Pen-based Input of Geometric Constructions*

Ulrich Kortenkamp, Dirk Materlik

July 16, 2,004

Abstract

Pen-based input is the classical approach to drawing. Without computers, using a pen was the natural approach to do drawings, and the simplicity of this "device" always called for similar electronic solutions.

Today, a broad range of electronic input devices is available that try to pair the simplicity of input with the power of computing. They come in all sizes: from small touch-screens with stylus in personal digital assistants, via desktop-size input tablets or interactive displays, to electronic whiteboards with rear- or front-projection. These devices all share a direct interaction paradigm.

The challenge in using these devices for geometric constructions is to automatically annotate the drawn objects with their relations to the other objects. A basic example is the drawing of a right-angled triangle: How can we automatically recognize the user's intention to have a right angle? Can we do that at all? Do we really want to do that? If not – what is the easiest pen-compatible way to have the user add this information? We will demonstrate several approaches to these problems, implemented in a development version of *Cinderella*.

Another interesting issue is the presentation of hand-drawn figures: Is it possible to preserve the look of a sketch while still guaranteeing mathematically correct figures? Using the additional information gathered by automatic theorem proving we show some solutions.

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

1 Pen-Driven devices

Recent advances in computer technology enable the use of Dynamic Geometry Software in new scenarios. *Personal Digital Assistants* (PDAs), such as the one shown in Fig. 1, are palm-sized handheld computers originally developed for keeping track of addresses, appointments, and so on. They have become powerful enough to run Dynamic Geometry Software. In particular, the Dynamic Geometry Software *Cinderella* [2] by Jürgen Richter-Gebert and Ulrich Kortenkamp can run as it is written in Java, although its user interface is less than optimal on these devices. PDAs are preferable to laptops or desktop computers in some situations because of portability. In other situations, e.g., in a classroom situation, they are preferable because they get less in the way of human-human interaction.

Figure 1: A Sharp *Zaurus* SL-C700 running Cinderella.

Looking at the other extreme in size, *Interactive Whiteboards* have become affordable enough to be in wide use. They aim to replace conventional black- or whiteboards with computer-based devices. The picture from the computer is projected onto the whiteboard using a projector. The board reports the position of a special pen back to the computer, where this data is interpreted as regular mouse movements. All this normally should be transparent to the application running on the computer. Fig. 2 shows a Numonics Whiteboard.

Interactive Whiteboards can enhance the typical situation of a presentation. Previously, the presenter had to concern him- or herself with the laptop computer while using Dynamic Geometry Software. Interactive whiteboards allow input from the natural position of the presenter, in front of the audience.

Both of these new devices are *pen-driven* – a physical object, the pen, is used to control

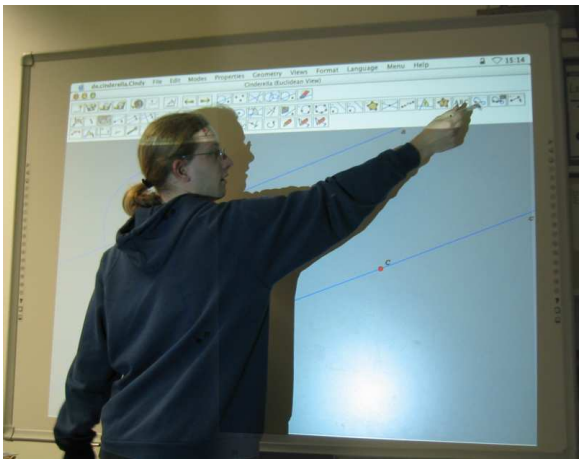


Figure 2: A Numonics Whiteboard attached to a computer running Cinderella.

the pointer position. This implies a direct interaction paradigm; the logical position of the mouse pointer always corresponds to the physical position of the user's hand. Although the devices simulate a mouse and their use is thus transparent to applications, applications should be adapted to this new paradigm. Normal user interfaces are not very suitable, requiring too much movement of the hand (e.g., toolbars) or too exact pointing (e.g., multi-level context menus).

Pen-driven input devices also exist for desktop computers in the form of graphics tablets. These face similar issues, but lack the immediacy of drawing on the displayed construction.

2 Constructions

The difference of Dynamic Geometry Software to vector-based drawing programs or CAD systems is that Dynamic Geometry Software internally stores the mathematical relationships of the elements. This is done implicitly as the construction is created. When one of the starting points is then moved, the computer recalculates the positions of all dependent elements. The user is presented with the illusion of a continuous, smooth movement. This allows him or her to see how the movements of the starting points affect the other elements, and thus intuitively grasp geometric relationships [1].

Typical elements of constructions are (free) points, lines (defined by two points), circles defined by midpoint and border point, circles defined by three border points, intersection points etc. For pen-based input of constructions, the goal is to allow the user to "just sketch" the desired elements, recognizing them automatically. Mathematical knowledge in the software is necessary to actually decide how the sketched element fits into the construction and create a fitting element for it. For example, when a user marks a point near the intersection of two lines, the intersection point should be inserted into the construction.

The main challenge in recognizing pen-drawn shapes is to correctly identify the relations, such as orthogonality, of the new element to elements already in the construction.

3 Recognizing Pen-Based Input

3.1 Identifying Relations in *Cinderella Scribbling*

To adapt Cinderella to the use on the new pen-driven devices introduced in section 1, we implemented a sketch-recognition framework to directly recognize shapes [3]. We call the mode utilizing this framework *Scribbling*. Fig. 3 shows the user's guide to Scribbling, listing the possible gestures and recognized elements.

One interesting aspect is how to recognize elements that depend on other elements. The case is relatively easy for, e.g., lines defined by two points: if the user draws a line-shaped sketch between two existing points, the line between those two points naturally is the line to insert.

However, already midpoints are more difficult: just recognizing a point drawn close to the midpoint between two already existing points as the midpoint quickly leads to undesired behavior when there are more than a few points in the construction. We thus took the approach of taking preselections into account: if two points are selected before another is drawn close to their midpoint, the midpoint is inserted. Selecting elements is done by tapping on them; when using a pen, this seems very natural.

For consistency, orthogonals and parallels are handled likewise: preselecting a line before drawing another one enables using it as an input for the new element.

Another approach is to use after-the-fact annotations. For instance, a short angled stroke across the angle of two intersecting lines could be used as an indicator of orthogonality, similar to the way right angles are annotated in traditional mathematical drawings. This possibly has the advantage of being the more intuitive way, building upon the user's existing knowledge. However, there are complications in actually implementing it. It is easy to think of cases in which it is not clear how to establish the right angle between two previously unrelated lines without breaking the construction. In a prototype, we implemented the after-the-fact annotations by restricting the mark to the last element added, which can always be modified accordingly. However, this restriction seems somewhat arbitrary to the user and it also leads to further problems when taking undo and redo operations into account.

On balance, the approach using preselections is exact and easy to use after a short time and we thus prefer it.

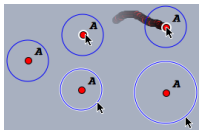
3.2 Presentation in a "Hand-Drawn" Way

While using *Scribbling*, users remarked that they feel being corrected by the computer because the elements are drawn straight and mathematically correct after being recognized. However, preserving the drawn shapes is not an option, either: the relations between elements need to be shown correctly.

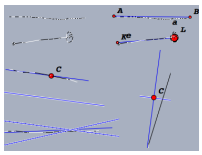
Figure 4 shows our solution, implemented by Richter-Gebert: the user's sketch is preserved, except at those points where it is necessary to deviate from it. In this example, the circle around the intersection of the perpendicular bisectors of the triangle's sides has to pass through all the vertices of the triangle.

The user's sketch is internally stored as deviations from the real path of the element.

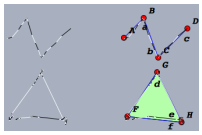
Scribbling is a new user interface for creating Cinderella constructions. It is based on the idea of directly recognizing sketches to the geometric objects you intended to draw. It is most useful on pen-driven devices such as interactive whiteboards, PDAs or graphics tablets.



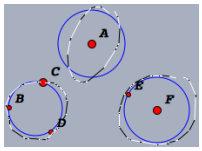
Selecting and Moving
Tap an object once to select or deselct it. Tap in an empty space to deselect all. Move a selected object by dragging it.
This applies to points as well as other free elements, such as *Circle by Radius.*



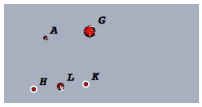
Lines
Draw a line to create a line through two points. The endpoints are created if they do not exist. Short lines are cut off at the endpoints, lines over 80% of the window are not.
Preselect another line and draw a scribble parallel or orthogonal to it to create a *Parallel* or *Orthogonal*.
Preselect two lines and draw an *Angular Bisector* to create one.



Polylines
Draw multiple lines to create a sequence of segments.
If the last point is near the first point, a *Polygon* is created.



Circles
Draw a circle to create a *Circle By Radius.* A midpoint is created if it does not exist. Preselect a midpoint for greater tolerance.
Pass exactly 3 existing points to create a *Circle By 3 Points.* Pass exactly 1 point to create a *Circle By Midpoint and Border Point.*



Points
Draw a small scribble to create a *Free Point.* Preselect two points and draw a small scribble in the middle of those to create a *Midpoint.*

Undo and Redo
Draw a horizontal, straight, quick scribble over 30% of the window width to the left to undo the last action; to the right to redo an action.

Edit Label
Double-click an element, i.e., tap it twice within a short time, to edit its label.

Delete All
Draw a quick, large gesture, crossing out all of the construction, to delete all elements.

Inspect
A straight line down, then back up to the startpoint opens the *Inspector.*

Right Click
A straight line up, then back down simulates a right mouse button click. Usually, this brings up the context menu.

Figure 3: The User's Guide to *Cinderella* Scribbling.

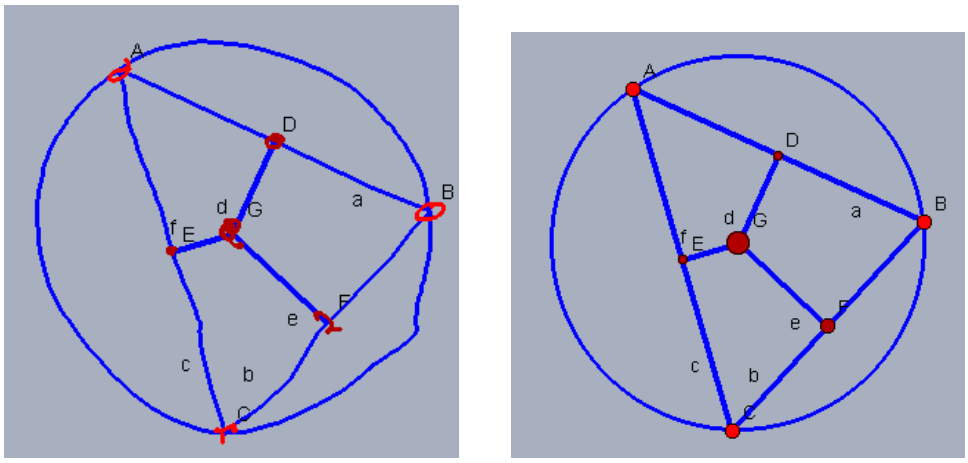


Figure 4: A construction shown without and with correction.

It is then modulated to become zero at those points where the element is incident to other elements. This approach is only possible because Cinderella has the mathematical knowledge to find these incidences via its internal prover. In the figure, Cinderella knows that the circle must be incident to the three points, although it has been drawn only through one of them. Storing the deviations also enables us to handle resizing in a natural way, e.g., when the points defining an element change in distance.

3.3 Conclusion

Mathematical knowledge about the construction is thus necessary at several levels when adapting Dynamic Geometry Software to pen-based input. We need it to find the possible candidates for a newly sketched element; to correctly guess the user's intention when drawing dependent elements; and to present the sketched elements in a "hand-drawn" way while preserving the construction's mathematical integrity.

References

[1] Ulrich Kortenkamp: Foundations of Dynamic Geometry. Dissertation at ETH Zürich, No. 13403. 1999.

[2] Ulrich Kortenkamp, Jürgen Richter-Gebert: The Interactive Geometry Software Cinderella. Springer-Verlag 1998.

[3] Dirk Materlik: Using Sketch Recognition to Enhance the Human-Computer Interface of Geometry Software. Diploma Thesis at FU Berlin, 2003.