not have a fundamental theorem of algebra for square and cubic roots, and we cannot give immediate bounds for the number of examples we need to prove that a theorem ist true (there is still a chance to find these bounds, see Ch. 12).

What we can prove is the following theorem that is just a formalized version of Poncelet's Principle of Continuity:

Theorem 7.17 (Continuity of Geometric Theorems) *Let $(\tilde{X},\tilde{R})$ be the instance of a reduced Dynamic Geometry Statement, and let the DGS be true at $\tilde{X}$ and at all other instances in a neighborhood of $\tilde{X}$. Then the statement is a theorem and true for all instances that can be reached using subsequent continuous moves.*

Observe that we cannot use a simple induction to prove this theorem, because we cannot make turns between the linear pieces. Also, we cannot just take the short-cut and use only one linear interpolation, because we could loose reachable instances by that. But we can use the simple observation that we have a neighborhood of the piecewise linear path which can be used to smoothen it.

Proof 7.18 *Since the continuous moves do not hit the discrete singularities, there is an open neighborhood of the linear interpolations without singularities. This means that there is a sequence of analytic paths within that neighborhood that converges to the piecewise linear path of continuous moves. For all these analytic paths the complete analytic continuation of the DGS scalar is identical to zero, and thus also for the piecewise linear path.* □

At this point it would be interesting to consider not only paths of one complex variable λ, i.e. slices of the configuration space of a geometric construction, but the complete space, given by all free variables of a GSP on the homogeneous RIS for points, lines and conics. But this is beyond the scope of this thesis and will be presented elsewhere.

7.5 Complexity issues

As the last section in the mathematical part we want to outline some results on complexity issues in Dynamic Geometry, which will be presented in full detail in [72].

The general question is:

> "Given two instances of geometric construction (given as a GSP), can we move continuously from one to the other using only pathes of a certain type?"

The answer depends on both the RIS that we use, as well as on the type of moves we allow.

For determined relational instruction sets the answer is easy for a suitable definition of continuous move, since in that case we can just check whether both instances actually are instances of the GSP. If so, we can move from one to the other, otherwise we cannot.

Let us briefly summarize some results for constructions using points, lines, and angular bisectors under the assumption of continuity as defined above. Recall that the instance we will reach while doing a series of analytic continuations along a certain path is unique if we do not hit singularities.

We will now restrict the paths we can use for moving from one instance at A to another instance at B, thus restricting the instances we can reach at B, and we will ask how hard it is to decide whether we can reach a certain given instance at B.

As we know already, it is not always possible to move from A to B on the real line given by $(1-\lambda)A+\lambda B$, $\lambda \in \mathbb{R}$, without hitting a singularity. Nevertheless, we can show that there are constructions where you will not hit a singularity and it is NP-hard to decide whether you will reach at given instance at B. The proof uses a transformation of the 3-SAT problem to a geometric construction on points, lines and angular bisectors.

The same proof also holds for paths in an ε-neighborhood of a real path (a complex cylinder), and since one can avoid all singularities using paths like that we have a nice result on the real reachability problem of constructions.

We do not have a result for the complexity class of the reachability problem if we do not restrict the paths, i.e. if we may use any path through complex space. It still could be that there is an easy way to check whether we can find a path connecting A and B, and in fact it would be a very useful result to have: If we want to do randomized theorem proving on points, lines and conics as we did on points and lines only, creating a number of random examples and checking each instance, we need a way to create these examples using a certain random distribution among a large number of possible examples. Up to now we only can get a reachable and thus valid instance at some other position by walking there along a path, and this is computationally expensive. If we could create a random *instance* instead of a random *input* only and check whether it could be reached in a reasonable time, we would have a much better chance to do randomized theorem verification in, say, polynomial time.

Let us reformulate the problem in a way that it can be attacked without having to worry about geometric constructions:

Problem 7.19 *Given a GSP $((\lambda),R,\Gamma)$ on a basic complex RIS, i.e. an instruction sequence using complex additions, multiplications, square roots and cubic roots. Let $(\tilde{\lambda},\tilde{R})$ and $(\hat{\lambda},\hat{R})$ be two instances. What is the algebraic complexity of the decision problem: Is there a complex path $\gamma\colon [0,1]\to\mathbb{C}$ from $\tilde{\lambda}$ to $\hat{\lambda}$ that avoids singularities, starts at $(\tilde{\lambda},\tilde{R})$ and ends at $(\hat{\lambda},\hat{R})$ such that all functions $\hat{f}_i$ induced by the R_i are continuous?*
– or: ***Can you get lost on a Riemann surface?***

Chapter 8

Java-based Software

In this chapter we will give a short overview about our choice of the programming language, and we will discuss the most important features and drawbacks that influenced the design and the coding of *Cinderella*. The experiences with Java are in part special to *Cinderella*, some others will apply to any Java geometry software, and some are true for any Java software. Many of the aspects are discussed in more detail in [48].

8.1 Why choose Java?

For every software project the programming language(s) to be used will influence the whole project. It is a major design decision, and it should be taken with care. In our case it was more a decision by heart, triggered by various circumstances, and decided within two or three days, but we would make the same decision again if we had to. In this section we want to present the reasons why we chose Java, and why this could be the right decision for other projects, too.

8.1.1 The History of *Cinderella*

Let us start with a (slightly personal) review of the history of *Cinderella*. The *Cinderella* project was started in 1992 by Jürgen Richter-Gebert and Henry Crapo, with phases of more and phases of less progress. The software was written for the NeXT platform, a very innovative, object-oriented computer system that – like other good computer systems – was not as successful as it deserved to be. The decision for the NeXT and its proprietary application builder tool, Objective-C and Display PostScript made it hard to port the software to another platform. Unfortunately, the decreasing availability of NeXT computers made it very difficult to demonstrate the software, and the problems culminated in July 1996 at the Mt. Holyoke conference on Discrete and Computational Geometry. Although there were several NeXT computers available, none of them could be used at first, because they were either running an outdated version of the operating system, or they could

not be connected to the projection device. It was only in the last minute that a suitable machine was available. The demonstration at the conference finally went very well, but it was clear for Jürgen when he returned to Berlin that he will never again give a software demonstration that relies on a computer system that is – although superior to most, if not all, of the competing systems – not used by enough people.

But on the other hand, the conference was a big success and the response of the audience was very friendly. And Jürgen was invited to give a talk at the CGAL startup meeting at ETH Zürich, where he wanted to present the software again. See [62, 14] for more information about the CGAL project.

It was clear that the program had to be ported or rewritten in another language. But it was also clear that the new language had to be object-oriented. At that time Java was just at the beginning of its career. Nobody at our institute (the math department of the Technical University Berlin) knew it, but it was claimed to be portable and easy to learn. But there were also rumours that it is much to slow to do any serious work. After thinking about it for a while we decided to give it a try. A first *applet* that supported points and lines 8.1 was very convincing: Java is fast enough, and we will try it. And it looks like it was the right decision: 3 years later many people are not only talking about, but they actually *use* Java (and *Cinderella*!).

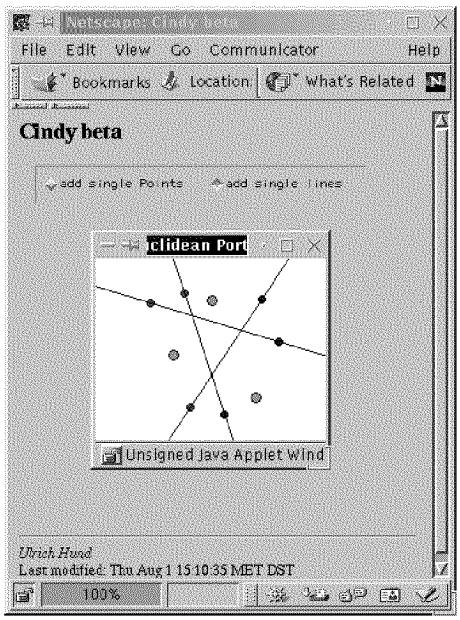


Figure 8.1: The first Java-based version of *Cinderella*, dating back to August 1996

We started to write the new version of *Cinderella* from scratch. Within three weeks we created a demoable Java version *Cinderella*, that almost replaced the old NeXT-*Cinderella* – at least it was good for the CGAL startup meeting demonstration. After three other weeks of development we submitted *Cinderella* for the MultimediaTransfer '97, a competition that should stimulate communication between industry and science. Another three weeks in January 1997 made the program good enough to be named "Most Innovative Multimedia Software." At that point the software had many features that could fill a demonstration, it was reasonably stable and well designed.

But still a lot was missing: It was not possible to print or even save constructions, the user interface needed a brush-up, measuring was not implemented at all (you could draw Euclidean, hyperbolic and elliptic circles), the interactive exercises were just a "technology preview" and could not be created without a lot of hacking, and there were too many (even critical) bugs that we knew how to avoid, but had to fix someday.

Nevertheless we were, based on our "rapid application development" experience with Java, convinced that we could do this in a few months of work. At the same time Springer-

Verlag and HEUREKA-Klett started negotiations with us about a publication of the software.

The rewrite started with the introduction of *algorithms* (see Ch. 9) and improvements on the user interface. We switched from Java 1.0.2 to Java 1.1, which meant that we had to rewrite both the mathematical kernel and the user interface.

After a year of negotiations and rewriting we were almost finished with both. But then we realized that we had to rewrite everything again: The continuity theory (Ch. 7) required that everything had to be implemented with complex numbers. Also the kernel became much more complex, the adaptive step-width algorithm for complex tracing (Sec. 9.2) needs much more effort than the easy update loop as found in usual Dynamic Geometry software.

The first running prototype of complex tracing worked in March 1998, and it took a few months until it was stable enough to be included in a commercial product. During 1998 we also added many other features on request of the publishers, the beta-testers and ourselves. The current version of *Cinderella* has not changed significantly since December 1998.

8.1.2 Rapid Application Development

The most important "feature" of Java that is often underestimated is that it is very easy to write Java software, and to create large (there are more than 55K lines of source code in about 340 classes, approximately 900 kilobyte that were written by only two persons) in projects from scratch. Here we just want to recommend Java for both academic and commercial purposes. The total cost of programming work (compare to total cost of ownership for operating systems) is very low according to our experience. We could create *Cinderella* using free and open-source tools only, XEmacs and CVS to mention the two most important ones. Development was done using computers running Solaris and Linux. We recommend the German book "Erfahrungen mit Java" and in particular the chapter about *Cinderella* [48] for more information about Java success stories (including information on the drawbacks and ways around them).

8.2 Deploying Java Applications

Although Java applets are easily distributable over the internet with the help of a standard web browser, there are no technical provisions in the Java specification that make it easy to distribute and install Java applications. Everything depends on a correctly installed Java VM on the host operating system, and there is no platform independent file system layer that can be used for additional resources like configuration or data files.

Another drawback of standalone Java applications – distributed as shrink wrapped packages – is that they do not offer the performance the user expects from a native package. Software running within a browser is not expected to be as fast as standalone appli-

cations, but if we hide the Java environment from the user and present a Java application outside a browser we have to try to create a software that performs best possible.

In this section we will describe two ways how we addressed these two issues, using post-optimization and a third-party installer tool. These two together make Java suited for shrink-wrapped software.

8.2.1 Post-Optimizing Java Applications

Java has to suffer from performance penalties due to the dynamic linking at runtime and the interpretation of Java byte code. These two together avoid most of the optimizations usually done at compile time. Inlining of code or jump optimizations by devirtualizing, common optimizations done in C++, are not possible across different classes.

Most of the dynamic loading features are never used in an average Java application. The classes that will be loaded are fixed, not at compile-time but at least at deployment time. *Cinderella* is installed as a jar-file containing all classes of the application. Whenever there would be a change that could create problems for inline optimizations or virtual function call resolution we could redistribute a complete new archive with all the changed classes.

In fact, we first started to do these optimizations by hand, for example in the base classes for complex number calculations. We replaced every virtual access and every method call with direct references to variables in ordered to avoid the enormous overhead introduced by the JVM. This created unreadable and unmaintainable, "write-once-read-never" code (Table 8.1), which was only justified by the performance gains by a factor of more than 3.

The Java technology since then has matured. Just-In-Time-compilers can do some optimizations that formerly have been done at compile time at runtime, since they do the whole compilation at runtime. The price we have to pay for these optimizations is a slow startup of Java applets and applications, which is very annoying since the JIT-compiler does the same compilations over and over again, for every new start of the program. As far as we know there is no JIT compiler that caches compilitations over sessions, and it does not look like a trivial task to implement a caching strategy that can survive the lifecycle of JVM instances. Another problem is that not every platform offers a JIT-compiler, and even those available differ extremely in performance.

The approach we take in *Cinderella* is to do the optimizations after the whole project has been compiled and frozen. At that point we know which classes will be loaded dynamically, which code may be inlined and how method calls can be devirtualized. Many other – standard – optimization techniques may be applied, like loop unrolling or constant expression elimination.

For *Cinderella* we used a tool supplied by IBM alphaworks called JAX [78], an acronym for Java Application eXtractor. *Cinderella* was used as a test suite by the JAX team [79]. JAX does a class hierarchy analysis on the code, using the additional information about dynamically loaded classes and dynamically invoked methods, and then it

```
public Vec solveCubic(double ar, double ai,
                      double br, double bi,
                      double cr, double ci,
                      double dr, double di) {

    // t1 = (4ac - b^2)
    double acr = (ar*cr-ai*ci);
    double aci = (ar*ci+ai*cr);
    double t1r = 4 * acr - (br*br - bi*bi);
    double t1i = 4 * aci - 2 * br*bi;

    // ab = ab
    double abr =  (ar*br - ai*bi);
    double abi =  (ar*bi + ai*br);

    // t3 = t1 *c - 18 ab * d = (4 ac - b*b)*c - 18 abd
    double t3r = (t1r*cr - t1i*ci) - 18 * (abr*dr - abi*di);
    double t3i = (t1r*ci + t1i*cr) - 18 * (abr*di + abi*dr);

    // aa = 27  a*a
    double aar = 27 * (ar*ar - ai*ai);
    double aai = 54 * (ai*ar);

    // aad =  aa *d = 27 aad
    double aadr = (aar*dr - aai*di);
    double aadi = (aar*di + aai*dr);

    // t1 = b^2
    double bbr =  (br*br - bi*bi);
    double bbi = 2 * br*bi;

    // w = b^3
    double wr = (bbr*br - bbi*bi);
    double wi = (bbr*bi + bbi*br);

    // t2 = aad + 4w = 27aad + 4bbb
    double t2r = aadr + 4 * wr;
    double t2i = aadi + 4 * wi;

    // t1 = 27 *(t3 * c + t2 *d)
    t1r =  (t3r*cr - t3i*ci + t2r*dr - t2i*di);
    t1i =  (t3r*ci + t3i*cr + t2r*di + t2i*dr);

    // DIS OK!!
    // w = -2 b^3
    wr *= -2;
    wi *= -2;

    // w = w + 9 a b c
    wr += 9 * (abr*cr - abi*ci);
    wi += 9 * (abr*ci + abi*cr);

    // w = w + -27 a*a d
    wr -= aadr;
    wi -= aadi;

    // t1 = (27 dis).Sqrt()
    t1r *= 27;
    t1i *= 27;
    t2r = Math.sqrt(Math.sqrt(t1r*t1r+t1i*t1i));
    t2i = Math.atan2(t1i,t1r);
    t1i = t2r*Math.sin(t2i/2);
    t1r = t2r*Math.cos(t2i/2);

    // w = w + a * dis // sqrt war schon oben
    wr += t1r * ar - t1i * ai;
    wi += t1r * ai + t1i * ar;

    // w ausgerechnet. Jetz w1 und w2
    //     w1.assign(wr,wi);
    //     w2.assign(wr,wi);
    //     w1.sqrt1_3();
    //     w2.sqrt2_3();
    double radius = Math.exp(Math.log(Math.sqrt(wr*wr+wi*wi))/3.0);
    double phi    = Math.atan2(wi,wr);
    double w1i = radius*Math.sin(phi/3);
    double w1r = radius*Math.cos(phi/3);
    radius *= radius;
    phi *= 2;
    double w2i = radius*Math.sin(phi/3);
    double w2r = radius*Math.cos(phi/3);

    // x = 2 b^2
    // x = x - 6 ac
    xr = 2*bbr - 6 * acr;
    xi = 2*bbi - 6 * aci;

    //y.assign(-c2).mul(b).mul(w1);
    yr = -c2 * (br * w1r - bi * w1i);
    yi = -c2 * (br * w1i + bi * w1r);

    //    z.assign(c1).mul(w2);
    zr = c1 * w2r;
    zi = c1 * w2i;

    //w1.mul(a).mul(3).mul(c2);
    t1r = c2 * 3 *(w1r * ar - w1i * ai);
    t1i = c2 * 3 *(w1r * ai + w1i * ar);
    double s = (t1r*t1r + t1i*t1i);
    t2r = (xr*t1r + xi*t1i) / s;
    t2i = (-xr*t1i + xi*t1r) / s;
    xr =t2r;
    xi =t2i;
    t2r = (yr*t1r + yi*t1i) / s;
    t2i = (-yr*t1i + yi*t1r) / s;
    yr =t2r;
    yi =t2i;
    t2r = (zr*t1r + zi*t1i) / s;
    t2i = (-zr*t1i + zi*t1r) / s;
    zr =t2r;
    zi =t2i;

    return this;
}
```

Table 8.1: This code, which dates back to the time where we did not use JAX for post-optimizing the Java byte code and had to do manual optimizations, finds the roots of a polynomial of degree three. The comments should help to understand what is actually done, but whether they really can be questioned.

	Cinderella Applet				*Cinderella* Application			
	zipfile	methods	fields	classes	zipfile	methods	fields	classes
before:	896758	3927	2754	383	896758	4033	2817	412
after:	306176	1985	1530	247	398657	2527	1795	292
savings:	590582	1942	1224	136	498101	1506	1022	120
	65%	49%	44%	35%	55%	37%	36%	29%

Table 8.2: Code compression with JAX

eliminates unused variables and methods and changes virtual access to methods and variables to direct access whereever possible. This results in an enormous reduction of the code size: The standalone application *Cinderella* is around 55% smaller than the original code, and the applet's size is reduced to about one third (Table 8.2).

While the decreased size of the standalone application is mainly caused by the usual optimizations of JAX, the additional significant gain in the applet version is achieved by dead code elimination of parts that will never be used in the applet, like the "save construction" operation or the PostScript export. This is a major advantage of using a tool like JAX for post-processing: We can use exactly the same code for applet and application, and JAX will care about removing the parts of the program that are not necessary for the applet at runtime automatically. All we need is different configuration file.

As a test we created a runtime that only supports Euclidean views and the "Move" mode. All dynamic references to other views and modes were eliminated from the configuration. This runtime was only one third of the current applet runtime, one nineth of the original code size. This demonstrates the power of Java application extracting tools: You can use the same codebase for several applications and applets, giving you apparent advantages like increased stability and less administrative effort, while still getting compact and optimized code.

More information on successful compression with JAX can be found in the research report by the JAX authors [79]. This report also covers the compression of *Cinderella*, since *Cinderella* was used by the JAX team for performance and regression tests. As a side remark we want to mention that the size reduction could be increased further if another class file layout could be used (e.g., as suggested by Pugh [65])

8.2.2 Platform Independent Installations

The targeted user group of *Cinderella* is very inhomogeneous. We cannot assume anything about the platforms *Cinderella* will be used on. Despite the fact that we could make *Cinderella* run on any platform that supports Java 1.1, we cannot hope that the users will be able to make *Cinderella* run if we only provide the jar file containing all the classes. In that situation the use of a third party tool to create a cross platform installer seemed to be

a good choice.

We found the following requirements for us:

- The installer must be able to install a Java Virtual Machine on Windows 95/98/NT and MacOS, for the case that there is no JVM available.
- The installer must run on any Java-1.1 platform.
- The installer must support internationalization.
- The installation should "look like any other installation on Windows," that is, a Windows user must not be confused with details about Java.

There are several cross platform installers for Java available (InstallShield for Java [35], J'Express [11], InstallToolkit [33] and InstallAnywhere [92]). Of these, only InstallAnywhere is able to install a JVM on MacOS. This was reason why we chose InstallAnywhere (IA) as installation toolkit for *Cinderella*. It also supports all the other requirements we listed above and even has some more interesting features. The installers created by InstallAnywhere are of high quality and integrate seamlessly into the Windows operating system, and offer the same ease of use on other platforms, including Unix. This makes them usable also for non-experts.

Despite its features, IA has several drawbacks, which we want to mention.

1. **Not fully scriptable.** Our development environment is based on GNU make and almost everything in the build process is automated. IA can be run from the command line, but it is not possible to pass a machine generated configuration script to it. All configurations of the installer have to be done using the graphical user interface of IA, which is very time consuming and error prone.

   IA offers a concept called "speedfolders," which can partly automate the build process, but they are not powerful enough to suit our needs. A speedfolder defines a directory which should be recursively included in the distribution. Inclusion or exclusion rules (but not both) based on suffix matching can be defined. Even with these rules it can happen that empty directories are included.

2. **No version control support.** The binary file format of the IA configuration files is not suitable for true versioning with CVS. Even worse, IA saves its localization information in a separate directory, and if this directory is added to the CVS system the extra `CVS` directory in that directory causes IA to crash. These problems could be avoided with a fully scriptable version of IA.

3. **Incomplete internationalization support.** Although the most recent version of IA supports creating installers in six languages (additional languages are available separately), the internationalization support is far from being perfect. It is not possible to create an internationalized version of an already existing installer. There is no

built-in support for different custom messages for the different locales, this is only available via post-editing by hand of some configuration files.

With respect to internationalization we have to recommend J'Express, which is much more powerful in that area. Unfortunately, J'Express does not support installing a JVM on MacOS. Other remarkable features of J'Express which make it a good alternative if you can do without MacOS support are silent installs (which can be run unattended) and automatic updates.

With InstallAnywhere we were able to create an installer which can be run from CD-ROM or even remotely via the web. We are very satisfied with this installer, although the creation of it is not as easy as it could be.

Chapter 9

Efficient Datastructures for Dynamic Geometry

In this chapter we will present some of the underlying data structures of *Cinderella*. The design of these data structures is crucial for the performance and stability of the whole software. We will also show how the mathematical theory of complex tracing can be implemented in an efficient way.

We want to emphasize that this description is not as complete as a project description in computer science could be. However, it is important to see that this thesis is not primarily about the implementation of a geometry software, but about the underlying mathematical foundation. This chapter should not be taken as the one-and-only recipe to create a software like *Cinderella*, but as a collection of *hints how we did it and how it could be done* – that's the reason why we did not present everything in a way that you can type in the code immediately. This is different compared to the mathematical part: the mathematics are determined already by a couple of formal requirements the software should fulfill, in particular the continuity assumption. The actual implementation depends more on personal parameters, like "programming style" or "software development paradigms." We are proud to have an *elegant* implementation of the theory, that reflects at least a little bit of the beauty of the mathematics, but it is not the only way to do it.

9.1 Comparision to the traditional approach

First we will discuss why we did not use the classic approach [69, 58] to object-oriented geometry software, which is used in most software for dynamic geometry and other applications today. We want to say in advance that it is not our goal to replace the traditional approach – it is still suited for many applications. Only in the special case of sophisticated dynamic geometry software that should be able to handle also intersections of circles and conics we want to suggest to rethink the way it is implemented.

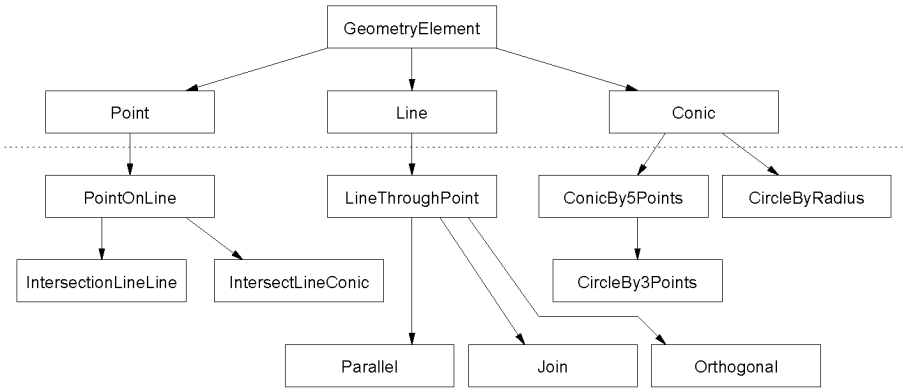


Figure 9.1: A part of the typical class hierarchy for a Dynamic Geometry System. A common root class is subclassed into the classes for different geometric objects, which are again subclassed in order to create the object definitions. *Cinderella* does not follow this hierarchy but replaces the lower class layer (below the dotted line) by the algorithm concept.

9.1.1 The Usual Approach: Subclassing

When you think about a class hierarchy for an object-oriented Dynamic Geometry system, you will probably end up with a tree like shown in Fig. 9.1. Common root of all geometric objects is a standard `GeometryElement` class. This class contains all the methods and fields that are useful for all geometric elements, for example information about the color and label of the element, and methods for showing the element on screen and to update it. Another important piece of information is a list of dependend elements.

Subclassed from this object are `Point`, `Line` and `Conic`, and maybe more, classes that correspond to the different object types that are available. These contain the additional data that is needed for them, like the coordinates for a point, the matrix of a conic, or the text of a text label, and override the display method.

The definition of an element, that is, the information whether it is a free point or a point of intersection, a free line, a parallel line or an orthogonal line, and so on, is maintained by another level (or more) of subclassing. The classes in this layer overload the recalculation method and introduce fields that store the additional information needed.

A mouse move is processed as follows: The coordinates of the moved element are changed to reflect its new position, and then all dependent elements are updated by traversing the list of (directly) dependent objects. All elements that have been updated add all their (directly) dependent elements to the list of elements that still need an update, unless they have been updated already. We omit the details, which are fairly standard. A much more detailed presentation of the internals of a traditional Dynamic Geometry software can be found in [58], using the software Euklid [59] as an example.

9.1.2 Elements and Algorithms

In *Cinderella* we moved away from the traditional class hierarchy and introduced a new architecture that is based on splitting up the geometric type, the visual representation and the constructive definition of the elements.

The class hierarchy also starts with common superclass for geometric objects, `PGElement` (the "PG" stands for "projective geometry"). This still contains some fields and methods that are useful for all types of elements, as well as certain administrative data. Subclasses of `PGElement` are `PGFlat` (which is subclassed again to `PGPoint` and `PGLine`), `PGConic` and other object types. But instead of subclassing these classes again to reflect the constructional definition of objects, every instance of `PGElement` stores an `Algorithm` object. All information about dependent elements, defining elements and update methods is encoded in these `Algorithm` objects.

Why did we detach the algorithms for updates from the elements? It turns out that the traditional subclassing approach is not feasible for efficient Dynamic Geometry software, and this holds in particular if we want to do complex tracing as described later in this chapter. The crucial point is that we will never calculate *one* intersection of a conic and a line, but always *both* intersections, we will never calculate *one* intersection of two conics, but all *four* of them. This means that in the case where we did define, say, a perpendicular bisector using two circles, we will have to find both intersections and select the right one of these *twice*. With the algorithm concept we do not have a strict "belongs to"-relation between geometric elements and their algorithms, but the algorithms just operate on the coordinates of several geometric elements, see Fig. 9.2.

The `Algorithm` approach of *Cinderella* is a convenient way to reuse the information we have already found, since the same algorithm instance is plugged into both `PGPoint`-objects. It also increases the stability of the selection process: It can never happen that two intersections collide suddenly (as it happens in the mirroring example, see Fig. 6.8 on p. 88).

An important question is how we identify whether we can use the same algorithm for two elements. In Sec. 9.3.1 we will explain it in detail, here we just mention that we use the internal theorem checking engine of *Cinderella*.

9.1.3 Data structures for updates

The mathematical kernel of *Cinderella* must update a construction whenever a free element has been moved. The movement of a free element itself is parameterized using a complex parameter λ. We will discuss in Sec. 9.2.1 how we can do the necessary reparameterizaton on the fly. Here we want to present how an update is propagated to all the dependent elements.

Since we do not store a list of dependent elements within the `PGElement` but maintain the dependency information inside the `Algorithm`-objects we must use another strategy as in the traditional class hierarchy situation.

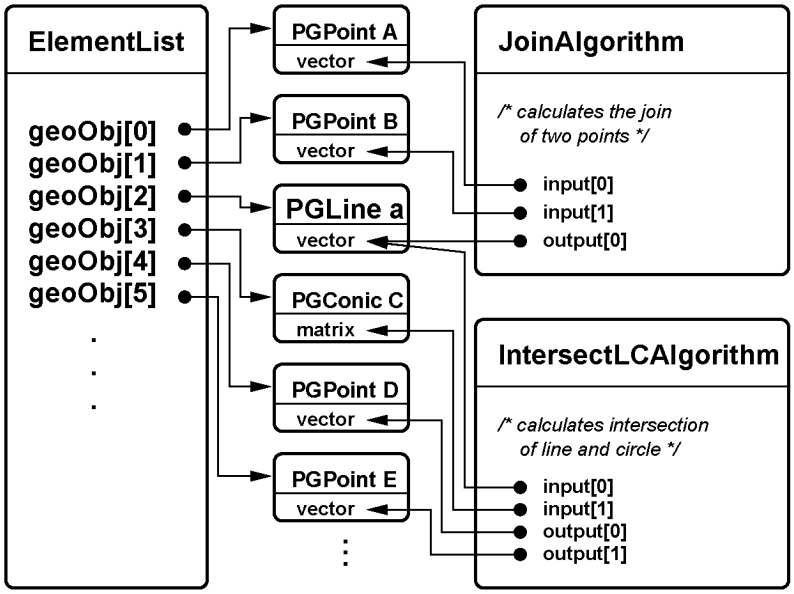


Figure 9.2: Part of the geometry kernel of Cinderlla. A list of geometric objects, which are instances of `PGElement`, is maintained. The algorithms, which do not belong to the geometric elements but are independet, operate on the coordinates that are stored inside the geometric elements. This makes it possible to reuse one algorithm for several geometric objects.

We want to minimize the time spent during an update, which suggests precomputing a list of algorithms to be updated when an element is moved. These list of "programs," is stored in a hash table referenced by the movable objects. Whenever an algorithm is added to the kernel we add it also to all programs that contain a reference to a defining element of the algorithm. Since we add the algorithm to the end of the list, we can be sure that the order of algorithms in the list is always consistent with the construction order. Note that this consitency is due to the way a construction is carried out, and this easy approach must be replaced by a more sophisticated one in the case we want to allow redefinitions of objects (changing the algorithm of an object). In that case we must take care of finding the correct linear order of a partially ordered set, and we have to avoid the introduction of circular references.

Whenever an element is moved, we just have to iterate through the list of dependent algorithms and update each of the algorithms. See Fig. 9.3

9.1.4 Model-View-Controller implementation

In the last sections we presented the mathematical kernel of *Cinderella*. In order to separate the mathematical representation and the screen output we organized the visual representation in another way than usual. The crucial concept uses the Model-View-Controller concept, where the kernel presented so far represents the model, the viewports correspond to the views, and the controller is given by the different modes of *Cinderella* (add a line,

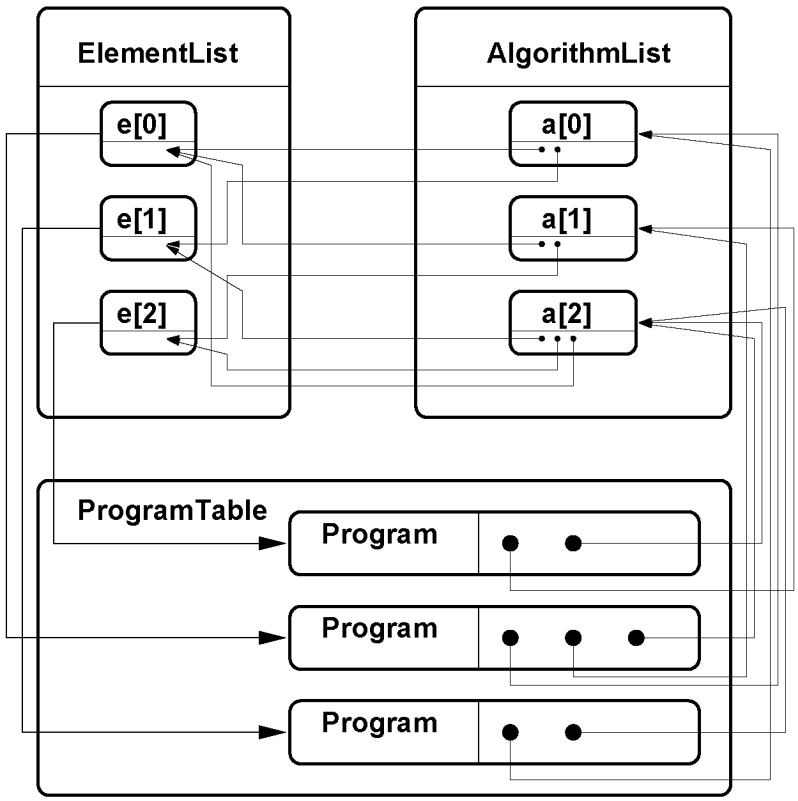


Figure 9.3: The list of elements and the list of algorithms are connected using a list of programs. The program of an element is traversed if the element is moved, causing all algorithms of dependent elements to be recalculated, and thus the update of all elements that must change.

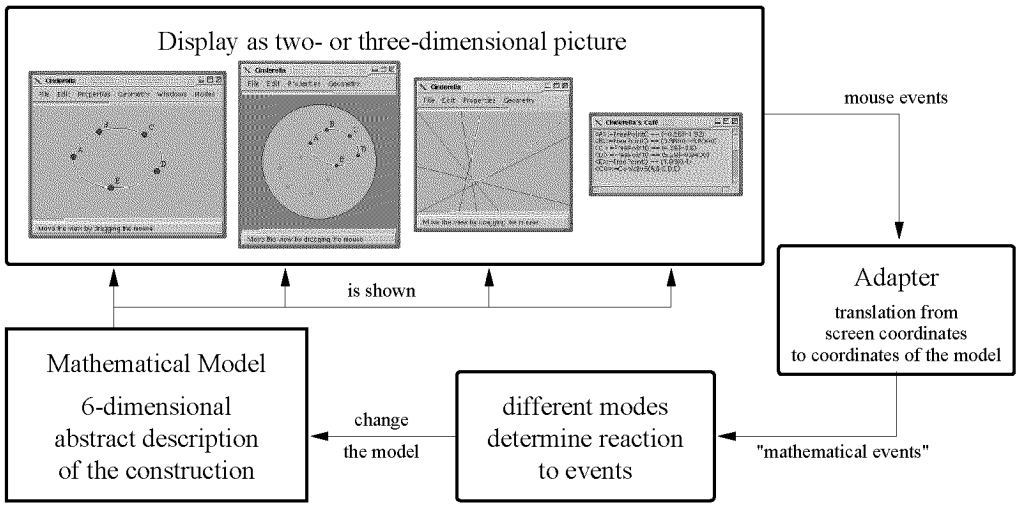


Figure 9.4: The Model-View-Controller structure of *Cinderella*.

move a point, etc.), see Fig. 9.4.

The mathematical objects in the kernel have no information on how to be displayed on the computer screen. Each viewport has a factory method that is able to create visual representations for the PG-Objects. Whenever a kernel object is created, every viewport is asked to create a suitable visual representation. These shadow objects are maintained and stored by the viewport, the kernel objects themselves do not know about their represenations.

After a kernel update every viewport is informed about that the construction has changed, and it must update its viewport elements. The kernel updates are a reaction to mouse events, which are interpreted using the current mode of *Cinderella*. In the move mode, a mouse drag causes a move of an object, in the "add a line"-mode up to three new elements are added to the kernel. The mouse events are coming from the different viewports, which figure out what elements are affected by mouse actions and at what position. This is the reason why the viewport elements must know the corresponding kernel elements.

9.2 Implementing Complex Tracing

We will now come to the heart of *Cinderella*s mathematical kernel, the part that implements complex tracing. Here is the basic tracing algorithm:

Algorithm 9.1 (Tracing Algorithm)
Input: A construction C, an instance at starting position A, and an end position B.
Output: An instance at B that is likely to be reachable by a continuous motion from A to B.

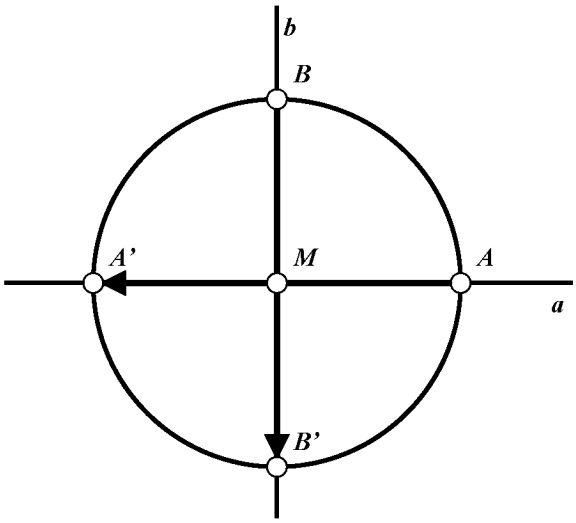


Figure 9.5: Here is a situation where the near-to decision will fail badly without a forced intermediate step into complex space. If we move the point A on the line directly to the position A' the near-to decision will leave B, the point of intersection of the circle and line b at its place. Instead it should move to B', since we would otherwise have constructed an absolute value function. You can imagine that the intermediate complex step creates a "turn" into complex space where both intersections are complex conjugates, and this forces additional intermediate steps.

1. Do a reparameterization of the construction such that a complex parameter λ creates the desired move while letting λ move from 0 to 1.
2. Set the initial position on the path γ: $t \mapsto (1 + e^{i\pi(1-t)})$ to $t = 0$. Set the initial stepwidth to $s = 0.5$.
3. Assume that the construction is in a valid position for $\lambda = 0$.
4. Add s to t and let $\lambda = \gamma(\max(t, 1))$.
5. Update the construction at λ and try to assign the new elements to the old elements of the last valid position based on nearness decisions.
6. If the assignment is possible, multiply the stepwidth s with a speed factor $\alpha \geq 1$, and go to 3. If $t = 1$, the algorithm terminates.
7. If the assignment fails, multiply the stepwidth s with a slowdown factor $\beta < 1$ and go to 4. If the stepwidth s is less than a fail value $\varepsilon > 0$ then the algorithms terminates.

The values for α and β must be chosen with care. Unfortunately, there is no theoretical result yet that tells us which values are best.

You see that we will always make at least one complex intermediate step. This is a heuristic that eliminates most of the wrong assignment decisions that occur in practice. Fig. 9.5 shows a typical example. A step into complex space will often, especially for complex conjugates, create the necessary intermediate steps to make the right decision.

9.2.1 Parametrization

How did we implement the reparameterization of the elements? We may have to create many intermediate steps for one mouse move, and we would like to have a as generic as

possible tracing engine.

In *Cinderella* we implemented the following: Whenever an element (this could be a point or a line, or even the radius of a circle) is moved from position A to a new position B, we will tell its algorithm object that a new position is requested. Every "movable" algorithm (free points, lines through a point, points bound to a line or circle) has a `initMove` method that is called with the new position as parameter.

After that initialization, subsequent calls to the `goTo` method passing a complex number will move the movable element to the abstract position $(1-\lambda)A+\lambda B$. Here *abstract* means that the new position of the moving element needs not to be exactly the linear combination as written above. The `goto` mechanism enables any algorithm to remap the complex path to another, better suited path than a simple linear interpolation. There are several movable algorithms that make use of that, for example all algorithms that allow underconstrained movements.

An example for an algorithm that is underconstrained is "Point On Line." Here the initialization and `goTo`-methods must take care that all intermediate elements on the path really are on the line the point is bound to. In particular, the end point must be projected to the line. Underconstrained elements are very hard to handle, the available degree of freedom must be used in a reasonable way, but in the end all ways are arbitrary. We refer to [87], where one way of handling these situations is discussed in detail, but we cannot claim that this way is superior to other design decisions. In the end it would be desirable to leave the choice to the user, which strategy or behavior he prefers.

9.2.2 Tracing

The assignments done in step 5 are based on nearness decisions. Let us illustrate the case where we have to assign two new points B_1 and B_2 to two old points A_1 and A_2. We will measure all pairwise distances, as in Fig. 9.6.

Then we will decide whether we can make a decision at all: If the two new points or the two old points are very close to each other compared to the sum of their distances to the old points, we will request an additional intermediate step. Else we will look at the difference between the direct path and the detoured path that create a triangle (in Fig. 9.6 illustrated by a gray triangle, the direct path is the diagonal one). If this is sufficiently high for both assignments, we will return the "shorter" one, else we will also request an intermediate step. To make this work, the distance measurements must obey the triangle inequality (what they should do anyway).

The performance of the algorithm is highly dependent on the choice of a numerically stable distance measure. So far we have not found a provably good one, but we rely on some kind of complex projective distance. In order to increase the numerical stability we tried to choose different measures based on the position in projective space. A problem that appeared when we had this flexible measurements was that the mixed distances of two point pairs were calculated using different measurements, which lead to wrong assignments or even a complete failure of the algorithm. If you try to use an approach like

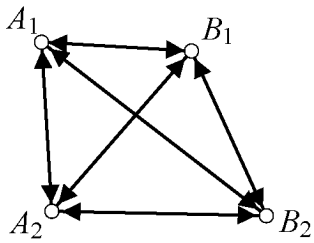


Figure 9.6: Two old points A_1 and A_2 shall be assigned to B_1 and B_2. This two-dimensional picture is confusingly easy to read, the original situation is taking place in $\mathbb{C}P^2$ and thus much more *complex*.

that, you must take care not to change the measurements during a near-to decision phase.

Another remark: We will do the assignments one by one by iterating through the program list associated with the moving element. If any assignment fails, we will immediately stop the execution of this program, since we cannot be sure that the assignments we find later are still valid when we do the intermediate step that is requested.

9.2.3 Decision making

The careful reader will have noticed that we are talking about decisions that assign new points to old points, while the analytic continuations of Ch. 7 work on complex numbers. This works because we know that the analytic continuations will give exactly the same results. But going up to the model of points, lines and conics, i.e. vectors and matrices, we can improve the numerical stability since we do not require all algorithms in a construction to use the same formula all the time. Any algorithm may choose under which conditions it will work best, and return just any representation of its results, that is, any scalar multiple of their homogeneous coordinates.

For example, the conic/conic-intersection can be calculated using any two of the three degenerate conics, or one of the degenerate conics and one of the original conics, just whatever is more stable for a certain situation. Or, the split operation can choose any of the three columns to return, just to make the maximal number of significant bits available. Or, even a simple algorithm that calculates the join of two points can scale/normalize its result to avoid numeric overflows.

Based on our knowledge that the original analytic continuation process will give the continuous motion we are looking for, we use the homogeneous objects for our decisions, being sure that this will give the same results.

9.2.4 Avoiding tracing

Not every algorithm (or primitive operation) is ambiguous, in fact, most of them are not. Whenever we have a part of the construction that need not to be traced, we want to avoid to include it in the recalculation of the intermediate steps. For this reason we split the program list (Fig. 9.2 in Sec. 9.1.2) into two, a list of all dependent elements and a list of the elements that have to be recalculated for the intermediate tracing steps. Note that these

are not only the ambiguous algorithms, but also all algorithms on which the dependent algorithms depend.

This split is even more valuable when we avoid tracing for elements where we have a kind of test function or certificate, see Sec. 9.3.1 below.

9.2.5 Backups and Singularities

Before we explain what happens when we hit a singularity other than that the tracing algorithm fails, we have to introduce a helper mechanism of the tracing kernel.

The tracing algorithm requires that we are able to recalculate a construction sequence for different input parameters quickly, and we must be able to store a valid construction sequence for later reference. For this we introduced a data structure `Register` (registry) that supports saving and retrieving the whole calculation into or from a numbered slot. Since we cannot use memory dumps in Java as we could do in C or C++, we require that everything that needs to be saved implements an `Assignable` interface and is registered with the `Register` class. The `Assignable` interface guarantees the implementation of an assign method, that can be used to copy the value of an instance onto another instance.

When an object is registered with the registry, the registry will create a number of objects of the same class and append them to the different slot arrays. These will be used as storage for the value of the original object, which is also stored by reference. A call to the `store` method of the registry will copy all original, registered assignables to the slot array, a call to the `retrieve` method wil copy the stored values back to the original objects.

Now we are able to work around singularities using the following strategy:

- If we start on a singularity, that is, if we are in a situation where certain pairs or quadruples of points cannot be properly distinguished, we will just don't care. If these points will move apart later, we will start to care by raising a flag.

- If we hit a singularity directly (for example, by moving a line intersecting a circle into the tangent situation), the tracing algorithm will fail. In that case, we will just jump to the final position $\lambda = 1$ and assign the points arbitrarily. This does not cause a problem, since the points we want to distinguish have the same coordinates, thus all decisions lead to the same result.

  The last valid position where we could distinguish the offending points is stored in a slot of the registry. If we move away from that singularity again, we will *not* start from there, but *we will instead start from that backup position.*

- If we hit a singularity on our path, but not where we wanted to go to, we can either restart at $\lambda = 0$ and choose another path (for example by scaling the imaginary part of it), or we proceed as in the direct hit case, with the problem of the chance of making the wrong decisions for $\lambda = 1$ (since we will go the backup position when we resume moving, we might not care for this single position).

We want to remark that the approach of *Cinderella* works fairly good in practice, but can be fooled. For example, *Cinderella* does not store a backup position for all elements that can be moved. Imagine you have a construction of several lines, circles and their intersections, and you move some lines into degenerate positions. Then you move them out of the singularities again in a completely arbitrary order. How could you decide which backup is the right one? *Cinderella* always resets its backup position to the current position in case we change the element that moves. This can causes unpredictable behavior, but it always occurs after stopping in degenerate situations. As an example you can try to move a triangle showing the angular bisector theorem into a highly degenerate position, best would be to move all vertices onto one point using the "snap to grid"-mode. If you now move some other element, a fourth point for example, and then make the triangle non-singular again, it can happen that you changed to an instance where the theorem is destroyed.

9.3 Automatic Theorem Checking

The automatic theorem checker of *Cinderella* is a very useful tool, not only for people who use it while they work, but also internally. It works by creating many new instances of a construction and evaluating a conjecture for all of these instances. If a counter-example is found, the conjecture is rejected, otherwise it is accepted as a theorem.

The automatic theorem checking proposed in Ch. 7 cannot be used directly – we would have to evaluate an infinite number of examples first. But we can mix the randomized theorem proving methods of Sec. 5.3 and the theoretical results for a simple proving heuristic: Move every free element of a construction into a random direction (using a special version of the `initMove`-method, `initRandomMove`), and repeat this several times to create many instances. If we do not find a counter-example within the instances, we assume that we have found a theorem.

Although we are still lacking theorems that support this strategy theoretically, we have found that the method works very well in practise. It is possible to make it fail on purpose, but for every day use it is fine.

9.3.1 Using Automatic Theorem Checking for Clean Data Structures

One internal use of the general automatic theorem checker in *Cinderella* is to help keeping the data structures clean. Whenever an element is added to the kernel, the theorem checker checks whether it is already known. It does not check whether the same object reference or an element with the same definition is contained in the construction data, but it "proves" whether one of elements currently in the construction is always at the same place as the new element. If that's the case, it will not insert the new element, but re-use the old one.

This is important for the ease-of-use of *Cinderella*. If you draw a line ℓ connecting two points A and B, put a point C on that line, and construct the line m connecting A and C, then

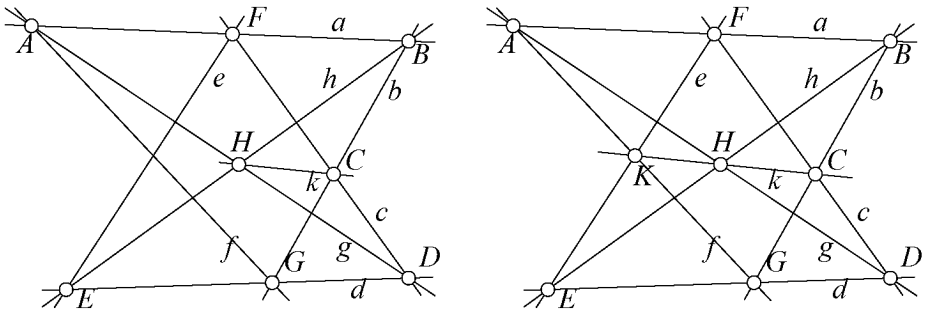


Figure 9.7: The theorem checker proves Pappos' theorem on the fly. We can see this by observing that after the last intersection point $K = \mathsf{Meet}(e, f)$ has been defined the line k is extended to include it.

ℓ and m will always have the same coordinates. Imagine for a moment that ℓ and m are represented by different elements. Now whenever you want to pick ℓ with the mouse, you will also pick m. This will confuse all operations that work on lines, because you will put an extra line into the input accidently whenever you choose ℓ as an input element. Also, the display of the line will be weird, because of rounding problems in the line drawing subroutines (provided by the operating system). Many other geometry softwares do at least check for double definitions (you cannot insert the line through A and B twice), but *Cinderella* is the only program that can handle identities caused by geometric theorems. *Cinderella* will insert the line m, prove that it is equal to ℓ, and remove m again.

The same technique is used for a separation strategy that can help to avoid the need of complex tracing. Whenever an intersection of a conic and a line or of two Euclidean circles is added to the kernel, it will be checked whether the other intersection – the method does not work very well for conic/conic-intersections – is already known. If so, then the defining algorithm of the point will be changed from "intersection conic/line" resp. "intersection circle/circle" to "other intersection of conic/line/point" resp. "other intersection conic/line/point." Since the "other intersection" algorithms, which calculate both intersections and return the one that is not equal to the point, are determined, they do not have to be traced.

This kind of shortcut can be applied very often in typical constructions. An example for which this strategy is optimal is a chain of circles of the same radius as in Fig. 2.9 on p. 28.

Another information maintained by *Cinderella* are point/line-incidence relations. These are needed for the line clipping mechanism, which can clip lines to its (Euclidean) endpoints. To make this work, we have to know which points lie on lines, and this information is created generically by the automatic theorem checker. See Fig. 9.7 for a non-trivial example.

9.3.2 Using Automatic Theorem Checking for Exercises

We can use the automatic theorem checking engine for a very exciting new feature of Dynamic Geometry software, the creation of interactive construction exercises. Students can work with a subset of the geometry tools of *Cinderella* inside a web browser, and the computer verifies the solution – which is not a fixed construction, but only some elements that should be construct in any way – or gives context sensitive hints. See Sec. 11.3 for some information about the use of automatic exercises in education. Here we will explain how the automatic exercises are realized in *Cinderella*.

A necessary ingredient for any automatic exercise is a construction for its solution. This is only fair, the one who creates the exercise has to know how it can be solved. But the real reason for this is that *Cinderella* uses the elements of the given solution as "certificates" to find out whether the solution has been found, or whether the elements corresponding to a certain hint have been constructed.

When a student starts up an interactive exercise sheet, he will see all "starting elements" of the solution construction that was provided, say, points A and B, and he will be asked to solve the exercise, let us take "Construct the midpoint of A and B." What he does not see is that the whole solution is already present in the mathematical kernel of *Cinderella*. The software has just bypassed the creation of the corresponding viewport elements (see Sec. 9.1.4) for all non-start elements.

Now while the student adds more and more elements to the construction, for every new element the automatic theorem checker will check whether this element is already present in the kernel, as usual, and when it is found, it will reuse this old element and, if necessary, report it to the viewports to make it appear. This has the effect that valid constructions for the already present elements will be detected on the fly. We can check for all elements that are reported to the kernel whether a hint or even a solution is associated with them, since exactly the elements that were already present in the example solution will be reported.

Another effect of having the construction already in the kernel is that we can easily track the current position of a solution element if the student changes some of the input parameters (the point A or B in our example).

This is enough for a basic exercise checking engine. In Fig. 9.8 you can see an example where the theorem checker accepts an alternate solution to the midpoint problem. The original solution was the standard construction of the bisector a using intersections C and D, and finding the intersection E of a and the line b connecting A and B. The two circles that are necessary for this particular solution have also been used by the student, but then he proceeded by adding two more circles, the intersections F, G, H and K, and he finished his solution by intersecting the lines c and d that connect F and G resp. H and K. This point of intersection has been identified by the theorem checker with E, and since E was marked as solution element during the design of the exercise, the construction is accepted as a solution.

This technique will not work whenever we have underconstrained elements, like in the angular bisector construction (Fig. 6.4 on page 83), or the point on Thales' circle in a

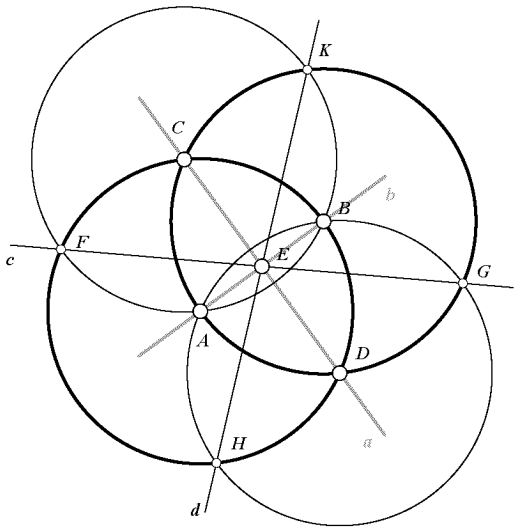


Figure 9.8: The internal construction is used to check the solution of the student. The two circles around A and B have been reused by the student, but then he left the standard construction for an alternative one. After he arrives at his solution the intersection of c and d is identified as being always equal to the solution element E. The lines a and b are not visible, but only used inside the kernel.

construction of an right angled triangle, or even a free point, for example when you want to construct the trisection of a segment. The kernel will usually not prove that a point A on a line a is equal to another point B on that line, because then we could bind at most one point to a given line.

In *Cinderella*'s theorem checker we introduced a method we called "guessing," which is disabled by default, and enabled while an exercise is solved. In that case, also "definition equality" will be accepted, which means that an underconstrained or free element that is present in the kernel, but not in the viewports, is reported as equal to another element that is added to the kernel, if the definitions of these two are equal, i.e. both are a free point, both are a point on the same line/circle, both are a circle centered at the same point, etc.

This approach works fine as long as the guessing cannot be wrong, for example, if all free and underconstrained elements are uniquely defined, i.e. there are no two free points, no two points bound to the same line, and so on. Otherwise *Cinderella* will probably not recognize a part of the construction properly if it is done in another order than in the original construction. Anyway, as soon as the elements for a hint are unique again, *Cinderella* will "synchronize" again. Also the checking of solutions which are unique will never be affected, which was the final reason why we included this feature, although it will not be guaranteed to work for ill-formulated exercises.

9.3.3 A Killer Example

The theorem checker is not perfect. On the one hand we do not have a theorem that guarantees that randomized examples are enough, although we are sure that there must be a way to extend the randomized techniques that work for polynomials also to the special analytic functions we deal with.

On the other hand we have a problem that is much worse, the numerical instability of the whole theorem checking process. Even if we can create many examples (which we can do now only by moving to them, introducing all problems of the tracing algorithm, see also Sec. 7.5), we do not have a method to check whether a certain value we calculated is zero or not. In fact, we are dealing with expressions built from additions, multiplications, square roots and third roots, and these are hard to handle, both symbolically and numerically, see [55, 7].

Here is an example which will crash the theorem checker of *Cinderella*. It is somewhat artificial, and you cannot draw it properly on a sheet of paper, which is the reason why we did not include a figure.

Example 9.2 (Killer Example) *Draw a Euclidean circle of radius* 1 *around the origin. Put a point A on the x-axis. Using von-Staudt Multiplication (see 5.5) we can construct a point* A^2 *on the x-axis whose x-coordinate is the square of the x-coordinate of A. Repeat this to get points* A^4, A^{16}, A^{256}, A^{65536}.

With a circular inversion you can get a point B with an x-coordinate that is the inverse of the x-coordinate of A^{65536}. *This value is very small for almost all positions of A, and the theorem checker will prove that B is equal to the origin.*

If you try this at home, you will see that it is hard to do the construction at all, but if you succeed, you will see the theorem checker fail.

9.4 Self-Exploring Loci

As a last implementation detail we want to explain the locus heuristics of *Cinderella*. It is not at all clear what we want to see when we ask for the locus, the trace of a point while a construction is moved under certain constraints. Since we are not working with a determined, but with a continuous system, we cannot just ask for *the* position of the dependent point for all positions of the moving point.

One possible alternative would be all positions of the dependent point we can reach by continuous moves. But this could lead to disconnected curves (disconnected in real space, of course not in complex space), which is highly undesirable. So we ended up with the following goal for *Cinderella*: We want to get the curve of all positions we can get by real continuous moves, continuous moves where the dependent point has real valued coordinates.

We get this curve by moving along the "road," the line or the circle where the moving point should reside on. Whenever we run into complex space, we immediately change the direction in which we traverse the road. Whenever we come back to the starting position and if we move in the same direction as in the beginning, we check whether we have the same instance as when we started. We are done then, else we have to continue our walk.

Because we turn around whenever we run into complex space but do not change the detour path we take around a singularity, we will circle around a singularity on the Rie-

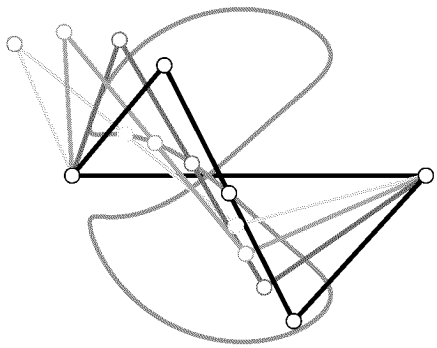


Figure 9.9: The locus of the center point of the upper bar in a four-bar-linkage (the lower bar is fixed). *Cinderella* shows the complete locus that would be created by a real mechanical linkage of the same dimensions. All determined systems can only show half of the locus.

mann surface of the dependent point: it just became complex, so the discriminant of some root must have switched from positive to negative and back again. Since we always move on the half-circular path that is on the left of the oriented segment connecting start and end, we will make a full turn around the singularity. This has the – desired! – effect of changing into another real component of the locus, we will explore another instance of the construction at the input points we just came from. See also Fig. 7.3 and try to follow the path of the two intersections using first $\mu = 1$ and then $\mu = -1$, and watch how the two points of intersection are interchanged.

In practice this gives a very natural way of drawing loci. The dependent points do not make sharp turns, they move differentiable. The loci show the "physical behavior" of the dependent points, for example in the case of the 4-bar-linkage shown in Fig. 9.9.

The speed while moving a point on a line is another important issue when we generate loci. It is not feasible to use a constant speed, because in that case we would never be able to walk along the whole line in finite time (see Fig. 1.1 and 2.2.6 for an example locus). *Cinderella* is built on a concept that uses speed-ups and slow-downs like we already did in the tracing algorithm 9.1. Although this concept works acceptable for many loci, it still fails sometimes for complicated curves. We do not want to miss details in a locus, but we also have to be able to rush through the parts that are not on screen or which are only interesting at a higher screen resolution. It remains an open problem to find an efficient algorithm for better locus generation.

Chapter 10

Creativity in Math Education

Dynamic Geometry Systems (DGS) are very powerful tools in math education, and some research has been done regarding the use of such systems in teaching (starting points could be [24, 51, 30, 19, 3]).

A particular interesting question is whether intellectual creativity can be stimulated by such tools. Here we will not cover all aspects of creativity in math education, but we want to investigate some implications for the software.

We want to emphasize that it is not important for one software to be able to cover all educational aspects. It should be in the responsibility of the teacher to choose the best suited tool from case to case. If, for example, axiomatic proofs are of importance, the teacher should choose a software like GEOLOG [27], that supports these with the built-in prolog inference engine.

However, a certain mathematical standard should be obeyed by any DGS. It will be one of the forthcoming tasks to specify a quality ensurance standard for Dynamic Geometry Software [38].

10.1 The need for mathematical consistency

Try to explain why your favorite Word processor crashes if you have a manuscript of more than 50 pages with some figures in it. You will not be able to do it other than by referring to implementation bugs. The good thing is that you know that something wrong has happened.

Try to explain why your favorite Word processor changes certain spellings. That's easy, it just corrects the mistakes you made. But how do you know that the computer is right and you are wrong? In some cases it is just confidence. You believe that the software manufacturer did the right thing, and the software will not fail.

We meet a similar situation when students use a computer algebra or Dynamic Geometry system. They assume that the computer is right, and what they see on screen is a proper picture of (abstract) geometry. As pointed out in Ch. 6 this is not the case for most

Dynamic Geometry Systems. This leads to the problem that the students do not learn what you want to teach them, e.g. Euclidean Geometry, but some other kind of geometry.

In [53] this is taken as a matter of fact, and Jean-Marie Laborde tries to create a consciousness for this. He explains that the addition of dynamics to Euclidean Geometry leads to another, very different type of geometry, called "Dynamic Geometry," and the implementation (e.g. in Cabri Géomètre II) is a third kind of geometry ("Cabri Geometry"). All three lie in "generic position," there is no common line joining these three.

The mathematical part of this thesis shows that we cannot agree with this point of view. Clearly, Dynamic Geometry is a much richer concept than Euclidean, projective, Hyperbolic, ... Geometry, that exceeds a simple parametrization of static geometry. But there are certain mathematical implications that prescribe the behavior of *any* implementation. Every software should try to come as close as possible to this "ideal" world of Dynamic Geometry. Our strong believe is that we can reach that goal within the next ten years. Two ingredients are needed: We need more research that stabilizes the method of complex tracing numerically (see also Sec. 9.2) and exploits ways to minimize the computational power needed, and we need faster computers that can do all the work.

Meanwhile teachers should ask for the best possible approximation of Dynamic Geometry, because anything else will confuse the student at some point. It is fun to watch and analyze the artefacts that are created by Dynamic Geometry Systems, but it also needs a lot of experience and knowledge of geometry to deal with them – exactly what we cannot expect from the users which are people who are new to geometry.

Here is an example, due to Laborde [53]: A student who tries to construct a reflection of a segment at a line using circles and varies the construction later might be confronted with a jumping point situation where the reflection coincedes with the pre-image or, even worse, only one end-point is on the wrong side (see Fig. 6.8 on page 88). This is very discouraging for the student and thereby questions the effective use of DGS in teaching. Cabri can solve this particular problem using a seperation technique similar to the "other intersection"-technique of *Cinderella* (see Ch. 9.2.4).

Another example that is not covered by this separation technique are the iterated angular bisectors as presented in Ch. 6.2.2, see Fig. 6.7. A clever student who just learned how to bisect an angle might be tempted to try the quadsection of an angle using two bisections. How can you explain him that his approach is good and correct, when the computer disproves it for some cases? How should he learn that this is a construction that is always correct, if he can see that it is not correct at least half the time?

10.2 The need for modularity

What can we do with a mathematically consistent system? Everything, and that is the point. It is possible to manipulate a construction, to move it around wildly, to draw surprising loci, and the visual feedback you get represents mathematics. What we eliminate by doing the right mathematics are the artificial borders to exploration.

We claim that it is absolutely necessary for a geometry software to have a strong mathematical background in order to enable the students to explore geometry.

It could be questioned whether the strict mathematical behavior of *Cinderella* is always good for geometry education. Take the midpoint construction as an example: While it is nice that *Cinderella* can work with complex intersections (see Ex. 6.12), it is probably confusing for 7th-grade students. We would rather prefer that they find a construction where the intersections cannot vanish.

The reason why we still think that the mathematical approach to Dynamic Geometry software is the right one (instead of a purely educational approach), is that it is easily possible to downgrade a mathematically correct software for special situations, but it is hard or impossible to upgrade a software that can only handle "low-level" geometry. While it is easy to "switch off" complex elements altogether in *Cinderella*, it is not easy at all to switch them on in other software. While it is easy to configure *Cinderella* to allow jumping elements, it is impossible to configure other software such hat it behaves continuously.

Also, it is always possible to remove certain functionality from the software. For the school edition of *Cinderella* we created a "normal" and a "professional" version of *Cinderella*, where the professional version is identical to the full-fledged university edition, and the "normal" version just lacks the support for other viewports than the Euclidean one, the support for other geometries, and several operations, including those for conics.

10.3 Raising creativity by restriction

The highly modularized approach of *Cinderella* is the basis for one approach to creativity: Creativity that is caused by restriction. The difficulty of the same task can range from impossible to easy for different sets of available tools. The McGyver-kind of student may be able to find the midpoint of a segments using only a compass, while most students will use compass *and* ruler.

This method of raising creativity by restricting the set of tools for an assignment can either be forced by exporting only a certain subsets of tools in an interactive exercise, or it can be suggested only and it will be backed by the automatic theorem checker that does not prescribe the exact construction sequence for a solution. So *Cinderella* leaves the opportunity for the teacher to encourage students to find a more challenging way to solve an exercise without forcing them to do so and without having to know this solutions themselves.

10.4 Exploring Geometry with Loci

Instead of restricting a student to raise creativity, we can also give him additional power in order to encourage him to look for interesting constructions.

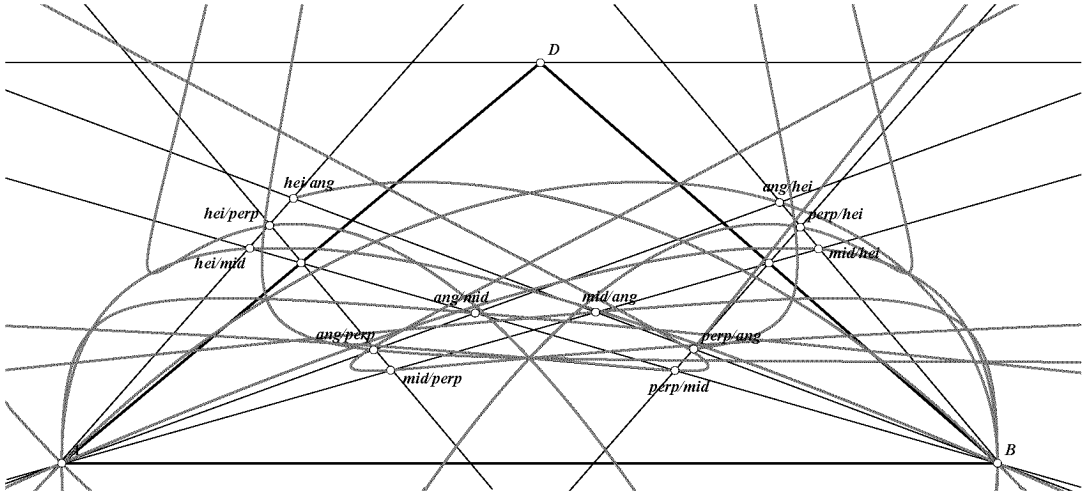


Figure 10.1: All loci generated by mixed intersections of the angular bisectors (ang), perpendicular bisectors (perp), edge bisectors (mid), and heights (hei).

A student who is looking at the locus of the intersection of the perpendicular bisectors of a triangle while moving one point on the line parallel to the non-adjacent edge will see that this moves on a line – of course, it moves on the fixed bisector due to the perpendicular bisector theorem. So what is going on if he breaks this dependency just to find out what happens? Fig. 10.1 shows all possible combinations in a single overcrowded picture. See the web site of Weth [84] for a discussion of this scenario from a purely educational point of view (using *Cinderella*).

Weth covers these rather new uses of Dynamic Geometry software in great detail [86, 85] using other geometry software, and we would like to cite his first reaction to *Cinderella*: "I just did some experiments with 'loci constructions.' Fantastic. *This is a new quality of interactive geometry software.*" This strongly indicates that our approach of concentrating on the mathematics and not on the particular requests of education, our stubborn attitude to make a construction tool only available if it is completely understood and correctly implemented, is nevertheless useful for education. We do not want to teach "geometry software," but geometry.

At this point – instead of only repeating the research of Weth – we just want to suggest another example where we can use loci in explorative education. Recall the mechanical linkage of Fig. 9.9. The four-bar-linkage itself has a lot of potential with respect to variation. Depending on the lengths of the bars several very different curves are possible, see Fig. 10.2 . Even more variation is possible if you allow to add another (or more) link to the construction. A challenging contest in class could be to get the wildest curve possible with a certain number of sticks.

It should be studied whether this use of Dynamic Geometry software can really help to stimulate the students' creativity, and we hope for first results in December 1999, when a comparative study [22] will be done at various schools in Germany. Although the main

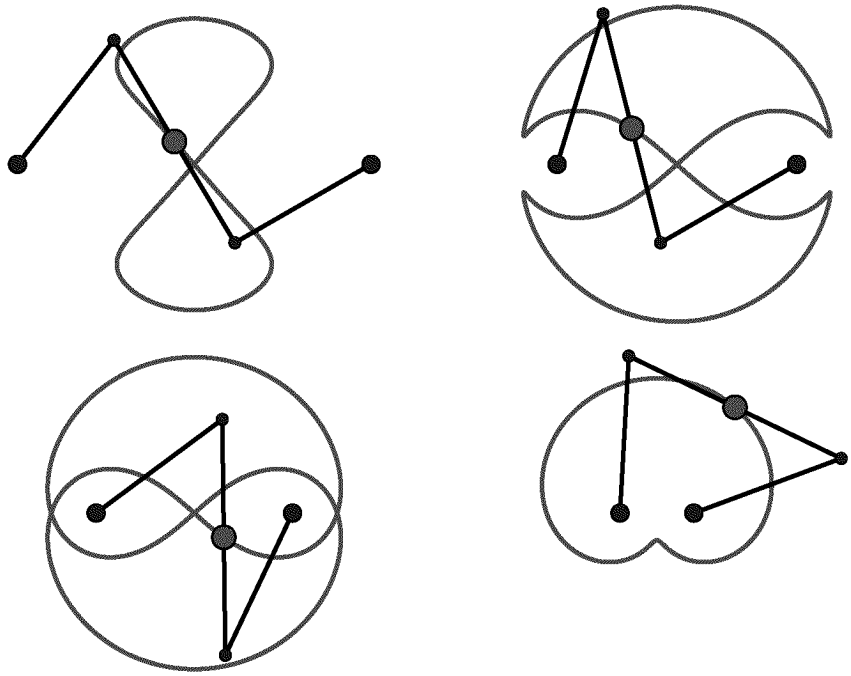


Figure 10.2: These different curves are the loci of the center point of the middle bar of the 4-bar-linkage. Only the distance between the two fixed points was changed to create these variations.

focus will lie on the impact of interactive exercises, there will be an approach to teach distances to children in 7th grade using loci of mechanical linkages.

Chapter 11

Geometry Education and the Internet

The world wide web has started a revolution. Ten years ago wide area networks were only used by big companies, and they were hard to maintain, expensive and – boring. This has changed radically. Anybody who has a computer can join the Internet today, at a low charge. It is easy, cheap and exciting.

After the first excitement has gone, it does not look as wonderful as it did. The Internet is very good at distributing information, but it does not create the content by itself, its the users who have to do it.

It all boils down to having a fairly good organized, or at least browsable and searchable, collection of flashing news, sports information, weather data, travel guides and things you never wanted to know about persons you never met. What's missing is content that really uses the fact that it is presented on a computer.

11.1 Creating Web content

Of course, this is a little bit exaggerated. What I really want to point out is that we need software that makes it easy for everybody to create real, interactive content for the Internet.

When I look up a geometric theorem on the web, I want more than some pages that I can print out. I want to experience it, use it, change it, compare it. I want to *interact* with it.

I am not talking about one all-purpose software. Instead, every software which is used to work on a computer must provide a way to export the current work to the internet. Then we can hope for accessability of interactive content in all areas.

11.2 Easy Creation of Interactive Web Pages

Following the claims of the preciding paragraphs, *Cinderella* offers an easy way to export geometric constructions. Since *Cinderella* is written in Java, it was easy to reuse the

application code for a runtime applet version that builds on the same mathematical kernel and the same display routines as the standalone version (see Sec. 8.2.1 for the technical details).

A very important constraint while designing *Cinderella* was that it must be very easy to create web pages, because we cannot expect that the broad audience we want to address knows anything about HTML. Creating web pages must be as easy as saving a construction to hard disk.

We met this goal at least partially: If you want to create a web page containing an interactive version – you can move free elements – of a construction, you can do this just by saving the construction and an automatically generated, very basic page description in HTML. To publish this on the internet two additional steps are needed. First you have to put a copy of the runtime library contained in the file `cindyrun.jar` into the directory which contains the web page, and then you must transfer the three files to a web server, using whatever method is provided by the Internet service provider.

If you want to export an animation to the web, its as easy as exporting an ordinary construction: The animation control dialog window that appears whenever you start an animation includes a web export button that now creates an HTML file that starts the animation using the current parameters. The runtime library is the same as above.

This easy creation of web pages makes it possible for everybody who uses the software to publish his work within seconds. This makes *Cinderella* different from other approaches like Cabri-Java [50] (which is close in ease of use, you can load files of the original standalone version of Cabri with the applet, given that they do not use objects or tools that are not present in the applet version) or Java Sketchpad (JSP [37], the applet version of Geometer's Sketchpad, which requires an additional conversion step from standalone files). The complete strength of the standalone can be put on the web without requiring additional work.

11.3 Using Automatic Theorem Checking for Exercises

The ordinary web export of *Cinderella* is very useful and opens up many exciting possibilities due to the tight Internet integration. But we did not make special use of the mathematical power of the software yet. The combination of the Java implementation and the automatic theorem checking presented in 9.3.2 unleashes a new level of Internet integration of Dynamic Geometry software.

A student who shall use a (standalone) geometry software to teach himself geometry is probably completely clueless about what to do with the software. It is almost like handing out pencil, ruler, compass and paper to the students and expecting that they will discover theorems by trial and error.

After clicking around with the mouse a little bit most students will be frustrated and stop working with software. You can compensate for this by handing out printed exercise sheets that suggest starting points for geometric exploration, or even interactive web pages

which illustrate your suggestions dynamically. What is still missing is a kind of feedback for the constructions the student does with the software, and these usually need a direct teacher interaction. Teaching with the help of a computer thus introduces lots of extra work, and it is very difficult to support every student personally.

A much better help would be if the software could give helpful tips or hints, or comment on the solutions to construction exercises. *Cinderella* contains support for creating special versions of the software that run inside a web page and offer a subset of the original tools to solve a particular construction exercise, and the internal theorem checking engine is used to provide context sensitive hints and to check for the correctness of a constructed solution.

The design of such an interactive exercise involves more work than the ordinary web export, and it is a non-trivial task to create a good exercise. An exercise consist of an example construction that solves the exercise, a set of tools that may be used to complete the construction, the definition of the starting elements that are presented directly after the applet has been started, the definition of a task text that is shown to the student, the definition of one or more solutions, and the optional definition of context sensitive hints.

The example construction will be used by *Cinderella* to verify solutions and to identify parts of the construction that seem to be useful. See Sec. 9.3.2 for details on the technical details of element identification and correctness checking. It serves also as a sanity check for the exercise – if the teacher can solve the exercise it is likely that it is doable.

The tool set may contain a subset of *Cinderella*'s tools. The default tools are the electronic versions of ruler and compass, together with the tools for moving elements, undoing construction steps and restarting the exercise.

The start elements are chosen from the construction, and the chosen set is complemented by all elements that were necessary to construct the start elements. This is necessary to avoid invisible constraints on the start elements.

The solution must also be defined. Here the theorem checking of *Cinderella* comes into the game: Not the complete construction sequence is a solution, neither it is required that the elements we are looking for must share the definition with the solution that is presented by the student. It will be checked by *Cinderella* whether the student's solution will always – for any placement of the starting elements – be at the same place as the solution elements in the example construction. See Fig. 9.8 on p. 136 for an example where two different constructions lead to the same solution.

The hints that can be defined are actually very similar to solutions, except that they will not end an exercise, but are some kind of checkpoint. If *Cinderella* recognizes that a hint has been completed by the student, it can give motivating comments like "you are on the right track." If the student asks for a hint, the next uncompleted hint will be given, first in a textual form, then by automatic addition of the necessary construction elements. If there no more hints available, the solution will be presented in a likewise manner.

A current restriction in the hint processing of *Cinderella* is that the software is not able to differ between to alternate ways to the solution. Theoretically it is possible to recognize which way the student is taking to solve the exercise and to present the correct hints in

either case. The missing part here is the user interface to input this kind of hinting, but it will be supported in a future version of *Cinderella*. First we will have to watch the user's (exercise author's) feedback on this first interactive exercise tool ever.

Since the software has just become available there is no empirical evidence on the educational impact of interactive exercises. We hope to receive some data within the next year, a first academic study using *Cinderella* in school which is focused on the use of interactive exercises will be done in December 1999 by Heintz [22].

11.4 Distance Teaching

Distance teaching is one of the upcoming educational challenges. Here we want to discuss briefly some of the parts of distance teaching where software like *Cinderella* could be helpful.

11.4.1 Remote Views

The Model-View-Controller architecture of *Cinderella* (see Sec. 9.1.4) can provide a kind of geometry phone or remote geometry blackboard. Within a local network, for example a computer lab in school or university, or using wide area networks like the Internet a geometric construction can be used on two different machines at the same time. Just think of it of opening second Euclidean viewport, not on the same display, but on the computer next to you, or even thousands of miles away. Whatever you do in one view will be shown one the other display at the same time (not counting network delays). The synchronization can be quite fast even on low-bandwith networks, because the viewports are responsible for the display of objects themselves, and the information that must be transmitted for every frame of a construction is on the order of a few hundred bytes.

This scenario is hypothetical and is not as developed in practice as it could be, since we first concentrated on web export and the standalone version of *Cinderella*. But even today you can use *Cinderella* remotely using tools like the Remote AWT, supplied by IBM [34]. But we want to remark that the viewport approach of *Cinderella* is a key ingredient to the successful synchronous electronic communication of geometry.

11.4.2 Web-based Education

In web-based education an educational institution or a private content provider creates a web site that is dedicated to deliver material for self-teaching (not only in geometry, but any other discipline). This is backed by communication tools like chat rooms, news-groups/message boards, email, etc. The maintainer of the web site will probably support the acquisition of material by having online teachers answer questions and help with the material. Depending on the scope of the "virtual campus" degrees may be offered that require examinations that either are done online or in person.

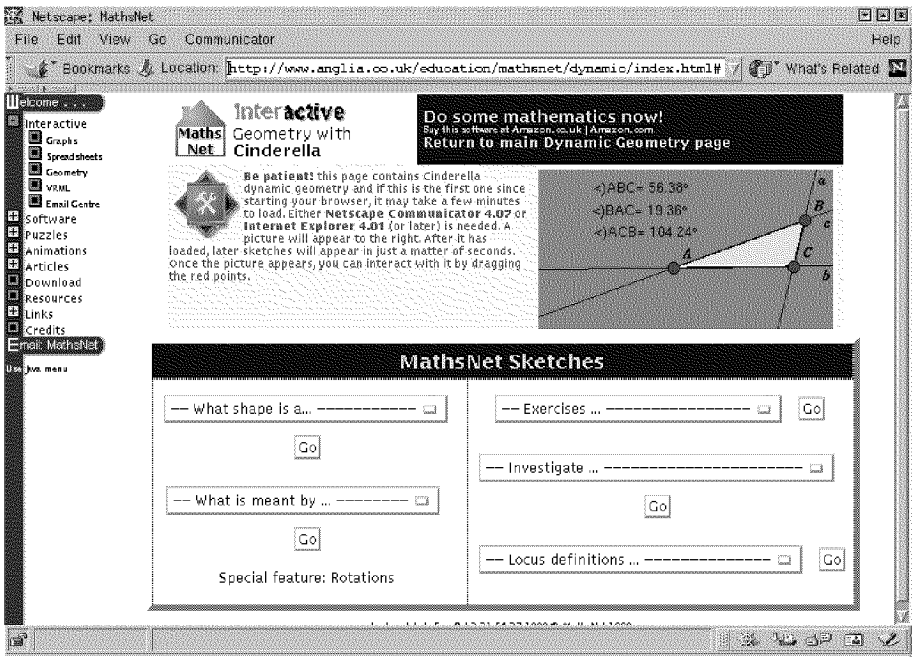


Figure 11.1: Starting point for Dynamic Geometry explorations on Mathsnet UK, the first educational website that used *Cinderella*.

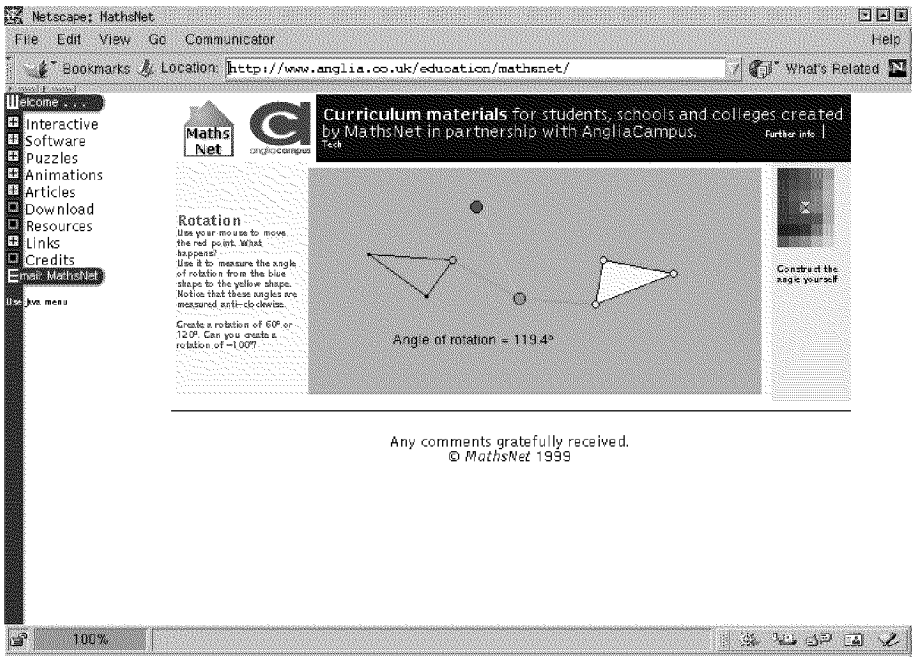


Figure 11.2: An Example of the material provided by Angliacampus [2]. Using the colored boxes on the right of the construction you can navigate the material. To the left and the right of your current position are similar pages, above are easier examples, below are harder ones.

Cinderella can help to increase the success of educational websites by providing a solution to present geometry and geometry-like figures interactively, thus adding another level of understanding. Using the interactive exercise features of *Cinderella* it will also be possible to do online examinations (currently there is no support for secure transmission of scores, but this could be added on demand).

We hope to receive more empirical information about the successful use of Dynamic Geometry software on educational web sites when AngliaCampus refurl:angliacampus, a UK-based subscription service to deliver educational material, will go online with *Cinderella* sketches. First impressions can be found on the Mathsnet UK website at [12], see Fig. 11.1 and Fig. 11.2.

We want to conclude that web based education is constantely gaining attention, and it is necessary to provide sophisticated, specialized authoring tools for content providers, otherwise the new possibilities of the Internet will not be exploited. It is not clear yet whether this form of teaching/learning will be "better" in any way than traditional approaches, but without the appropriate tools it will not be possible to use the new media to their full extent.

11.4.3 Communities

As the last topic in distance education we want to mention the concept of net communities. For everybody who has Internet access today it is possible to communicate with other, to find information on the net, and to publish information yourself at low or even no cost. These publishing possibilities were not available before, and they have started a revolution.

One problem about this unlimited publishing is, that it is almost impossible to *find* the information. Even search engines like AltaVista or Internet directories like Yahoo do not really help, since they will find much more web pages related to a topic than you will ever be able to review. Do the following experiment now: Try to find material about Pythagoras' theorem on the net. On Oct. 18, 1999, a simple search with AltaVista found around 1000 web pages containing information about Pythagoras' theorem; the example construction on the *Cinderella* website [47] was number 19.

A concept that could avoid the scattered and unbrowsable information on the net are *communities*. A web community is a place of exchange on the web, focused on a topic, open for everybody, like a market place. It provides space for user-contributed material, and it is the first stop whenever you are looking for something related to the special interest of the community.

We are currently planning to provide such a community exchange for *Cinderella* users, where it will be possible to collect constructions and exercises that are either specific to a certain age or grade, or to a special topic, like descriptive geometry or optics. We hope that this place will be accepted and filled with life by the people using *Cinderella*, and that within a few years there will be a complete collection of geometric constructions.

Chapter 12

Future developments

What's next in Dynamic geometry? In this chapter I want to present what I think are important questions that should be answered within the next year, both to complete the mathematical results of this thesis, and to make them work also in other contexts. I also want to point out some other questions in relation to Dynamic Geometry software and its application in education. Finally, I will give some examples where we should try to apply the methods found in this thesis.

12.1 Computation on Riemann Surfaces

The main contribution of this thesis is that it resurrected the existing theory for Dynamic Geometry and made it applicable. Computer science has neglected many of the old results in geometry that at first sight seemed to have no direct application or even a connection to questions in computational geometry. Many problems in Dynamic Geometry software could have been solved years ago, but apparantly the advances in the mathematical treatment of geometry did not find their way.

12.1.1 Complex Tracing

The tracing algorithm of Sec. 9.2 is the heart of *Cinderella*. It used to "walk" on the implicitly given Riemann surfaces of the construction.

In this thesis we did not give criteria for the numerical stability of the algorithm, and actually all of our results concerning the implementation of complex tracing are just empirical. The heuristics do work, and they seem to be reasonable, but there is no proof whatsoever that guarantees the correctness of the results or gives performance guarantees.

It is important for the further application of complex tracing also in other contexts that this gap is closed. The heuristics that work for *Cinderella* may fail in other places, and we do not even have a method to verify whether we get correct results.

A reasonable approach to answer the questions about finding the right stepwidth would be to use the methods of the homotopy method of Smale et al. (see [5]). The

homotopy method is used to find the roots of polynomial equations by tracing them along a path. Further inspection shows that the results of Smale are not directly applicable and the connection of homotopy methods to complex tracing is not as tight as it seems to be at the first look, but still it is promising to merge the tools used for the homotopy method into an analysis of complex tracing.

Even if we can find criteria that make complex tracing work reliably, we still have to work around the numerical problems that arise. The introduction of roots makes it difficult to work with fixed bitlength calculations.

12.1.2 Symbolic Methods

It could be that there is a better way to walk on Riemann surfaces. If we know, for instance, where the singularities of the analytic functions lie and where we have to flip decisions, we might have a chance to go directly, without intermediate steps, to the right instance of the construction. Or, perhaps we could find a symbolic representation of the associated Riemann surface that can be used for fast moving on it.

12.1.3 Parameterization

The parameterization we introduced in Sec. 9.2.1 was the key to the reduction from functions in several variables to functions in one complex variable. Without it we could not have used the results of complex analysis like the identity theorem for analytic functions. However, the continuity theorem for geometric theorems (Thm. 7.17) indicates that there must be a way to handle the situation also for the multivariate case. It is an important next step to find the proper theoretical setup for multivariate geometry.

12.1.4 Automatic Theorem Proving

The automatic theorem checking that is done by *Cinderella* is based on the strong feeling that it is unlikely to find many true instances of a Dynamic Geometry Statement without having a Dynamic Geometry Theorem. In the case of point/line-constructions we could use randomized proving for multivariate polynomials, and the methods there prove that we have the right feeling, it is unlikely to find a true instance of the statement unless its true everywhere.

In the general case we have to deal with special analytic functions, mainly constructed by addition, multiplication, and inverses of squares and cubes. It still seems to be possible to prove theorems by random instances, but we could not yet find theorems analogous to the Schwartz-Zippel Theorem 5.15.

Actually, the situation is even worse. We do not even have a way to create many random instances of a statement. We worked around this using a kind of random walks, but this is an extrace obstacle for true randomized theorem proving as opposed to the theorem checking presented in this thesis (however, we want to point out that we could

at least fix what we mean by a Dynamic Geometry Theorem, which was not clear before, and that this was a necessary step towards a theory of randomized theorem proving).

12.1.5 Complexity issues

One key ingredient of automatic theorem proving is the fast generation of random instances. As we said above, we have to resort to a kind of random walk on a Riemann surface. What we really would like to do is to jump to other instances. This means that we must be able to decide whether there is a path on the Riemann surface from the known instance to the generated one. Currently we do neither know an algorithm to decide whether to positions are connected, nor do we know the algorithmic complexity of this questions. It is still possible that one can do this efficiently.

12.1.6 Constraint based configurations

All geometric constructions we considered in this thesis really were *constructions*: Starting with elements that could be placed freely, we added one element at a time. The dependencies of the objects are partially ordered.

This leads to a certain inflexibility of the geometric configurations, we cannot pick just any element and move it, but we have to pick an element that has a degree of freedom. Sometimes this is undesirable, and actually there is a way to describe configurations without having to construct elements one by one. Using *constraints* that fix the relative positioning of objects we can break the orientation of dependency graph. The constraint based approach, which is used in software for computer aided construction, is much more flexible (see Rem. 5.38) for a geometric theorem that cannot be constructed, but described using constraints), but introduces also a lot of new problems. The most important question is, however, whether we can apply sort of the theory of complex tracing also to constraint based configurations.

12.2 Dynamic Geometry Software

After we could solve the continuity problem for Dynamic Geometry Software, we still have some questions that should be addressed in the near future.

12.2.1 Third Dimension

Many people ask for *Cinderella*-3D, a version of the software that can work with constructions in 3-space. In the three-dimensional setup we still can apply the theory of complex tracing, although it is a little bit more involved. The reason why we think that it is a challenging project that needs at least some years of work does not lie in the area of continuity problems.

The first hurdle to take is the design of an intuitive user interface for three dimensional dynamic geometry, that can be compared in both ease-of-use and expressional quality to Dynamic Geometry software in 2D. There is hope that this will be easy given the capabilities of todays and tomorrows computers.

A much bigger problem is the exploding complexity of geometric operations in 3-space. If we just want to be able to intersect linear and quadratic objects, like we can do in 2D, we will have an enourmous amount of special objects that occur as the intersection of quadrics. To create a versatile software for 3D will need much more manpower than the *Cinderella* project.

12.2.2 Macros

So far we did not address macros at all. A macro is a part of a construction that can be reused for other input elements, comparable to a subroutine in a computer program. *Cinderella* does not offer macros yet, and it is a feature that many people miss.

Macros are easy to add to a conservative geometry software, because we do not have to care at all about ambiguous decisions. In a continuous system it is much harder even to find out what a macro should do.

Here is, very briefly, a way to implement macros in a continuous system: Store a complete instance of the macro construction. If the user applies the macro to some other input, move the macro construction continuously from its instance to the new input parameters. Then you can copy the necessary parts of the macro to the construction where you want to apply it.

This is an easy way to have macros, and you might ask why it is not implemented yet. The top reason is that the user interface to input macros still has to be designed, and that we have to find strategies to handle underconstrained constructions in macros.

12.2.3 Education

The *Cinderella* project was always driven by mathematics, and not by mathematics education. We are still sure that it does not make sense to adjust geometry in a way that it complies with the curricula, but it must be the curricula that are adjusted to match geometry. It is our strong believe that the principle of continuity is important for a proper understanding of geometry, and that the new possibilities of a software like *Cinderella* – we mean the mathematical possibilities, not the Internet support – are a chance to use geometry for teaching again. The trend should be to go back to the roots of modern geometry, and use this wonderful part of mathematics with help of the computer to teach better mathematics.

Of course, this a very personal view, and there is no empirical evidence that mathematically founded geometry software does really make a difference to other geometry software. This is a field of research for the future; but it will be hard to do this research because it needs the rethinking of geometry and math education as a whole. Again, it

does not make sense to put mathematical founded geometry software into the corset of a curriculum that has been built over decades and could not be focused on the new possibilities of Dynamic Geometry – if we can overcome this, we will have a chance to go new ways in education.

12.3 Other Applications

Finally, I want to mention a few other areas where the ideas presented in this thesis could be applied successfully after some modifications or extensions.

12.3.1 Computational Geometry

Computational Geometry is the part of computer science that works with data structures and algorithms for everything that is more or less related to geometry. We are sure that the methods that can handle the dynamic aspect of constructions can also be used to handle dynamic behavior of other geometry-like data.

A particular problem that must be solved for this is the proper handling of complex elements and orientations. Consider as an example that you want to add an algorithm for the smallest enclosing ellipse [18] to *Cinderella*. The algorithm is not able to handle complex points as input, since at some point there is a "$x \geq 0$" test required, which does not make sense for complex x. However, for points with real coordinates the smallest enlosing ellipse is continuous in the input parameters. So can we change the algorithm in a way that it also handles complex input?

Another prominent problem are convex hulls of point sets in the plane. These also need a kind of "sidedness"-test, it must be possible to check whether a point lies to the left or to the right of a line. Can we include a convex hull construction in *Cinderella*?

Both examples cannot be easily merged with *Cinderella*s complex analysis approach: The stationary behavior of both smallest enclosing ellipse and convex hull – if you move a point that is inside the ellipse or the hull, nothing happens until you try to move it across the boundary – shows that these are not real analytic functions, so even if we can extend the algorithms to complex space, we will never have analytic functions. In fact, the orientation decision could easily be used to create an absolute value function, which is not possible in a continuous system (Thm 6.19).

However, this does not exclude the chance of having a theory of partial continuity, where effects that are created by the orientation decisions are accepted as unavoidable, but for all other situations we still get continuous behavior. It will be part of the research in the next years to find a way to get merge continuity with orientations without loosing too much of each.

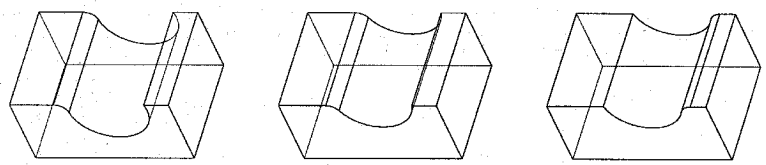


Figure 12.1: A jumping situation in parametric CAD: If the center of the circle that defines the drill hole in the block is moved across the boundary of the original block, the bevel jumps from one side to the other.

12.3.2 Parametric CAD/CAM

In *variational* or *parametric computer aided design* (parametric CAD) we find a situation that is similar to what we have in a Dynamic Geometry Software. It should be possible to change parameters of a CAD construction in way that slight parameter changes do not cause big changes in the construction. This can be used for instance to have a single prototype construction which can be customized quickly, or for data compression when we have to store a large number of similar objects, or just for easy and rapid construction of new models by starting with an approximate sketch that is made exact later.

Not only the situation is similar to Dynamic Geometry, the problems [9, 26, 25] are similar, too, see Fig. 12.1. And there is great hope that these problems, in the first place the "persistent naming problem," can be solved using the theory presented in this thesis. The most important next step to a complete solution of the persistent naming problem would be to extend the methods presented to constraint based design. The problems that arise in Computational Geometry we mentioned in the last section must be addressed, too, because many constraints that are used today in CAD are orientation-related.

12.3.3 Computational Kinematics

So far all constructions that we considered were abstract, there was motion, but no concept of mass. If we want to do physics simulation in a Dynamic Geometry system we need some other ingredients.

One thing would be to have a more sophisticated way to handle motion, for example based on the energy that is in the system. This could also help for the generation of loci. Another part would be to include some way to detect collisions between elements, a part that has been neglected so far. Closer inspection reveals that this again calls for a integration of orientation information into the methods of complex tracing. This shows that orientations should be #1 on our priority list of future research.

12.3.4 Virtual Reality

As the last example for areas where we hope to use the technology of Dynamic Geometry we want to mention "Virtual Reality," whatever that is.

Most of the research in the context of simulating the world on a computer is spent for 3D visualization and human-machine interfaces. These are important issues, and these are also very appealing topics since they create wonderful pictures and exciting technology.

Anyway, on the mathematical front there is still a lack of methods. Most simulations are "faked" in the sense that they either fail for certain situations, or they exclude these failures in the very beginning and thus are not as general as reality would be.

Try to find a simulation of Lego construction kit. There are some approaches, but you will not find one that really "works." The reason is that the same continuity problems that were unsolved for years in Dynamic Geometry software have to be solved in real world simulations. Just consider the 4-bar-linkage (Fig. 9.9 on p. 9.9). The locus that is generated by animating the linkage can be created by building a Lego model that draws the curve. If you can simulate Lego, you can create the complete locus – so you must be able to create complete loci when you want to be able to simulate the world.

It is not at all clear whether we want to have a simulated world, but if we do, we will have to understand the mathematics first, otherwise virtual reality will always remain a fake.

Chapter 13

Conclusion

"I would like to recall a man who stands somewhat on the side, the elder Carnot. The book of his that interests us here, his *Géomètrie de position*, appeared in 1803. Carnot (1753–1823) was a student of Monge in Mezières. As a general and a stalwart republican he played a significant role during the Revolutionary period. Only later did he regain the leisure for scientific work, mainly on fundamental mathematical problems.

His *Géomètrie* is a very remarkable book. It contains the significant modern thought: In geometry on should not isolate the various cases presented by a figure according to the arrangement of its parts – something that had been standard since Euclid – rather they should be given a common, unified treatment by introducing the principle of signs. But Carnot did not express this thought in quite this way. On the contrary, he stubbornly defended himself against the theory of signs so usual in analysis, considering it badly founded and contradictory. He believed he had proved this by constantly operating with many-valued functions in a purely formal manner, arriving at "false" results like $\sqrt{-a} \cdot \sqrt{-a} = \sqrt{a^2} = a$, etc. In geometry he intended that the rule of signs arise merely from considering the figure and its variations. On this bases he created a *"théorie des figures corrélatives."* In this way geometry was to be freed from the "hieroglyphics of analysis" and to arise anew in a purely synthethic form.

The execution of these ideas is occasionally rich in insight but often elementary to the point of triviality. Perhaps one may see this book as the counterpart to Carnot's incorruptible but not brilliant personality.

As a separate point I should mention that the very well known elementary theorem on the equality of the products of the segments formed on the sides of a triangle by an arbitrary transversal is due to Carnot, and it is often called Carnot's theorem.

Carnot's book is of historical importance for its rejection of analysis. This was the source of a dispute soon to emerge between analytic and synthetic modern geometry and which finally developed into an antagonism of major importance.

If Carnot's work already contains a vague presentiment of the direction in which modern geometry was to develop, then in Poncelet we find its great creator. He adopted Monge's and Carnot's ideas with the greatest brilliance and, conquering all difficulties,

created a breakthrough. By setting up "projection" and "duality" as unifying geometrical principles, he became the discoverer and founder of "projective geometry," which, uniting all previous oppositions, was to progress to great fertility. It was a new kind of geometrical intuition, "projective thinking," that enabled him to surpass his predecessors.

We have already spoken of the genesis of Poncelet's great geometrical work, the *Traité des propriétés projectives des figures* (1813 and 1822).

Poncelet began with a study of *central projections* and the relations among the parts of a figure that remain invariant under arbitrary central projections. This approach induced him to add to the ordinary geometrical elements certain definite "infinitely distant" ones: to the line he adjoined an infinitely distant point, to the plane an infinitely distant line, and to space an infinitely distant plane. He was then able to state theorems in all generality. Among these theorems the one on the constancy of the cross-ratio of four points on a line plays a major role. I do not wish to examine here the extent to which such ideas had been touched on by previous authors; it was Poncelet who made them the foundation of all further development and therin lay the essential progress of his conception.

A second important element of the new geometry was the theory of poles and polars for quadric curves and surfaces, leading to a general *theory of duality*. Points and lines in the plane, and points and planes in space are considered equivalent and can be substituted for each other as basic geometric elements. For example, to a plane curve composed of points there corresponds the curve enveloped by the tangents of the first, to a space curve there corresponds its developable curve, etc.

To these two new ideas there was then added the *principle of continuity*; this is Carnot's idea of the "correlativity of figure," but freed of all vagueness and brilliantly executed. This states that a relation known to hold with sufficient generality for a given figure also holds for all other figures that may be derived from it by continuous variation.

Poncelet made daring and far-reaching uses of this principle, venturing without a qualm into the realm of imaginaries when conditions seemed to require it. For example, from the fact that two conics intersect in at most 4 points, he deduced that the number of intersection points must always be 4, only that 2 or 4 of them may be imaginary. With this he provided a *projective definition of the circle* as a conic with two fixed imaginary points – today we call them the "circular points" – on the infinitely distant line. Likewise he defined the sphere as a quadric surface which intersects the infinitely distant plane in a given imaginary curve (known today as the "spherical circle"). Because the hyperboloid of one sheet is generated by two families of straight lines, this must also hold for the ellipsoid, only in this case the lines are clearly imaginary, etc.

What foundation did Poncelet provide for these audacious ideas? None at all, we are astonished to find. He gave no proof for the principle of continuity, which was intuitively clear to him; and he made no attempt to define the imaginary point. Evidently he felt no need to do anything of the kind, especially since his final results always contained conjugate complex elements and therefore moved entirely within the space of reals.

Only reference to analysis, which Poncelet rejected on principle, could have provided a secure foundation for these new concepts. An imaginary point, like a real one, is just

a common solution of a number of simultaneous equations, each of which represents one of the intersecting geometrical objects. That the same number of objects of equal degree always produce intersections of the same kind, e.g., the same number of "points," is just a theorem on the common solutions of algebraic equations; the solutions may be real or complex depending on the relations among the coefficients, but systems having the same number of equations of the same degrees have the same number of solutions. And as for the principle of continuity itself, it is not hard to establish rigorously by the modern theory of functions. Every geometrical statement can be expressed analytically (understanding geometry in the restricted sense that was usual in Poncelet's day) by equating to zero an algebraic or even analytic function $f(a, b, c, \dots)$ of the parts $a, b, c, \dots$ of the figure. Then the principle of continuity simply states that an analytic function which vanishes on any part, no matter how small, of its domain, must vanish everywhere.

Poncelet should be seen as one of the greatest representatives of that class of mathematicians whom we have characterized as daring conquerors. His influence runs through the whole 19th century and has become an essential part of our thought."

Felix Klein, Development of Mathematics in The 19th Century, Berlin 1928
(Translation by M. Ackermann)

Appendix A

Alphabetic Glossary

Angular Bisector One of the two lines that cuts one of the two included angles of two other lines into two equal parts. Most →Dynamic Geometry systems cannot handle iterated angular bisectors correctly.

Applet A →Java program that is embedded in a web page and can be displayed by a Java-compliant →Browser.

Browser A software like Netscape or Internet Explorer that is used to display →HTML, which might be downloaded via the →Internet.

Cayley-Klein Geometry Unified treatment of measurements in →hyperbolic, →elliptic, relativistic and →Euclidean geometry (and some others). Needs a →fundamental conic and uses →cross ratios to calculate the distances and angles. See section 5.4

Cabri A Dynamic Geometry software [54].

Cinderella A new Dynamic Geometry software [71, 70].

Conic (or conic section) A curve that is the solution space of a quadratic form. Another way to charaterize conics is to cut a 3-dimensional circular cone with a plane. The curve created on the plane by the cone is the conic section.

Cross Ratio The cross ratio $(AB|CD)$ of four points on a line is defined by introducing coordinates on the line and calculating $\frac{(A-C)(B-D)}{(A-D)(B-C)}$, identifying the points with their coordinates. The cross ratio is projectively invariant and can be used to do measurements with respect to a →fundamental conic.

Drag Press the mouse button, move the mouse, release the mouse button. If you drag objects, you start by pressing the mouse on the object, like grabbing it.

Dynamic Geometry system A computer software which can be used to construct with points, lines, circles, →conics and possibly more objects,

and which lets you →drag →free elements later while maintaining the construction automatically.

Fundamental conic A →conic that determines the measurements in →Cayley-Klein geometries. It plays the role of infinity. In case of degenerate conics we need also a dual conic to form a →fundamental pair.

Fundamental pair (of conics) A →conic and a dual conic that together determine the measurements in →Cayley-Klein geometries.

General Position A set of geometric objects that avoids certain degeneracies that have to be specified somewhere is in general position. Opposite of →special position.

HTML Markup language to describe web pages.

Internet The revolution of the millenium. Computers are connected all around the world, and can communicate using simple protocols. Most popular applications of the Internet are the →World-Wide-Web and eMail.

Java An interpreted, platform independent computer language created by James Gosling (Sun Microsystems) [20]. Can be used inside web pages, but also for standalone applications.

Locus The path of a dependent point while moving a movable object. The locus of a line is usually the envelope created by the moving line.

Pappos' Theorem The "smallest" incidence theorem in the projective plane. Very useful as an example. A generalization is →Pascal's Theorem.

Pascal's Theorem Given two groups A_i and B_i of three points on a conic, then the three intersections C_i of A_jB_k and A_kB_j, $k \neq j \neq i \neq k$, are collinear. Specialized version is Pappos' Theorem. The polar version of this theorem is Brianchon's theorem.

Sketchpad A Dynamic Geometry software [36]

Special Position A set of geometric elements that satisfies a certain degeneracy condition. Opposite of →general position.

Appendix B

Bibliography

[1] *Mechanical theorem proving in geometries. Basic principles. Transl. from the Chinese by Xiaofan Jin and Dongming Wang.* Texts and Monographs in Symbolic Computation. Springer-Verlag, Berlin, 1994.

[2] Angliacampus. http://www.angliacampus.uk. Educational website using Cinderella for geometry.

[3] A. Beckmann. Zweischrittiger Computereinsatz beim Beweislernen im Geometrieunterricht. Satzfindung und Beweisfindung. *Mathematik in der Schule*, pages 301–308, 1997.

[4] Alex Below, Ulrich Kortenkamp, and Jürgen Richter-Gebert. *Primitive Geometric Operations*. in preparation, 2000.

[5] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation*. Springer, New York, 1998. The homotopy method might be a key for finding stability criteria for complex tracing.

[6] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *A Series of Comprehensive Studies in Mathematics*, chapter 4, pages 103–124. Springer-Verlag, Berlin Heidelberg New York, 1997. Straightline programs in algebraic complexity theory.

[7] Christoph Burnikel, Rudolf Fleischer, Kurt Mehlhorn, and Stefan Schirra. A strong and easily computable separation bound for arithmetic expressions involving square roots. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA97)*, pages 702–709, New York, Philadelphia, January 1997. ACM Press/SIAM Publications. Although the bound is easily computable, square root expressions are not.

[8] Otfried Cheong. The Ipe extendible drawing editor. http://www.cs.ust.hk/ otfried/Ipe/Ipe.html, 1994. A nice extensible drawing editor.

[9] D-Cubed. Dcm benchmarks. http://www.d-cubed.co.uk/dcm_benchmarks.html. Some benchmarks for variational/parametric capabilities of CAD software.

[10] Mike Deng. The parallel numerical method of proving the constructive geometric theorem. *Chinese Sci. Bull.*, 34:1066–1070, 1989.

[11] J'Express. http://www.denova.com. A Java software installation software.

[12] Bryan Dye. Mathsnet. http://www.anglia.co.uk/education/mathsnet/. First external collection of interactive geometry examples created with Cinderella.

[13] Maurits Cornelis Escher and F. Bool. *M.C. Escher: His Life and Complete Graphic Work*. Harry N. Abrams, 1992. A 99.8% complete collection of Escher's work.

[14] Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schönherr. On the design of CGAL, the computational geometry algorithms library. *Software – Practice and Experience*, 1999. to appear; preliminary version available as technical report #291, Departement Informatik, ETH Zurich, Switzerland, February 1998.

[15] Wolfgang Fischer and Ingo Lieb. *Ausgewählte Kapitel aus der Funktionentheorie*. Vieweg, Braunschweig, Wiesbaden, 1988. All the complex analysis you need for this thesis is contained in chapter II of this book.

[16] Wolfgang Fischer and Ingo Lieb. *Funktionentheorie*. Vieweg, Braunschweig, Wiesbaden, 1988. A very basic introduction into complex analysis.

[17] Hans Freudenthal. The impact of von Staudt's foundations of geometry. In R. S. Cohen, J. J. Stachel, and M. W. Wartofsky, editors, *For Dirk Struik*, pages 189–200. D. Reidel, Dordrecht-Holland, 1974. An article emphasizing the foundation-laying contribution (in terms of purely algebraic description) of von Staudt to projective geometry.

[18] B. Gärtner and S. Schönherr. Exact primitives for smallest enclosing ellipses. *Information Processing Letters*, 68:33–38, 1998.

[19] Th. Gawlick. Beeinflusst der Einsatz von Geometrie-Software das Herausbilden von Grundvorstellungen? In *Beiträge zum Mathematikunterricht*. Franzbecker, Bad Salzdetfurth, 1999. (to appear).

[20] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. The Java Series. Addison-Wesley, September 1996.

[21] Robin Hartshorne. *Companion to Euclid*. Springer-Verlag, New York, 2000. A modern guide to the elements of Euclid.

[22] Gaby Heintz. *Überprüfung des Lernerfolgs durch die Benutzung von Tagebuch-Protokollen bei interaktiven Geometrieprogrammen – am Beispiel von Cinderella.* PhD thesis, Gerhard Mercator Universität Duisburg, in preparation.

[23] Joost Heintz and Claus-Peter Schnorr. Testing polynomials which are easy to compute. In *Logic and algorithmic, int. Symp., Zuerich 1980, Monogr. L'Enseign. Math. 30, 237-254 (1982).*

[24] Reinhard Hölzl. *Im Zugmodus der Cabri Geometrie.* PhD thesis, Universität Augsburg, 1994. An empirical study concerning the use of geometry software in mathematics education.

[25] Christoph M. Hoffmann. How solid is solid modeling? In Ming C. Lin and Dinesh Manosha, editors, *Applied Computational Geometry – Towards Geometric Engineering*, LNCS, pages 1–8. Springer-Verlag, 19?? Hoffmann is asking for a theory for parametric CAD, which is presented in this thesis.

[26] Christoph M. Hoffmann. *Solid Modeling*, chapter 47, pages 863–880. Discrete Mathematics and its Applications. CRC Press, Boca Raton, New York, 1997. Here Hoffmann explains the "difficult problem of persistent naming", and underlines its theoretical and practical importance. In the future a working approach to persistent naming could help to modularize solid modeling systems into standardized components.

[27] Gerhard Holland. *Geolog-Win.* Dümmler, Bonn, 1996. A axiomatics-based geometry software, containing Prolog-driven exercise facilities.

[28] Jiawei Hong. Can we prove geometry theorems by computing an example? *Sci. Sinica*, 29:824–834, 1986.

[29] Jiawei Hong. Proving by example and gap theorems. In *Proc. 27th Ann. Symp. Foundations Comp. Science*, pages 107–116, Toronto, October 1986. IEEE. Gives bounds for values that can be used for proving polynomials to be identical to zero by evaluating at one point.

[30] C. Hoyles. Microworlds / schoolworlds: the transformation of an innovation. In *Learning from computers: mathematics education and technology.* C. Keitel and K. Ruthven, Berlin, 1993.

[31] Ulrich Huckenbeck. *Geometrische Maschinenmodelle.* PhD thesis, Bayerische Julius-Maximilians-Universität Würzburg, 1986. Huckenbeck discusses what functions are computable using ruler and compass or ruler and right angles. However, he is considering the situation non-dynamically, which considerably restricts the set of computable functions.

[32] Oscar H. Ibarra and Brian S. Leininger. On the simplification and equivalence problems for straight-line programs. *J. Assoc. Comput. Mach.*, 30:641–656, 1983. It is shown that the simplification and equivalence problems for straight-line programs are unsolvable for all nontrivial classes.

[33] Install Toolkit for Java. http://www.alphaworks.ibm.com/tech/installtoolkit. A Java software installation software.

[34] Remote AWT. http://www.alphaworks.ibm.com/tech/remoteawt. A client/server software that is able to route the Java window system to another machine on the network, without requiring X-Windows or similar software.

[35] Installshield for Java. http://www.installshield.com. A Java software installation software.

[36] Nicholas Jackiw. *The Geometer's Sketchpad.* Key Curriculum Press, Berkeley, 1991–1995.

[37] Nick Jackiw. Javasketchpad. http://www.keypress.com/sketchpad/java_gsp/index.html. A Java applet that can display geometric constructions that are similar to constructions done with Geometers' Sketchpad.

[38] Nicolas Jackiw, Ulrich Kortenkamp, Jean-Marie Laborde, Jürgen Richter-Gebert, and al. A quality standard for dynamic geometry software. This shall set a lower standard for geometry software, divided into categories like user interface, mathematics, overall stability. A future goal is an exchange format for geometric constructions. Hopefully this paper will be the result of a developer's conference for Dynamic Geometry., in preparation.

[39] Erich Kaltofen. Greatest common divisors of polynomials given by straight-line programs. *Journal of the Association for Computing Machinery*, 35(1):231–264, January 1988. Notion of algebraic straight-line programs.

[40] Felix Klein. *Elementarmathematik vom höheren Standpunkt aus.*, volume 2 of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen*. Springer-Verlag, reprint 1968 edition, 1925. Felix Klein would not have been surprised at all by the solution to the continuity problem in Dynamic Geometry.

[41] Felix Klein. *Vorlesungen über die Entwicklung der Mathematik im 19. Jahrhundert.* Springer-Verlag, Berlin, 1928. Wonderful.

[42] Felix Klein. *Vorlesungen über nicht-euklidische Geometrie*, volume 26 of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen*. Springer-Verlag, Berlin, 1928. Reprint 1968. A must-read for everyone doing geometry.

[43] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, Mass., 1968. A must-read even today.

[44] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, Mass., 1969.

[45] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, Mass., 1969. Use heapsort!

[46] Ulrich Kortenkamp and Jürgen Richter-Gebert. Cinderella website. http://www.cinderella.de. All about Cinderella: examples, infos, links, etc.

[47] Ulrich Kortenkamp and Jürgen Richter-Gebert. Geometry and education in the internet age. In *Proceedings of the ED-MEDIA & ED-TELECOM 1998 World Conference on Educational Multimedia, Hypermedia and Telecommunications*, pages 790–799, Freiburg, March 1998. Association for the Advancement of Computing in Education. Implications of Java based, internet-aware software on (geometry) education. Available at http://www.cinderella.de/papers/geo-i.pdf.gz.

[48] Ulrich Kortenkamp and Jürgen Richter-Gebert. Cinderella. In *Erfahrungen mit Java*, chapter 16, pages 381–401. dpunkt.Verlag, Heidelberg, 1999. Overview over the Cinderella project with special emphasis on the use of Java (in German).

[49] Ulrich Kortenkamp and Jürgen Richter-Gebert. Euklidische und Nicht-Euklidische Geometrie in Cinderella. In Thomas Weth, editor, *Nürnberger Kolloquium zur Didaktik der Mathematik*, Nürnberg, April 1999. A high-speed introduction starting from scratch into non-Euclidean geometry. Available at http://www.cinderella.de/papers/nichtEuklidisch.pdf.

[50] Gilles Kuntz. Cabri-java. http://www.cabri.net/cabrijava. A Java applet that can read some original Cabri files and display them interactively.

[51] Colette Laborde. Visual phenomena in the teaching/learning of geometry in a computer-based environment. In Carmelo Mammana, editor, *Perspectives on the Teaching of Geometry for the 21st Century*. 1998.

[52] Jean-Marie Laborde. Exploring non-euclidean geometry in a dynamic geometry environment like Cabri-géomètre. In James King and Doris Schattschneider, editors, *Geometry Turned On*, volume 41 of *MAA Notes*, pages 185–192. MAA, 1997. Jean-Marie Laborde asks for a complete model for dynamic geometry, and stresses the importance of a real mathematical treatment.

[53] Jean-Marie Laborde. Some issues raised by the development of implemented dynamic geometry as with cabri-géomètre. In Hervé Brönnimann, editor, *Proceedings of the 15th European Workshop on Computational Geometry*, pages 7–19, Antibes

– Juan-les-Pins, March 1999. INRIA Sophia Antipolis. Extended abstract of a talk given at the Conference. During the talk the Differences between Cabri and Cinderella were shown on a computer (not mentioned in the abstract).

[54] Jean-Marie Laborde and Franck Bellemain. *Cabri-Geometry II.* Texas Instruments, 1993–1998. Copyright Texas Instruments and Université Joseph Fourier, CNRS.

[55] Susan Landau. How to tangle with a nested radical. *Mathematical Intelligencer*, 16(2):49–55, 1994. An expository paper that demonstrates that it is very hard to denest (or simplify) an expression of nested radicals.

[56] Serge Lang. *Algebra.* Addison-Wesley, Reading, MA, 2nd edition, 1984. A standard reference textbook for algebra.

[57] Nancy A. Lynch. Straight-line program length as a parameter for complexity analysis. *J. Comput. Syst. Sci.*, 21:251–280, 1980.

[58] Roland Mechling. Euklid. http://www.mechling.de. Homepage of Euklid, a german shareware geometry software.

[59] Roland Mechling. Euklid. Another geometry software.

[60] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*, chapter 7. Cambridge University Press, Cambridge, 1995. Good and broad introduction to the theory of randomized Algorithms.

[61] Jürg Nievergelt, Peter Schorn, Michele De Lorenzi, Christoph Ammann, and Adrian Brüngger. eXperimental geometrY Zurich – software for geometric computation. Gelbe Reihe 163, Departement Informatik, Institut fü Theoretische Informatik, ETH Zürich, Zürich, July 1991. A project which was very similar to CGAL.

[62] Mark H. Overmars. Designing the computational geometry algorithms library cgal. In M. C. Lin and D. Manocha, editors, *ACM Workshop on Applied Computational Geometry*, Philadelphia, Pennsylvenia, May, 27–28 1996. Lecture Notes in Computer Science 1148.

[63] Julius Plücker. Ueber ein neues Coordinatensystem. *Crelle's Journal*, 5:1–36, 1829. A very nice article that describes the benefits of homogenization.

[64] Jean-Victor Poncelet. *Traité des propriétés projectives des figures.* Gauthier-Villars, 1822.

[65] William Pugh. Compressing java class files. In *Proceedings of the ACM SIGPLAN '99 Conference on Programming Language Design and Implementation*, Atlanta,G GA, 1999. An alternative class file format for Java that reduces the byte code size by a significant factor.

[66] Jürgen Richter-Gebert. Mechanical theorem proving in projective geometry. *Annals of Mathematics and Artificial Intelligence*, 13:139–172, 1995. Presentation of the method of proving projective geometry theorems using bi-quadratic final polynomials.

[67] Jürgen Richter-Gebert. How to do geometry on a computer. A 2-hour lecture given by Jürgen at various occasions that is a jump start into oriented Projective Geometry., 1996–1999.

[68] Jürgen Richter-Gebert. Primitives for geometric operations. Course material for the Equinoctial School, ETH Zürich, September 1997. A concise introduction to geometric operations in projective and metric geometry.

[69] Jürgen Richter-Gebert and Ulrich Kortenkamp. OO-Graphikprogrammierung in Java. Technical report, ETH Zürich, 1997–1999. Course material for a two-day introductory course to Java. In the exercises a class hierarchy for dynamic drawing software is developed, following the traditional model of Dynamic Geoemtry software.

[70] 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.

[71] Jürgen Richter-Gebert and Ulrich Kortenkamp. The interactive geometry software Cinderella. Book & CD-ROM, Springer-Verlag, Berlin Heidelberg New York, 1999. First commercial release of the Cinderella software.

[72] Jürgen Richter-Gebert and Ulrich Kortenkamp. Complexity issues in Dynamic Geometry. *in preparation*, 2000. A complete coverage of the complexity results in Dynamic Geometry.

[73] Franco Saliola. Non-Euclidean Geometry with Cinderella on the Internet. http://members.xoom.com/fsaliola. First examples on the net using hyperbolic geometry with Cinderella.

[74] Jacob T. Schwartz. Probabilistic algorithms for verification of polynomial identities. In *Symbolic and algebraic computation, EUROSAM '79, int. Symp., Marseille 1979, Lect. Notes Comput. Sci. 72, 200-215 (1979).*

[75] Jorge Stolfi. *Oriented Projective Geometry: A Framework for Geometric Computations*. Academic Press, New York, NY, 1991. A complete description of an oriented version of Projective Geometry, for all dimensions, but only for flats (linear subspaces). Suitable for computer scientists.

[76] Volker Strassen. Berechnung und Programm I. *Acta Informatica*, 1:320–335, 1972. Wonderful example of straight-line programs in an algebraic setting.

[77] Bernd Sturmfels. *Algorithms in Invariant Theory*. Texts and Monographs in Symbolic Computation. Springer-Verlag, Wien New York, 1993. Covers important topics, like the straightening algorithm and its applications in automatic theorem proving.

[78] Frank Tip, Chris Laffra, and Peter F. Sweeney. Jax – java application extractor. http://www.alphaworks.ibm.com/formula/jax. A post-optimization tool for Java software. You can provide additional knowledge about your software to disable dynamic linking for certain classes. Also removes dead code and classes.

[79] Frank Tip, Chris Laffra, Peter F. Sweeney, and David Streeter. Pratical experience with an application extractor for java. In *Proceedings of the 14th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '99)*, Denver, Colorado, November 1999. ACM. Among others, Cinderella is used as a benchmark example for Jax to show the benefits of using an application extractor approach in object oriented programming.

[80] Karl Georg Christian von Staudt. *Geometrie der Lage*. Bauer & Raspe, Nürnberg, 1847.

[81] Karl Georg Christian von Staudt. *Beiträge zur Geometrie der Lage*. Bauer & Raspe, Nürnberg, 1856.

[82] Dongming Wang. Geometry machines: From ai to smc. In Jaques Calmet, John A. Campbell, and Jochen Pfalzgraf, editors, *Proc. Artificial Intelligence and Symbolic Mathematical Computation 3*, volume 1138 of *Lecture Notes in Computer Science*, pages 213–239, Steyr, September 1996. Springer-Verlag. Very good overview over the currently used techniques in automatic theorem proving (including 171 references).

[83] Thomas Weth. Begleitmaterial zu Vorlesungen der Didaktik der Mathematik. http://www.didmath.ewf.uni-erlangen.de/Vorlesungen/Geometrie_HS/index.htm. Prof. Weth uses Cinderella to illustrate the course material for his lectures.

[84] Thomas Weth. Kegelschnitte und höhere Kurven als Ortslinien in Dreiecken. http://www.didmath.ewf.uni-erlangen.de/kegel_weth/index.html. Accompaining web based examples to Weth's creativity research.

[85] Thomas Weth. Kreative Zugänge zum Kurvenbegriff. *Der Mathematikunterricht*, 4/5, 1998.

[86] Thomas Weth. *Kreativität im Mathematikunterricht – Begriffsbildung als kreatives Tun*. Franzbecker, Hildesheim, Berlin, 1999. A very interesting approach to creativity that shows (among other things) the new possibilities that arise when computers are used in teaching.

[87] Harald Winroth. *Projective Dynamic Geometry*. PhD thesis, KTH Stockholm, March 1999. Many basics of Projective Geometry on a computer. Tries to solve the continuity problem with orientations. Available at http://www.lib.kth.se/fulltext/winroth990324.pdf.

[88] Niklaus Wirth. The programming language Pascal. *Acta Informatica*, 1:35–63, 1971. This issue of Acta Informatica is a must-read for everybody who is interested in the history of computer science.

[89] Wen-tsuen Wu. On the decision problem and the mechanization of theorem-proving in elementary geometry. *Contemp. Math. 29*, pages 213–234, 1984.

[90] Lu Yang. A new method of automated theorem proving. In J. Johnson and M. Loomes, editors, *The mathematical revolution inspired by computing*, pages 115–126. Oxford University Press, New York, 1991.

[91] Lu Yang, Jingzhong Zhang, and C.-Z. Li. A prover for parallel numerical verification of a class of constructive geometry theorems. In *Proc. IWMM '92*, pages 244–250, Beijing, July 1992.

[92] InstallAnywhere. http://www.zerog.com. A Java software installation software.

[93] Jingzhong Zhang, Lu Yang, and Mike Deng. The parallel numerical method of mechanical theorem proving. *Theoretical Computer Science*, 74:253–271, 1990.

Appendix C

Curriculum Vitae

Ich wurde am 18. Oktober 1970 als Ulrich Hund in Köln am Rhein geboren, und bin das einzige Kind von Ulrike Hund, geborene Müller, und Gerhard Hund. In Kerpen, Erftkreis, NRW, wuchs ich auf und besuchte zunächst die Evangelische Grundschule Kerpen und dann die Gemeinschaftsgrundschule Kerpen-Sindorf. 1980 wechselte ich auf das Tagesheimgymnasium Kerpen (inzwischen Gymnasium Kerpen), welches ich am 19. Mai 1989 mit Abitur wieder verlassen durfte. Ich belegte dort die Leistungskurse Physik und Mathematik, und ich hatte 823 von 900 möglichen Punkten, was eine 1,0 als Endnote ergab.

Mein akademisches Leben begann in Münster, Westfalen, wo ich zum Wintersemester 1989/1990 das Studium der Mathematik mit Nebenfach Informatik aufnahm. Am 17. Oktober 1991 erhielt ich das Vordiplom. Zwischen Dezember 1994 bis Februar 1995 machte ich die Diplomprüfungen, und am 31. März erhielt ich das Diplom. Der Titel meiner Diplomarbeit lautete „Pseudosphärenarrangements zu Orientierten Matroiden".

Bereits im März 1993 war ich nach Berlin gezogen, wo ich dann ab 1. April 1994 an der TU als Wissenschaftlicher Mitarbeiter angestellt war. Mein ehemaliger Doktorvater dort war der Koreferent dieser Arbeit, Günter Ziegler. Ich habe meine Dissertation in Berlin nicht abgeschlossen, weil ich die Gelegenheit ergriff an das Departement Informatik der ETH Zürich zu wechseln, wo mir Jürgen Richter-Gebert eine Anstellung als Mathematiker angeboten hatte. Seit September 1997 bin ich am Institut für Theoretische Informatik angestellt.

Am 4. Oktober 1996 habe ich Doro Kortenkamp geheiratet. Wir haben zwei Kinder, Mara, geboren am 5. März 1997, und Julius, geboren am 22. Februar 1999.

I was born as Ulrich Hund on October 18, 1970 in Cologne, Germany, being the only child of Ulrike Hund, born Müller, and Gerhard Hund. I grew up in Kerpen, Erftkreis, NRW, Germany, where I attended the Evangelische Grundschule Kerpen and the Gemeinschaftsgrundschule Kerpen-Sindorf as primary school. Then I changed in 1980 to the Tagesheimgymnasium Kerpen (now Gymnasium Kerpen), where I got my Abitur on May, 19th, 1989. My profile courses were Physics and Mathematics, the final score was 823 out of 900 points, a 1,0 in the German censoring scale.

My academic life began at Münster, Westphalia, where I studied Mathematics with a minor in Computer Science starting in the winter term of 1989/1990. On October 17, 1991, I earned the Vordiplom in Mathematics. From December 1994 to February 1995 I took the examinations for the math diploma, which I received on March 31st, 1995. The title of my diploma thesis was "Pseudosphärenarrangements zu Orientierten Matroiden."

I had moved to Berlin already in March 1993, and I got a position as teaching assistent in the maths department at the Technical University of Berlin starting on the first of April 1994. My former advisor there was the co-examiner of this thesis, Günter Ziegler. I did not finish my Ph.D. studies in Berlin, instead I took the opportunity to change to the Department of Computer Science at the ETH Zürich where Jürgen Richter-Gebert offered me a position as a mathematician. I work at the Institute of Theoretical Computer Science since September 1997.

On October 4, 1996, I married Doro Kortenkamp. We have two children, Mara, born March 5, 1997 and Julius, born on February 22, 1999.