Learning Algorithms with an Electronic Chalkboard over the Web

Margarita Esponda Argüero and Raúl Rojas

Institut für Informatik, Freie Universität Berlin, Takustr. 9, 14195 Berlin
{esponda,rojas}@inf.fu-berlin.de

Abstract. This paper describes a system for the animation of algorithms on an electronic chalkboard. The instructor teaching an algorithm can enter data directly through a drawing – the algorithm then makes use of this data, for example numbers, or the image of a graph. The drawing becomes alive. The result is a more natural way of teaching and starting algorithmic animations. The paper also describes how to couple a sign and handwriting recognition engine with the animation system. The lecturer can then write programs using her own handwriting, and the programs runs. All animations can be enriched with sound and explanations from the lecturer and can be posted to the Web.

1 Motivation

Algorithmic animations are becoming popular for teaching computer science [1]. There are now many visualizations of algorithmic animations available on-line. Most are used by students, as part of a course, and have been developed by university staff.

We have recently developed an algorithmic animation system called Chalk Animator as an extension for the E-Chalk system. Conventional systems for algorithmic animation handle only computer generated images as building blocks in an animation. Perfect rectangles, circles, arrows, etc. are drawn by the user using a graphical editor or are generated automatically by the system. In this paper we consider a more radical alternative: the generation of algorithmic animations starting from sketches drawn by the user on an electronic blackboard, which is both the presentation tool and the user interface for the lecturer. Not only is this approach time-effective for the lecturer, but also the "look and feel" of animated sketches is very different from computer generated graphics. Sketch animation resembles best the kind of teaching done using a traditional chalkboard. The animations. Once completed, can be see and heard through the Web.

2 The e-Chalk System

The main idea of the E-Chalk system is to provide the functionality of the traditional chalkboard using a large contact sensitive computer screen, but enhanced with all the capabilities of a digital system [7,2,3,4,5]. An electronic blackboard should be as easy to use as a traditional one. The only interface to the electronic board should be a stylus, instead of a piece of chalk.

When an E-Chalk session is started, the server computer starts storing and sending three streams: the board events, the audio channel, and an optional video channel. The three streams can be accessed from a Web page, by starting the E-Chalk client, which is a collection of three client Applets, one for each stream. The streams are synchronized by the audio time stamp. Therefore, an E-Chalk session can be recorded completely, but the quality of the reproduction of the board on the client side is much higher, since the board is repainted with the full resolution of the computer screen. Fig. 1 below shows schematically a teaching scenario: a lecturer teaches to a live audience; the E-Chalk server transmits the audio, video and board streams and stores an archival copy. A remote viewer watches the class using an Internet browser.

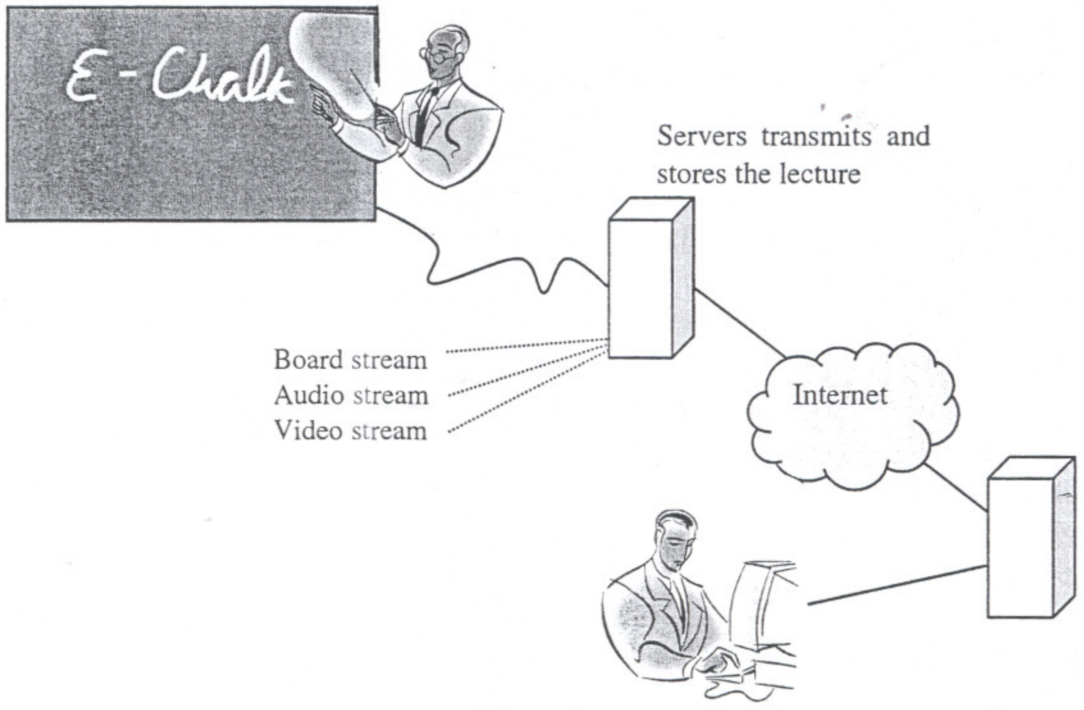


Fig. 1. The E-Chalk system. A lecturer writes on an electronic blackboard. Audio, video, and board contents are stored and are streamed through the Internet. Remote viewers watch a lecture using a Java enabled browser.

Fig. 2 is a screen dump of an actual lecture, as seen by a remote viewer in his browser. The look and feel of the screen is that of a lecture on a good blackboard. The use of color helps to emphasize some important aspects of the lecture.

The main feature of E-Chalk is to go beyond the original blackboard metaphor and provide "intelligence and information on demand". This means, that a series of special programs is running in parallel with E-Chalk and is watching the user interacting with the screen. Certain programs can then become active when certain conditions are met. E.g., a program can observe the handwriting of the user and if a mathematical formula is entered, and if the user writes a special stop symbol, the program can interpret the formula. If it is an equation, it can solve it. This capability has been already implemented in E-Chalk, using Mathematica as the mathematical equation solver [8,9,10].

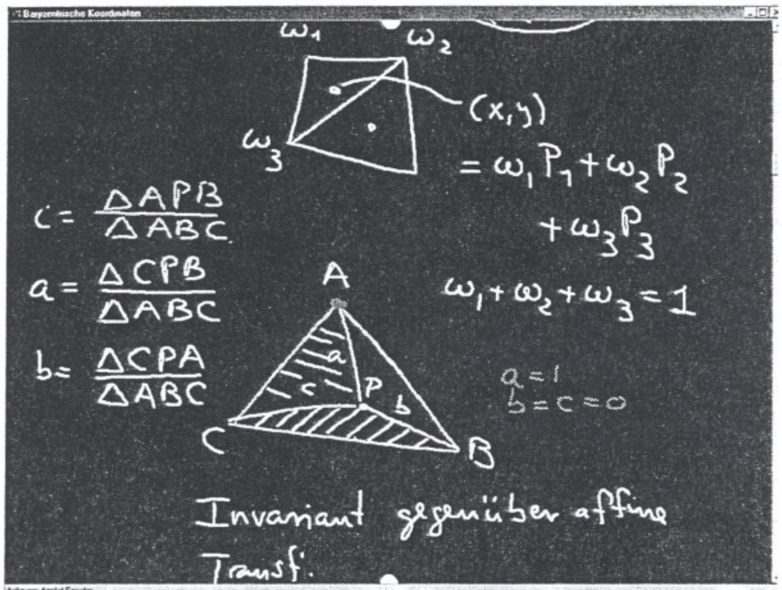

ω_1
ω_2
ω_3
(x,y)
$= \omega_1 P_1 + \omega_2 P_2$
$+ \omega_3 P_3$
$\omega_1 + \omega_2 + \omega_3 = 1$
$c = \frac{\Delta APB}{\Delta ABC}$
$a = \frac{\Delta CPB}{\Delta ABC}$
$b = \frac{\Delta CPA}{\Delta ABC}$
A
B
C
P
a
b
c
$a = 1$
$b = c = 0$
Invariant gegenüber affine
Transf.

Fig. 2. A real E-Chalk lecture about geometric concepts.

Regarding algorithmic animations there are two things which come immediately to mind as possible extensions of the blackboard metaphor: a) the immediate execution of code written on the blackboard, b) the animation of algorithms started by the lecturer.

3 Executing a Programming Language in e-Chalk

In this section we describe a further educational innovation implemented for the E-Chalk system as. A simple interpreter for the programming language BASIC, was written, and the interpreter was coupled with the handwriting recognition machinery of E-Chalk. The lecturer can now write a BASIC program on the chalkboard and request its immediate execution. This implementation provides a glimpse of what will become possible in the future.

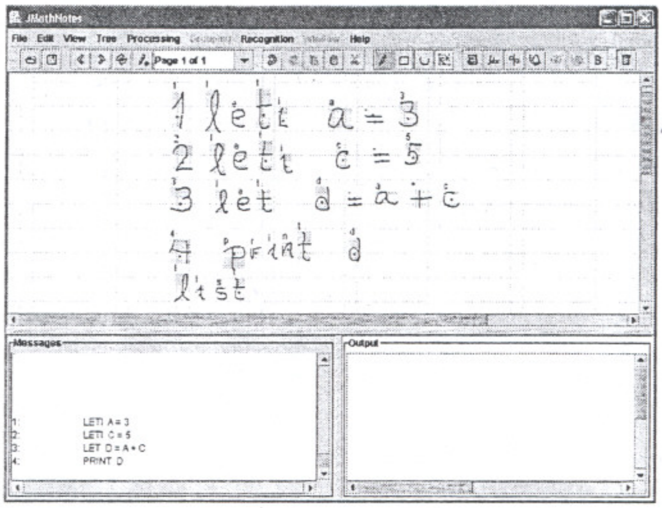
A handwriting recognizer is needed, if code written on the electronic blackboard is to be executed. The E-Chalk handwriting recognizer is a pattern recognition system developed by Ernesto Tapia at the FU Berlin [8].

Symbols are recognized by processing line strokes and extracting relevant features. Such features are, for example, the length of the line stroke, its centroid (in a unitary square), the distance between start and end of the stroke, divided by the total length. Also, the coordinates of a few points along the stroke can be added to the feature vector. The most relevant points for the shape are selected using a shape simplification algorithm. Once a symbol has been transformed into a feature vector, it is given to a neural network or support vector machine, which has been trained previously to recognize this symbol.

As a proof of concept for a programming system based on handwriting recognition, we defined a minimum subset of BASIC, which is nevertheless general purpose and universal. Our own version of BASIC is called *Tiniest BASIC* and consists of only two interpreter commands and six types of instructions.

The two instructions "RUN" and "LIST" can also be entered. RUN starts the program at the first line of code. LIST provides a listing of the current code lines. Non-trivial programs can be written with this language.

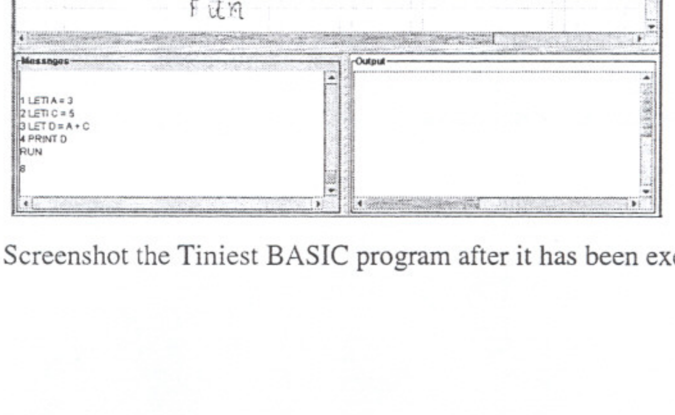
Fig. 3 shows JMATH, the program developed by Ernesto Tapia to train the character recognizer [10], coupled with our BASIC interpreter. A Tiniest Basic program has been entered. Each character is surrounded by a box and an identifier of the character which has been recognized. After writing the "LIST" command, the user closes the input by pressing on the B button (BASIC). The window to the lower left shows the output of the Tiniest BASIC interpreter.



```
1:          LETI A = 3
2:          LETI C = 5
3:          LET D = A + C
4:          PRINT D
```

Fig. 3. Screenshot of JMATH, the editor and training program for formula recognition developed by E. Tapia, coupled to the BASIC interpreter.

Fig. 4 shows the same program, but now the command "RUN" has been entered. The output of the program is visible in the lower left window, it is the constant 8.



```
1 LETI A = 3
2 LETI C = 5
3 LET D = A + C
4 PRINT D
RUN
8
```

Fig. 4. Screenshot the Tiniest BASIC program after it has been executed.

Many of the options of the JMATH editor will be available in future releases of E-Chalk. Symbols can be erased, for example, by scribbling rapidly on them. Symbols can be moved from their position. If a gap is needed between two lines, the symbols below the first line are selected and moved down. This gesture recognition could allow to edit and annotate programs written in more complex programming languages, as explored in the next section.

4 Interactive Computer Driven Animation of Sketches

In E-Chalk, there is a special color which can be set at the beginning of a session. Strokes drawn with this color are processed by the handwriting recognizer engine. The engine receives all the strokes (as sequences of lines), processes the strokes and gives the result of the recognition to another application (in the case of mathematical formulas, to Mathematica from Wolfram Research). The application processes the input and gives back an ASCII string to E-Chalk or a picture in GIF or JPEG format. This application output (a number, a graph, a drawing, etc.) is pasted to the blackboard.

For our purposes this is not enough. An animation produces multiple frames, which have to be pasted at the same position. If only the history of an algorithm is being drawn, then the handwriting interface is all that is needed, but the output can only be a static image.

There is a way to generate an animation for E-Chalk using its macro recording capability and the handwriting recognizer. In E-Chalk, stroke input drawn with a previously chosen color is always passed to the shape recognition engine. The shape recognition engine groups strokes according to proximity or overlap (some digits, for example, consist of more than one stroke) and recognizes the shape among a library of symbols. This information can be passed to an algorithm animator, not to Mathematica. Our application is an animation engine for E-Chalk, which has loaded an algorithm written in Java. The Java algorithm receives the input from the shape recognition engine, runs the algorithm, and produces an animation script. The animation script is processed as a stream by the animation script interpreter, which in turn produces the code for a macro. After entering the input, the user can then go to the macro menu and start the new macro, which will be played as directed by the animated algorithm. The diagram of Fig. 5 shows schematically the information flow in the E-Chalk system. All steps are transparent for the user, who only has to call the appropriate macro at the end.

The shape recognition engine generates an object library. This is needed, because if we want to reuse the shapes entered by the user, the shapes must be stored by the recognition engine. They are assigned an object number, which can then be used by the animation algorithm generating the script.

5 Examples

After the object library has been defined by the user (by entering its input), the animation algorithm uses this objects in the animation. The instrumented algorithm produces an animation script, which after processing yields a macro file for E-Chalk. The animation runs on the electronic blackboard. Fig. 6 shows the start of the animation and the input written by the user (upper left picture). The upper right side of Fig. 6 shows the progression of the algorithm. The pivot for the comparison is painted red, the other number being compared is painted pink. After a number has reached its final position it is painted green, with a ticker line.

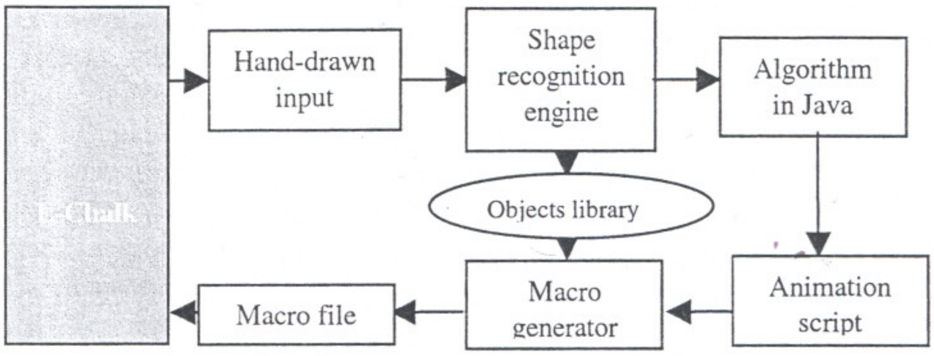


Fig. 5. Information flow for the animation of sketched input.

This experiment is also interesting, since multiple visual cues have been used to reinforce the idea of magnitude. The size of the digits is proportional to their value. The value itself is read by the user. Sorted numbers are shown in green, a color which automatically recalls the association of something being right or correct. The action is taking place at the place where the pivot is, which is painted red to signify work. The position of the two numbers being compared is also marked by two horizontal bars which slide across the array.

The animations produced in this manner can be easily collected as macros and can be replayed by an instructor during class.

In the example above, the separation of the strokes was made by considering only connected objects made of one single stroke sweep. This property is easy to test when given the coordinates of the strokes points.

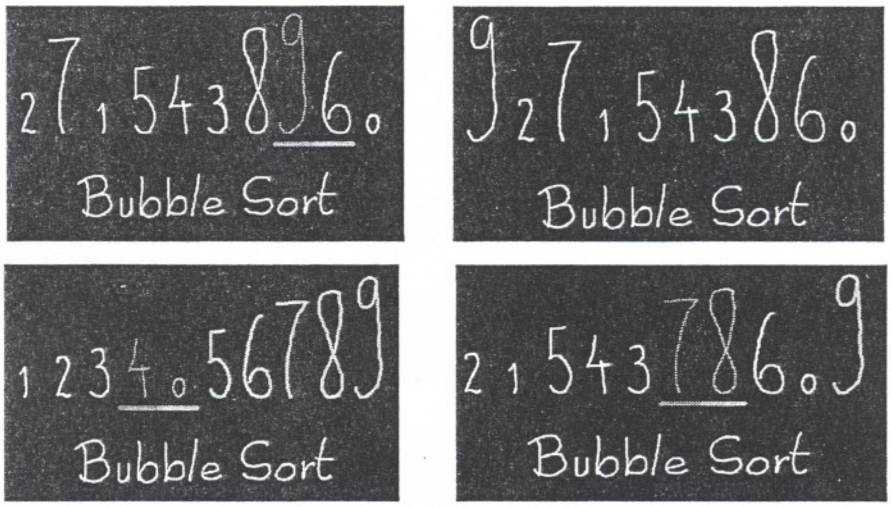


Fig. 6. The bubble sort algorithm running, animated with user input.

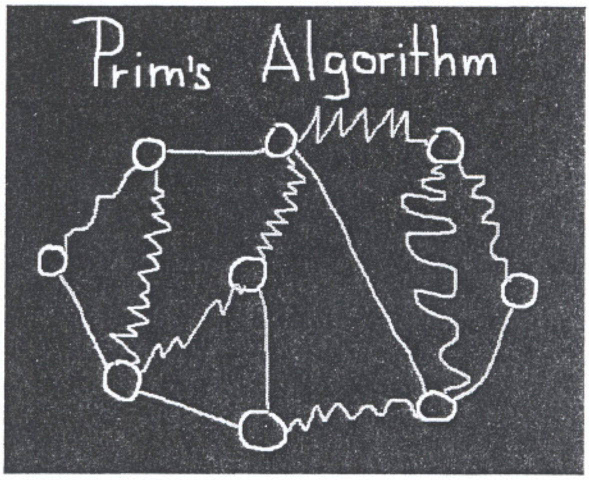


Fig. 7. The user enters the input. Nodes are painted white, edges are green.

The next example is a more spectacular illustration of the kind of animations which become possible once the user is empowered to enter her input using a pen computer. A program which implements the popular Prim algorithm for the computation of a minimum spanning tree was written. A spanning tree is a subset of the graph's edges which touches all nodes of the graph without producing cycles. The minimum spanning tree has minimum total weight (each edge has an associated weight). Fig. 7 shows how the user enters her input: by drawing the nodes and edges of the graph. In this case the extractor program was coded to expect nodes as white elements, edges as green elements. It would have been possible to detect automatically which elements are edges and which circles, but in this case we settled for a simple extractor. The weight of an edge is its total length. The reader can imagine that these are cities connected by roads of different length.

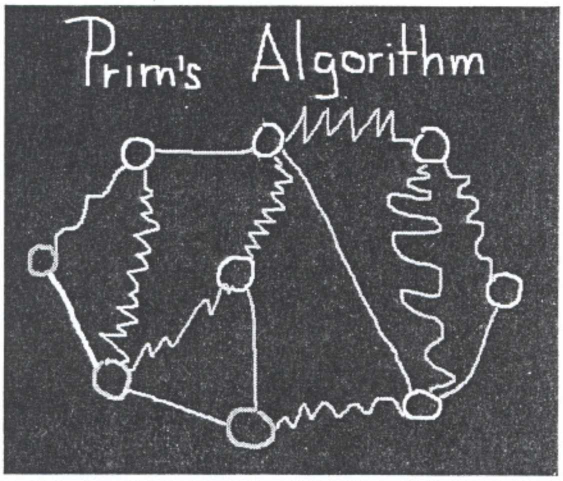


Fig. 8. An edge has been added to the spanning tree (yellow edge). The next node to be visited is painted pink (lower middle).

Fig. 8 shows the further progress of the animation. Nodes in pink have been selected and yellow edges already belong to the spanning tree. The next edge selected is the shortest edge touching the pink nodes. Fig. 9 shows the animation some frames later. The edge to the middle right is being painted yellow after being selected.

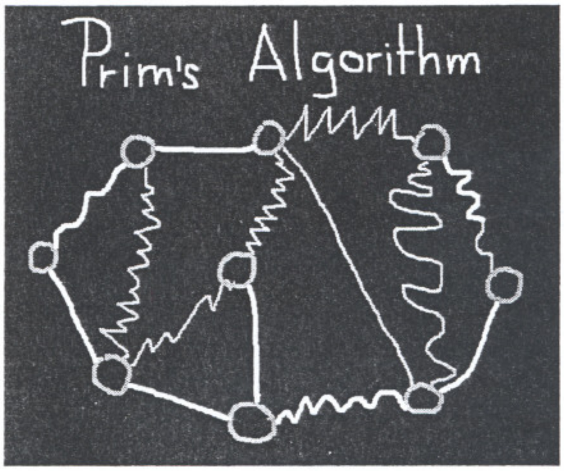


Fig. 9. The last edge is being painted yellow.

Fig. 10 shows the end result. All nodes have been visited and the "road map" is the shortest possible tree connecting all cities.

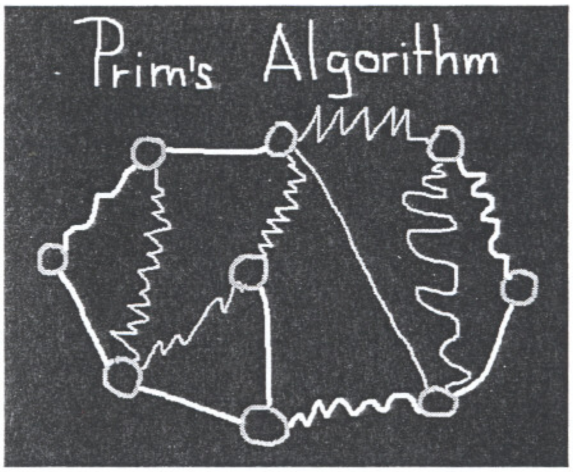


Fig. 10. Final minimum spanning tree.

The more impressive results from the animation are obtained when the user changes the input and the animation runs automatically. Fig. 11 shows four screen-shots of the same algorithm running on a new graph sketched by the user.
In the case of this animation the technique of representing the weight of an edge by its length gives an intuitive feeling for the correctness of the decisions taken at each step. Some algorithmic animation systems represent the weight of an edge sometimes by the width of the edge, but this is the first attempt to represent the weight using a wig-gled line. It is also clear why: other algorithmic animation systems render graphs using lines and circles. It is very difficult to draw esthetically appealing graphs using this convention (length of edge equals weight), but not when the user can sketch on a

blackboard. Here we find a definitive advantage of the blackboard compared to simple graph drawing programs. The animation produced by the Chalk animator looks almost like an abstract painting.

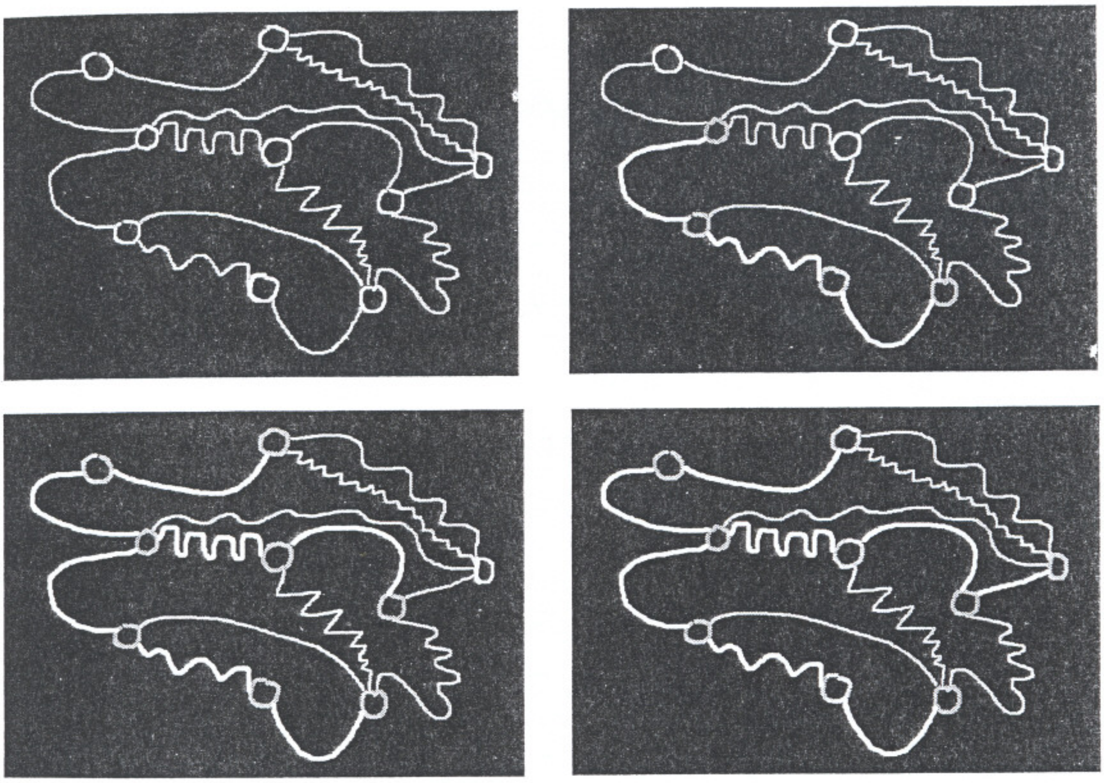


Fig. 11. Prim's algorithm working on another graph (from upper left to lower right).

6 Summary and Discussion

The kind of animations possible with our system has never been tried before: in this paper we have shown how to produce animations directly, using the input possibilities of pen computing. An instructor in a classroom needs a more natural interface with the teaching instrument, i.e. the blackboard, and this is what we have investigated in this paper. Although sketches are used to illustrate algorithms or books, few experiments have been conducted towards coupling sketches and algorithmic animations, the exception being the "low fidelity" algorithms of Hundhausen [6].

Given an electronic blackboard, it is natural to ask if the algorithms could be directly written on the blackboard and could be interpreted by the computer. We have called this "handwriting programming". Our experiment with the Tiniest BASIC interpreter shows that is indeed possible to couple a handwriting recognizer with an interpreter, in order to provide this functionality. The Tiniest BASIC interpreter was written in a few hours. Future work will be oriented towards recognizing pseudocode languages, such as Python.

The second half of the paper deals with E-Chalk and how to implement a two way communication channel between the user and the algorithm to be animated. The E-

Chalk API was modified with this objective, but the same functionality can be obtained through the use of macros. We showed that a user can enter his or her input as a macro, which is then analyzed by a symbol recognizer. The symbol recognizer builds a library of symbols. An animation written in our scripting language, can then access these objects and use them during the animation. We showed, that this method can be extended even to the labels of the animation, which can be produced by text synthesis, that is, by reproducing the handwriting of the algorithm animator. This preserves the sketch form of the diagrams, even when they contain text.

References

1. Brown M., *Algorithm Animation,* MIT Press, Cambridge MA, 1988.
2. Friedland G., Knipping L. and Rojas R., "E-Chalk Technical Description," Technical Report B-02-11, Faculty of Computer Science, Freie Universität Berlin, May 2002.
3. Friedland G., Knipping L., Rojas R. and E. Tapia, "Das E-Chalk System: Stand der Entwicklung," Technical Report B-03-03, Department of Mathematics and Computer Science, Freie Universität Berlin, February 2003.
4. Friedland G., Knipping L., Tapia E. and R. Rojas, "Web Based Education as a Result of AI Supported Classroom Teaching", *Proceedings of the Seventh Conference on Knowledge-Based Intelligent Information & Engineering Systems* (KES), Oxford, England, September 3-5, 2003. To appear in: Lecture Notes in Artificial Intelligence (LNAI) Vol. 2774, Springer-Verlag, Berlin Heidelberg.
5. Friedland G., Knipping L., Rojas R. and C. Zick, "Mapping the Classroom into the Web: Case Studies from several Institutions", *Proceedings of the 12th EDEN Annual Conference*, Rhodos, June 2003.
6. Hundhausen C., Douglas S., "A Language and System for Constructing and Presenting Low Fidelity Algorithm Visualizations", *Proceedings of Software Visualization 2001*, Dagstuhl, Germany, May 20-25, 2001, pp. 227–240.
7. Rojas R., Knipping L., Raffel U. and Friedland G, "Elektronische Kreide: Eine Java-Multimedia-Tafel für den Präsenz und Fernunterreicht", *Informatik: Forschung und Entwicklun,* Vol. 16, No. 3, pp. 159–168.
8. Tapia E. and Rojas R., "Recognition of Handwritten Digits in the E-Chalk System using Support Vector Machines," Technical Report B02-14, Department of Mathematics and Computer Science, Freie Universität Berlin, Oktober 2002.
9. Tapia E. and Rojas R., "Recognition of On-Line Handwritten Mathematical Expressions using a Minimum Spanning Tree Construction and Symbol Dominance," *Proceedings of the Fifth IAPR International Workshop on Graphics Recognition* (GREC), Barcelona, Spain, July 30-31, 2003. To appear in *Lecture Notes in Computer Science*, Special Issue on Graphics Recognition, Springer-Verlag, 2003.
10. Tapia E. and Rojas R., "Recognition of On-line Handwritten Mathematical Formulas in the E-Chalk System," *Proceedings of the Seventh International Conference on Document Analysis and Recognition* (ICDAR), Edinburgh, Scotland, August 3-6, 2003.