From CAS/DGS Integration to Algorithms in Educational Math Software*

Ulrich Kortenkamp and *Andreas Fest*
kortenkamp@ph-karlsruhe.de / andreas.fest@ph-gmuend.de
Department of Mathematics and Computer Science
University of Education Karlsruhe
and University of Education Schwäbisch Gmünd
Germany

Abstract

The standard triumvirate of "New Media" in the mathematics curriculum – CAS, DGS and spreadsheet – is establishing its secured position in teaching. This article illustrates, using the evolution of the software Cinderella as a guiding example, how a further integration of these products will allow for new teaching approaches. These will support the formation of important mathematical competencies, such as the ability to find and rate mathematical models for real world problems. A central observation is the algorithmic nature of user-defined functions and that it is crucial for the learning process to be able to execute algorithms step by step.

1 Introduction

Many teachers today consider integrating the computer into their curriculum, or they already made this step. We are not discussing the reasons for this trend in this article, and neither will we judge this as either good or bad. Instead, we will point out how software interoperability and integration offers new possibilities of media use in the classroom for mathematics education.

During the last 15 years, three major software pillars emerged in that area. Computer Algebra Systems, CAS, constitute the symbolic counterpart to calculators, Interactive (or Dynamic) Geometry Software, DGS, is the modern replacement for ruler and compass, and spreadsheet applications are used for organizing and structuring data for easier handling and analysis, thus also replacing the notion of a variable by the notion of a table cell. All three found their place in teaching, but in our experience many teachers are focusing on only *one* of the three products. It is rarely seen, again in our experience, that teachers have the time and energy to master several different systems and to feel confident enough to use all of them in teaching.

*Part of this work has been supported by the DFG Research Center MATHEON

However, this is less of a problem – given that we appreciate the use of New Media in the classroom – as one might expect. In many cases, a computer-enriched activity can be based on any of the three software types: An exploration can be carried out by dragging points in a sketch, or by evaluating a formula for many different point configurations using a spreadsheet. A functional dependence can be formulated using the language of a CAS, and it can be constructed in a DGS. A stochastic process can be described and visualized in a table, or it can be simulated using a short program in a CAS, etc. Often, we can choose whatever tool is accessible for the students (or better: us), and find a way to use the computer in teaching.

This is supported by the availability of features from other software types in the other ones. For example, even early versions of DGS like Cabri Geometry offer a table feature, where a set of measurements and calculations can be displayed row wise for different states of a construction. In a spreadsheet application data can be visualized in a geometric way, and it is even possible to base a sketch on cell data, by which we can simulate a dynamic move mode. All modern CAS do support graphics, primarily plots of functions in 2D and 3D, but some of them also offer geometric primitives that can be used to build up complex scenes, again based on computed or external data. Modern CAS even offer interactivity, see for example the Wolfram Demonstrations Project at `http://demonstrations.wolfram.com/`. Vice versa, most DGS offer function plots or even allow for the placement of points on curves given by symbolic expressions. There are even some DGS, notably GeoGebra of Hohenwarter [Hohenwarter, 2008], that allow for symbolic manipulations like symbolic derivation, or can find symbolic expressions for the coordinates of dependent elements [Todd, 2006, Todd, 2008].

One might tend to ask whether there is still a need for these different tools, when they all seem to be similarly suited for teaching mathematics. But still, each of them has specific strengths that should be taken into account when one plans to really enhance teaching. It is absolutely necessary to be aware of these strengths when the use of the computer should go beyond a technophile and blind adoption. Here we mention just of few of them, as our main focus will be elsewhere.

Immediate visualization of and hands-on interaction with mathematics is at the core of DGS's powers. If we want to activate as many senses as possible and have parallel stimulations in the learners' mind, then DGS play an important role. The movement of the mouse that is carried out by the student will have much more impact than a mere visualization that is consumed like a TV show.

Structured access to large quantities of data is something spreadsheet applications provide much better than any other mathematical tool. It is paired with a built-in need for discretization and modeling, because the table structure is not suitable for continuous or non-categorized data. Using 2D or 3D graphs of data we can use multi-view capabilities that will activate more regions in the mind, and thus offer the opportunity to have several representations that may suit more students and also let them prescind from the particular representation to discover the mathematical meaning behind it. The last point is not specific to spreadsheets, but applies to all multimodal approaches.

Finally, the symbolic manipulation of formulas in CAS offers a rich field of experimentation that is not surpassed by any of the other mathematical software types. With CAS it is possible to leave the paths that everybody else is going for finding solutions (and questions), at the risk of getting lost completely (fortunately we can get back by interrupting the current calculation!). In [Kortenkamp, 2004] this is discussed in more depth. Other benefits that come with a symbolic (in its literal sense) system are the stepwise mode of working and the formulation of subroutines. A not to be neglected fact is the form factor of CAS: systems that are built in into calculators, for example the TI Voyager or

Casio Classpad series, are robust and can be used spontaneously in the classroom, whereas PC-based software often needs setup and configuration time that is not available.

In the remainder of the article we will describe how the integration of CAS capabilities into a DGS and the introduction of new topics into the school curriculum, leads to an area that was not accessible for the standard tools so far. As we base our observations on the evolution of the interactive geometry software Cinderella.2 [Richter-Gebert and Kortenkamp, 2006], the view might be biased. However, we try to emphasize the universal principles that drove this evolution and that are applicable to other software as well.

Our approach supports any learner-centric and process-oriented educational theoretic frameworks. For example, in conceptual change theory [Posner et al., 1982], which acknowledges that learners do have previous concepts that might not match the concepts that are necessary to understand a given situation correctly, students need a chance to experience the conflicts between their concepts and reality. In the context of supplantation [Salomon, 1994], the combination of an interactive design and computer algebra can help to build mental models (see [Vogel et al., 2007] for an in-depth description of using interactive function graphs). As a last example consider the model of cognitive apprenticeship [Collins et al., 1989], where students need a way to reproduce and comprehend what they are shown (see also [Bescherer et al., 2009]).

We describe in [Kortenkamp, 2007] how the technical aspects mentioned here correspond to the various roles of computer use in mathematics education.

2 Functions and their Graphs in DGS

It is obvious that a geometric system should be able to serve as a function-plotting device. This is the straightforward way of transferring a functional relationship into a geometric context, and many examples that focus on using the computer as a discovery or presentation tool show how useful this feature is (see, for example, [Schumann, 1998]).

For a DGS it is not enough to support the automatic generation of loci, even though this approach might be more geometric to some. This is due to the fact that in schools geometry has a strong focus on its analytic nature, and it is extremely important to offer a bridge between formulas that are written down and their geometric interpretation. *Functional thinking* is one of the most important concepts that were introduced over 100 years ago in the course of the Meraner Reform (see [Klein, 1907, Gutzmer, 1908]). It has been brought to attention again in recent decades [Vollrath, 1989], and it is still topic of modern research [Krüger, 1999, Krüger, 2000], in particular with the use of computers in mind [Roth, 2005, Hoffkamp, 2009]. So, without doubt, a proper handling of functions, both for display (of their graphs) and as parts of constructions, is highly desirable and should be supported by any DGS.[1]

The Interactive Geometry Software Cinderella [Richter-Gebert and Kortenkamp, 1999], a DGS co-developed by one of the authors, was at first a pristine incarnation of coordinate-free Projective Geometry. Measurements are done using Cayley-Klein Geometry via cross-ratios of certain points or lines. A special treatment of constructions in complex projective space made it possible to guarantee a continuous (actually: analytic) behavior of constructions. In this framework, it was nearly impossible

[1] It is also important to see that DGS offer a coordinate-free approach to geometry as well, that should not be concealed too early by the mandatory use of coordinates

to include user-defined functions. A major obstruction was that many functions just are not analytic, like the absolute-value-function abs, or the square root function $\sqrt{\cdot}$. As it is not feasible to omit these, we had to find a solution to include arbitrary functions without loosing the continuity properties of the system as a whole. If it were for display only, this would not be a problem, but if we cannot interact with the function graphs, i.e. place on points on them, we loose the interactivity and immediateness of a DGS.

2.1 2-way Communication between CAS and DGS

The way out of the dilemma described above was found when we tried to setup a two-way communication between the DGS and the CAS Mathematica [Kortenkamp, 2001]. Of course, a general CAS includes the problematic functions mentioned above, so a true communication, where results from either software are used by the other one, would face the same problems we already had with including non-analytic functions. The main idea was to assume that the CAS acts as another user, i.e. elements that depend on calculations in the CAS are moved discretely from one position to the other. The continuity is preserved by interpolating the start and end position linearly, in the same way as it is done with the discrete mouse positions that are reported by the window system when the user manipulates a construction. At the conference *Multimedia Tools for Communicating Mathematics* we were able to present this as a working prototype.

2.2 Functions and Programming

After the main problem was solved, it was only a matter of time to introduce functions into the software. A parser for mathematical expressions had to be written, and the elementary functions had to be provided as built-ins. After some time it became clear, that as soon as it is possible to work with custom definitions, lists, iterators, and conditionals, the function parser will become as powerful as any programming language, and by adding a few convenience methods a versatile functional language (CindyScript) for extensions of the original software was born. A growing set of examples shows the power of this language (see, for example [Fest, 2005, Ladel and Kortenkamp, 2008, Richter-Gebert and Kortenkamp, 2007]). The most important benefit is the complete customizability of interactive learning environments for geometry. Teachers can adjust interactive exercise sheets to their and their students' specific needs.

Let us just point out the didactical value of programming with a minimalist example. In Fig. 1 we see three instances of the same configuration, a triangle ABC. Students can move the vertices A, B, and C and get some feedback about the area of the triangle. They do not get the full information that the software has available (the area of the triangle), but only three different statements: The area is less than 10 cm^2, more than 10 cm^2 or equal to 10 cm^2. This information hiding enables students to focus on something when they drag. They can find out that the "border" between less-than and more-than is a line at a certain distance to the baseline of the triangle. If we show and update automatically the measured area, i.e. if we do not hide that information, students might be distracted and just observe that "everything changes" when they drag a point. This is similar to the information overload that can be observed in certain situations where students work with hands-on material [Kaminski et al., 2006, Kaminski et al., 2008].

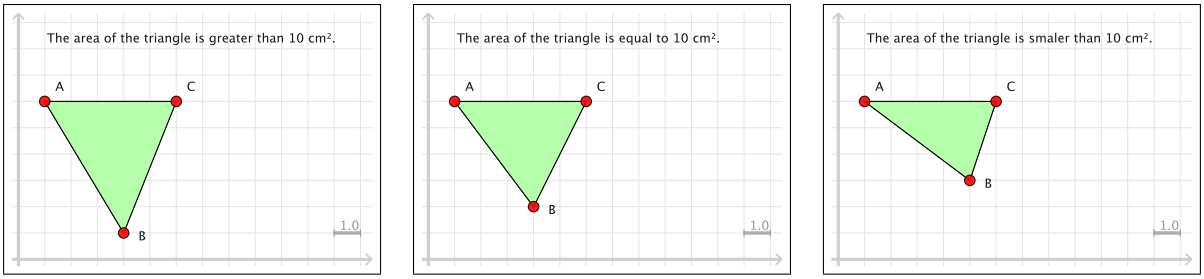

Figure 1: Showing only part of the available information

Remember that this is only a basic building block. There are almost no limits on what can be achieved using this kind of "functions," as the examples of [Fest, 2005] and others show. At the same time, it is also clear from a theoretical point of view, because the function language includes all necessary ingredients to be Turing-complete [Brainerd and Landweber, 1974].

Besides customizing visualizations by the teacher, programming by students can also be used successfully for learning. [Fuchs, 2007] states that functional thinking can be substantiated by dealing with algorithms. He claims that students sense functional implementing to be very efficient for assisting functional thinking. And – of course – for the development of students' algorithmic thinking programming by themselves is essential.

3 Algorithms and Functions

While the function support of Cinderella was also a necessary consequence of the experience of users with other DGS packages, we had, at the same time, the wish to support something else in the software that we could not find elsewhere. Discrete Mathematics, in particular graphs and algorithms on graphs, has always been a gratifying area of mathematics for education; however, most of the time it is seen as an add-on for spare time and not as a central topic that has to be covered. In the DFG Research Center MATHEON we are working on educational material that can be used for a proper integration of topics from Discrete Mathematics into curricula, and at the same time we try to advance the inclusion of this kind of mathematics into the official curricula.

Since graphs can be visualized using points and line segments or arcs, and also many of the real-world problems we want to model with graphs come from a geometric context, we were eager to be able to work with graphs in our software. Also, the algorithms we are using on the graphs should be visualized using the DGS. Both ideas are not new – the famous DGS Cabri started as a tool for graph visualization, and there are numerous software packages for visualizing graph algorithms. It was new that we wanted to combine both aspects in the same software, in order to be able to stick to a well-known user-interface and to preserve the user experience.

3.1 Integrating Algorithms into DGS

We came up with an add-on package called Visage [Geschke et al., 2005] that supports all standard algorithms accessible to 7th to 12th grade students, and more. More information can be found on the

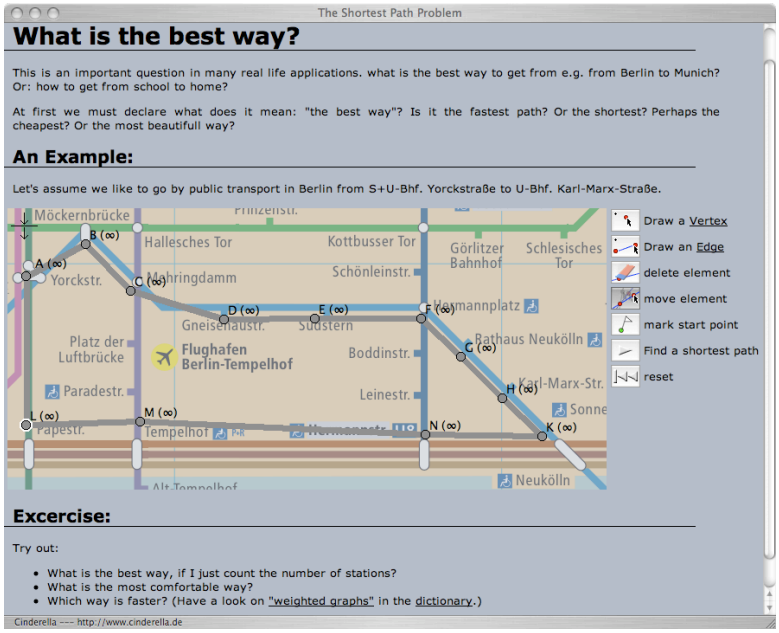

Figure 2: An example worksheet from the Visage package. Students can model a graph directly on a map of the local subway system and use shortest-paths algorithms interactively. Material created by Anne Geschke, TU Berlin.

project homepage at `http://cinderella.de/visage`, where you will also find the software.

Using the software, it is possible to visualize graphs (including a dual-view that shows the adjacency matrix) and to step through graph algorithms (depth and breadth-first search, Dijkstra's algorithm, bipartite matchings, and others). A pseudo-code version of the source can be shown that helps students to trace the algorithm's execution.

All included algorithms have been implemented in Java and strictly followed an imperative programming paradigma. They were hard coded into the software package which means that they are neither editable nor extendable for users, a major drawback that we will discuss below.

The algorithm framework we used for Visage supports threaded execution of the interactive geometry part in parallel with the graph algorithms. This is crucial for the stepping control, where we can decide whether the algorithm is running slowly, as fast as possible, or one statement at a time, triggered by mouse clicks.

There are some subtleties to be aware of, as we are facing a different paradigm when we start dealing with external algorithms. An algorithm might change not only the state of a construction (i.e., move points or change attributes of elements), but it can also change the underlying construction sequence by adding or removing geometric elements. At this point we are facing the fundamental discrepancy between a sequential resp. procedural approach and a pure functional approach.

3.2 Merging Functions and Algorithms

In the previous sections we discussed two topics, the inclusion of functions into a DGS, and the inclusion of algorithms into a DGS. These might appear only loosely related at first sight, but we claim

that they are highly connected: Both are genuine applications of CAS. Working with functions is a core activity with CAS, and the programming languages that are used for CAS and the corresponding command-line interface for it are providing all we need for writing algorithms and executing them step-by-step.

A final evidence for the similar nature of the two is that after some changes in the code abstraction we could easily add another function evaluation language instead of CindyScript, in our case Jython, a Java-based Python clone. Now it is possible to use both CindyScript and Jython as defining languages for functions. As Jython is an object-oriented scripting language that is similar to Java, there is no problem in using it instead of Java for the implementation of the graph algorithms.

This means that we are conceptually very close to a powerful combination of concepts: Once we can use the script language Jython (or CindyScript) for the graph algorithms, we are also be able to allow for user-defined algorithms or alterations to the build-in algorithms.

Back in 2005, our final goal was to provide a flexible environment where one can work with algorithms both as extensions to the interactivity of DGS, i.e. in the function sense that was described in Sec. 2, and in the step-by-step mode that lets algorithms be the subject of research themselves. In an interactive environment any algorithm might ideally be used in both ways without changes in the code.

For example, the calculation of minimum spanning trees for weighted graphs can be solved by some algorithm. This algorithm can also be seen as a function that assigns a corresponding tree to each graph. Figure 3 depicts an exemplary implementation of Prim's MST-Algorithms using the CindyScript programming language. While in the step-by-step mode, for a certain input set the minimum spanning tree is calculated once, in an interactive environment this assignment must be re-evaluated each time the input graph changes. We already demonstrated this with the Java-based version of "external algorithms" [Kortenkamp, 2005], where one could use the same code for a step-by-step demonstration of Boruvka's Minimum Spanning Tree algorithm or a dynamic visualization of the MST. A similar demonstration using the CindyScript programming interface can be found on the Visage project homepage. This approach aditionally offers the possibility of showing multiple algorithms in the same visualization. Users can explore the differences and similarities of different algorithms for the same problem set. For example, we provide a worksheet that lets the user construct an arbitrary graph. On this graph the user can run the depth-first search and the breadth-first search algorithm either stepwise or with the result shown dynamically while the graph changes.

3.3 Automated Algorithm Validation (AAV)

A promising new area of research is the automatic validation of user actions with respect to given algorithms, which we will illustrate with an example below. AAV shareA similar approach is followed in the project SAiL-M (see [Bescherer and Spannagel, 2009] and the project homepage at `http://www.sail-m.de`). Starting October 2008, this project investigates how students' learning processes in learning mathematical processes can be analyzed and aided by automated and semi-automated validations using the computer as an interactive learning environment. First results of SAiL-M relevant to our setting are presented in [Fest and Kortenkamp, 2009].

In our concrete scenario here, the software is not used just for running the algorithm, but also for accounting while the user executes an algorithm manually. For depth-first search, the software had to offer ways for the user to mark vertices, move from the current vertex to a neighbor of it, and maintain

```
//finding cheapest edge that connects a new Node to the tree
firstConnectingEdge(edges, tree):={
   newtreeedge=false;              // no new tree edge found yet

   forall(edges,edge,
      vw=incidentvertices(edge);   // the two incident vertices of the edge
      vwtree = common(tree,vw);    // which vertices belong to the tree?

      // Is the edge connecting a new vertex to the tree?
      if(length(vwtree)==1,
           if(newtreeedge==false, // only use the cheapest (first) edge!
              newtreeedge = edge; // found tree edge
          );
     );
  );
  newtreeedge;    // return the new tree edge
};


// Algorithm of Prim
prim(vertices,edges):={

  treenodes=[vertices_1];     // the first vertex is the starting tree vertex
  treeedges=[];               // the edges belonging to the tree

  edges=sort(alledges(),getweight(#));  // list of all edges sorted by weight

  forall(1..(length(vertices)-1),
                               // we have to choose one edge less than vertices

     newtreeedge=firstConnectingEdge(edges,tree); // find the cheapest new edge
     treeedges = treeedges :> newtreeedge;        // add edge to tree
    treenodes = set(treenodes,incidentvertices(newtreeedge));
                                                  // add vertex to tree
  );

  treeedges;                                  // return tree edges
};    //end of Prim


// execution and coloring
forall(alledges(),#.color=[1,0,0]);         // coloring all edges red
                                            // coloring all tree edges green
forall(prim(allvertices(),alledges()),#.color=[0,1,0]);
```

Figure 3: CindyScript implementation of the algorithm of Prim

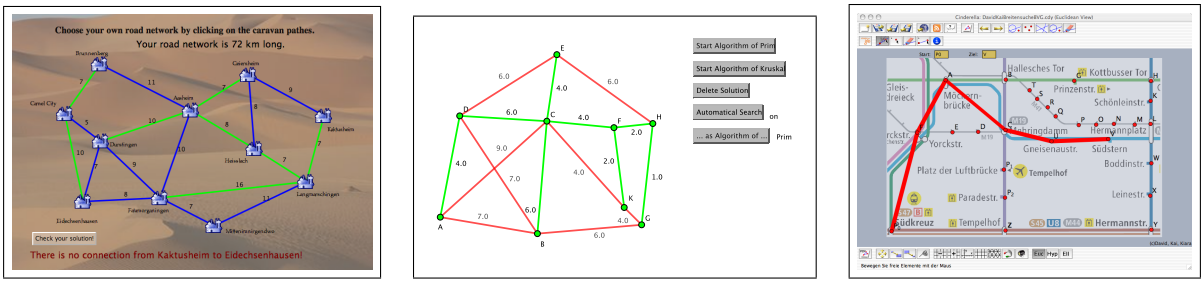

Figure 4: A teaching unit on MST: a) exploring own strategies: the computer judges the solution; b) comparing two textbook algorithms; c) a route planning tool developed by students.

a stack of vertices. Then, the user could execute actions that correspond to a depth-first search, while the computer will check whether the order of actions is indeed a *valid* depth-first traversal. Like in the case of automatic checking of construction sequences (see the auto-checking exercises possible described in [Richter-Gebert and Kortenkamp, 1999]), there is an exponential number of possible execution orders, and the validation cannot be done trivially. Thus, the software has to be able to compare the users' actions with statements like "with every neighbor of the current vertex do . . . end". The didactical contribution of such a package would be that we have an open learning environment, which still provides a feedback mechanism for unattended and independent work of the students.

4 Exemplary Implementation and Results

The new Berlin state curriculum for secondary schools [LISUM, 2006] contains a section on Minimum Spanning Trees, which we implemented using the Visage software. The unit was carried out with two groups of students at the age of 13 to 17 years. A more detailed description of this class room experiment is given in [Fest, 2008].

4.1 AAV for Student's Solution Processes

We use automated algorithm validation as a student's tool for the development of own algorithmic solutions. The main educational objectives are that the students embrace the structure of the problem and can use usual textbook algorithms to solve the problem. The first goal is reached by modeling the problem and finding own solutions, the second by investigation and implementation of the algorithms.

First, the students should develop their own solutions for the problem of connecting all cities of an imaginary country with roads with the cheapest connections possible. The students were asked to reflect on their own solution strategies. After some unstructured trials using paper and pencils the students were given an electronic worksheet to support their thoughts. In this worksheet the student can mark or unmark possible roads by simple mouse clicks. They get immediate feedback, the length of the chosen network, for each solution they try.

Additionally, the computer can *judge* the solution in three steps. First, using the depth-first search algorithm, the computer checks wether the student's solution does connect all cities. Next, the same algorithm is used to check the absence of cycles in the solution, which is a necessary condition for the

optimality of a solution. Finally, if a student has constructed a tree solution, the optimality is checked using Kruskal's MST algorithm.

The whole worksheet – handling of user interactions and algorithmic checks – was created using the script programming interface. Our classroom experience showed that the students were faster in developing good solution strategies when they used the electronic worksheet as their attention is more focused on the right aspects. The students who used the worksheet also had fewer problems in verbalizing their strategies afterwards.

After finding own solution strategies the students were asked to explore the standard textbook algorithms by Kruskal and by Prim. Therefore, another worksheet similar to the BFS/DFS example described in section 3.2 was implemented. In this worksheet both MST algorithms can be executed step-by-step on arbitrary graphs.

4.2 Students' Programming

The students also had the possibility to implement Kruskal's algorithm on their own using the CindyScript programming interface. The programming language had to be learned on the fly; as an aid the students had an example implementation of Prim's algorithm. Although *none* of the students had *any* programming experience before, some of them succeeded implementing the algorithm within only two and a half hours. As the implementation of an algorithm requires a mental restructuring of the student's knowledge, programming Kruskal's algorithm helped the students to deepen their understanding of the algorithms.

By dint of the easy-to-learn programming language and its good integration into the DGS even the younger students were able to create own interactive worksheets. This also animates the students to vary the problem settings. In the following lessons some students of 8th grade created an electronic worksheet on the Traveling Salesman Problem on their own. They had to understand the Double-Tree-Approximation of the TSP for this task. Another group of older students engaged themselves in the development of an online route planning tool for the Berlin subway net.

All in all, this first experiment showed the viability of our approach, and asks for a proper investigation and empirical research in future studies. In particular, we would like to know how automated algorithm verification can help to foster conceptualization in mathematics. First steps of this analysis take place in the project SAiL-M (see Sec. 3.3). The technology presented here is one of the necessary tools to carry out such studies.

5 Future Directions and Acknowledgements

We want to summarize the three main conclusions from the above:

- Functions are intrinsically algorithmic. The main tools for working with functions are CAS, and their symbolic user interface is presenting a (mathematical) programming language. When we want to include functions into any other software, we are very close to opening a door to algorithmic explorations.
- A proper experimentation tool for algorithms, which can foster the understanding of mathematical models and how they are used to find solutions to real world problems, has to provide

mechanisms for a stepwise and user-controlled execution, and an interactive mode for an immediate hands-on user experience.

- The presence of a simple programming tool strongly connected to a mathematical visualization tool opens new ways to investigate in functional and algorithmic dependencies. Using it, students seem to be motivated to explore problems and questions which they introduce on their own.

In the near future we will revise and publish Cinderella's Java API which supports the algorithm framework mentioned in Sec. 3. A prototype of the API was already successfully approved in a course on graphs and network algorithms at the Technical University of Berlin in summer term 2008. The new API allows for almost full user access to the interactive geometry kernel and the threading control of the algorithms. The published source code of exemplary built-in algorithms allows a deeper exploration of the algorithms by the students. The use of the API seems to be especially suited in university teaching.

On the educational side, based on our first experiences in using our tools, more empirical studies should be done within precise theoretical frameworks. We expect that, though the tools are applicable in various settings, the results will differ depending on other circumstances caused by the corresponding theory.

We would like to thank Jürgen Richter-Gebert for his enormous amount of programming for Cinderella.2, Anne Geschke for her work on the Visage material, and Dirk Materlik for his invaluable help in debugging and extending this piece of software.

References

[Bescherer and Spannagel, 2009] Bescherer, C. and Spannagel, C. (2009). Learning mathematics in technology enhanced scenarios - SAiL-M. In *Proceedings of ICTMT 9*, Metz, France.

[Bescherer et al., 2009] Bescherer, C., Spannagel, C., Kortenkamp, U., and Müller, W. (2009). Intelligent computer-aided assessment in mathematics classrooms. In McDougall, A., Murnane, J., Jones, A., and Reynolds, N., editors, *Issues in Research on IT in Education*. Routledge.

[Brainerd and Landweber, 1974] Brainerd, W. S. and Landweber, L. H. (1974). *Theory of Computation*. Wiley, New York.

[Collins et al., 1989] Collins, A., Brown, J. S., and Newman, S. E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In Glaser, R. and Resnick, L. B., editors, *Knowing, learning, and instruction*, chapter 14. Lawrence Erlbaum Associates.

[Fest, 2005] Fest, A. (2005). Several examples for using CindyScript in teaching. Online Resources at `http://www.math.tu-berlin.de/~fest/DATA/`.

[Fest, 2008] Fest, A. (2008). Using the cinderella/visage framework for studying graph algorithms. In *Proceedings of CERME 6*, Lyon, France.

[Fest and Kortenkamp, 2009] Fest, A. and Kortenkamp, U. (2009). Process oriented learning environments for interactive geometry lessons. In *Proceedings of ICTMT 9*, Metz, France.

[Fuchs, 2007] Fuchs, K. J. (2007). Functional thinking: a fundamental idea in teaching computer algebra systems. In *Informatics, Mathematics and ICT: a 'golden triangle'.*, Boston.

[Geschke et al., 2005] Geschke, A., Kortenkamp, U., Lutz-Westphal, B., and Materlik, D. (2005). Visage – visualization of algorithms in discrete mathematics. *Zentralblatt für Didaktik der Mathematik*, 37(5):395–401.

[Gutzmer, 1908] Gutzmer, A. (1908). *Die Tätigkeit der Unterrichtskommission der Gesellschaft Deutscher Naturforscher und Ärzte*. Teubner, Leipzig.

[Hoffkamp, 2009] Hoffkamp, A. (2009). Enhancing functional thinking using the computer for representational transfer. In *Proceedings of CERME 6*, Lyon, France.

[Hohenwarter, 2008] Hohenwarter, M. (2008). GeoGebra. `http://www.geogebra.org`.

[Kaminski et al., 2006] Kaminski, J. A., Sloutsky, V. M., and Heckler, A. F. (2006). Do children need concrete instantiations to learn an abstract concept? In *Proceedings of the XXVIII Annual Conference of the Cognitive Science Society*, pages 1167–1172, Mahwah, NJ. Erlbaum.

[Kaminski et al., 2008] Kaminski, J. A., Sloutsky, V. M., and Heckler, A. F. (2008). The advantage of learning abstract examples in learning math. *Science*, 320:454–455.

[Klein, 1907] Klein, F. (1907). *Vorträge über den mathematischen Unterricht an den höheren Schulen*. Teubner, Leipzig.

[Kortenkamp, 2001] Kortenkamp, U. (2001). The future of mathematical software. In *Proceedings of MTCM 2000*. Springer-Verlag.

[Kortenkamp, 2004] Kortenkamp, U. (2004). Experimental mathematics and proofs – what is secure mathematical knowledge? *Zentralblatt für Didaktik der Mathematik*, 36(2):61–66.

[Kortenkamp, 2005] Kortenkamp, U. (2005). Dokumentation, Diskussion und Protokolle: Wie kommuniziert man Geometrie im Internetzeitalter? In Engel, J., Vogel, R., and Wessolowski, S., editors, *Strukturieren – Modellieren – Kommunizieren. Leitbilder mathematischer und informatischer Aktivitäten*. Franzbecker, Hildesheim.

[Kortenkamp, 2007] Kortenkamp, U. (2007). Guidelines for using computers creatively in mathematics education. In Ko, K. H. and Arganbright, D., editors, *Enhancing University Mathematics: Proceedings of the First KAIST International Symposium on Teaching*, volume 14 of *CBMS Issues in Mathematics Education*, pages 129–138. AMS.

[Krüger, 1999] Krüger, K. (1999). *Erziehung zum funktionalen Denken. Zur Begriffsgeschichte eines didaktischen Prinzips*. Logos, Berlin.

[Krüger, 2000] Krüger, K. (2000). Kinematisch-funktionales Denken als Ziel des höheren Mathematikunterrichts – das Scheitern der Meraner Reform. *Mathematische Semesterberichte*, 47(2):221–241.

[Ladel and Kortenkamp, 2008] Ladel, S. and Kortenkamp, U. (2008). Verdoppeln, Halbieren und Zerlegen. Online at `http://kortenkamps.net/material/doppelmoppel`.

[LISUM, 2006] LISUM (2006). Rahmenlehrplan Mathematik für die Sekundarstufe I. Technical report, Senatsverwaltung Bildung, Jugend, Sport.

[Posner et al., 1982] Posner, G., Strike, K., Hewson, P., and Gertzog, W. (1982). Accomodation of a scientific conception: toward a theory of conceptual change. *Science Education*, 66:211–227.

[Richter-Gebert and Kortenkamp, 2006] Richter-Gebert, J. and Kortenkamp, U. (2006). *Cinderella.2 – Math in Motion*. Springer-Verlag. Software, available at `http://cinderella.de`, manual to appear.

[Richter-Gebert and Kortenkamp, 2007] Richter-Gebert, J. and Kortenkamp, U. (2007). Cinderella blog. Online at `http://blog.cinderella.de`.

[Richter-Gebert and Kortenkamp, 1999] Richter-Gebert, J. and Kortenkamp, U. H. (1999). *The Interactive Geometry Software Cinderella*. Springer-Verlag, Heidelberg.

[Roth, 2005] Roth, J. (2005). *Bewegliches Denken im Mathematikunterricht*. Franzbecker, Hildesheim.

[Salomon, 1994] Salomon, G. (1994). *Interaction of media, cognition, and learning: an exploration of how symbolic forms cultivate mental skills and affect knowledge acquisition*. Routledge.

[Schumann, 1998] Schumann, H. (1998). Dynamische Behandlung elementarer Funktionen mittels Cabri Géomètre II. *MNU*, 3/1998.

[Todd, 2006] Todd, P. (2006). Geometry expressions: A constraint based interactive symbolic geometry system. *Computeralgebra-Rundbrief*, 39.

[Todd, 2008] Todd, P. (2008). Geometry expressions v1.1. `http://www.geometryexpressions.com`.

[Vogel et al., 2007] Vogel, M., Girwidz, R., and Engel, J. (2007). Supplantation of mental operations on graphs. *Computers & Education*, 49(4):1287–1298.

[Vollrath, 1989] Vollrath, H.-J. (1989). Funktionales Denken. *Journal für Mathematikdidaktik*, 10:3–37.