Teaching a Mouse How to Draw

David L. Maulsby
Ian H. Witten

Knowledge Sciences Laboratory, Department of Computer Science
The University of Calgary, 2500 University Drive NW
Calgary, Canada T2N 1N4

Email: maulsby@UCalgary.CA, ian@UCalgary.CA

Abstract

An apprenticeship metaphor is described for programming by example, graphically, which enables untrained end users to add composite operations to a drawing program using constructive methods traditionally employed in drafting. To combat the extraneous activity that plagues naturally-produced traces, interaction is used to constrain induction by suppressing, or at least controlling, variability. A device called "Metamouse" serves to concentrate the user's attention on the system's limited inferential capabilities. It predicts actions, asks for constructions, solicits input parameters when required, and induces a program. Implications for machine learning include the benefits of simulating a pupil to complete the teaching metaphor, and the role of user interaction in constraining the search for apt generalizations. Implications for computer graphics include the feasibility of teaching an interactive editor to perform repetitive graphical procedures and the use of familiar drawing primitives as a basis for more complex transformations.

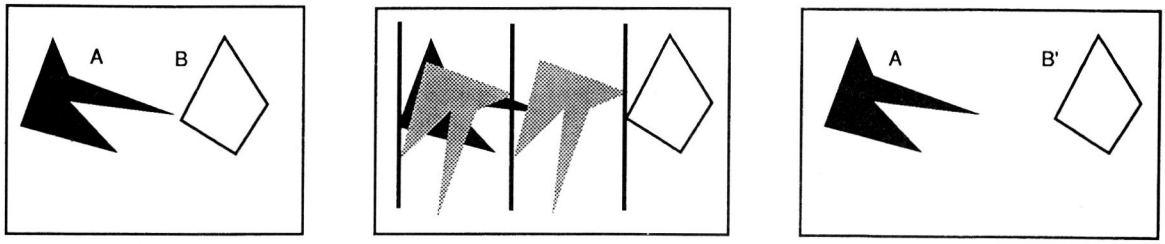
KEYWORDS: graphical programming, machine learning, user interface, graphical construction

I. The Drawing Domain

Interactive graphics provide excellent examples of human-computer cooperation. Here we find human and computer sharing the work of constructing what amounts to a program whose output is a hard-copy drawing. Popular drawing programs (eg. MacDraw [Cutter 87]) augment the software analog of drafting tools with extensive editing capabilities and the rudiments of positional constraint. These facilities enable casual users to master some of the mechanical skills of draftsmen and to concentrate more upon design. Moreover, they allow operations to be composed into re-usable structures. Many elements of a user's style are just such composites, carefully constructed, even parameterized. If executed frequently or with difficulty they are well worth adding to the system. Clearly, it is highly desirable that the user himself be able to augment the drawing system, but we cannot expect him to write abstract specifications. Since he already executes his own programs in the "concrete" language of the user-interface, it suffices that the computer translate such programs into machine-executable form, with appropriate operators, constants and variables. The result is a system for programming by example, graphically.

Figures 1-5 illustrate several tasks to be programmed. The input picture is transformed into the output picture by constructive operations having both ad hoc and derived parameters. For example, consider the "box-to-line" task shown in Figure 4. Given a set of boxes, the artist aligns their lower right-hand corners to an axis (guide-line). To ensure that translation is strictly horizontal, the artist employs a horizontal sweep-line. The "box-to-line" procedure is to set up the task environment (draw the guide-line and sweep-line, select the boxes), then iterate over the transformation (translate each box along the sweep-line until its lower right corner touches the guide-line), and finish by dismantling the task environment (remove the guide- and sweep-lines).

We propose a graphical system with a metaphorical pupil and minimal use of symbolic abstractions. [Halbert 84] and [Myers 86a] review graphical programming and programming by example. Classic systems such as Pygmalion [Smith 75],



a. input: two polygons

b. construction: measure height of A by duplicating and rotating 90°; translate B

c. output: A and B' (=B transformed)

Figure 1. Graphical task: translate polygon B such that distance from left extreme of A to left extreme of B is 2 x height of A.

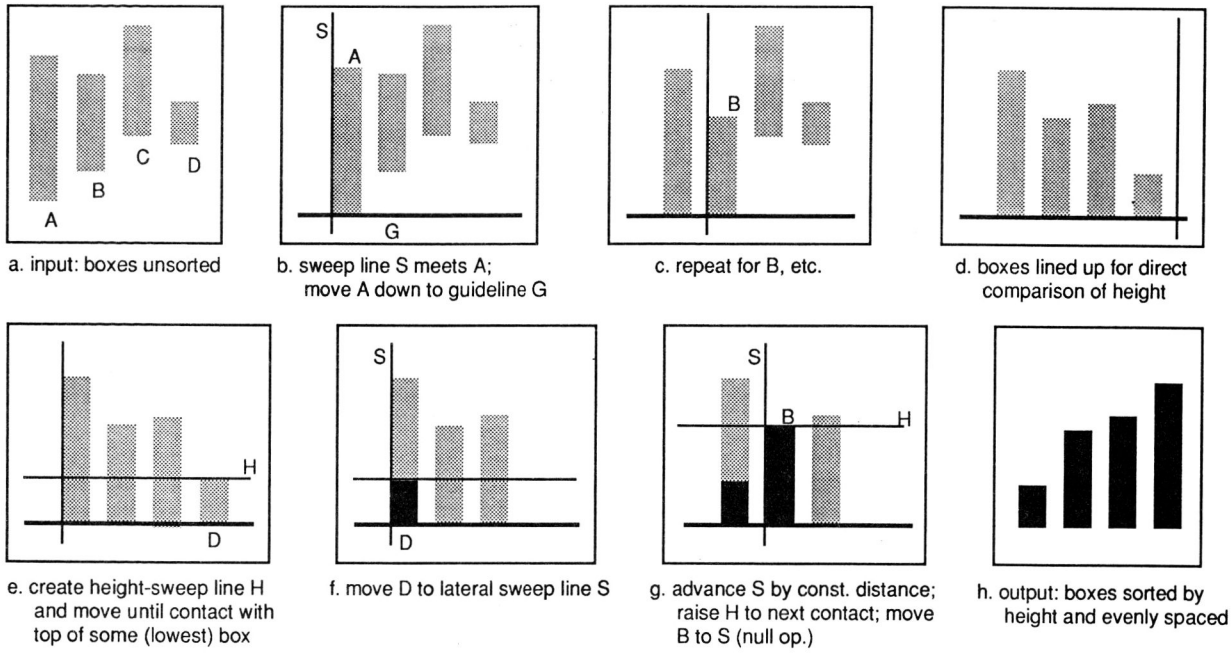


Figure 2. Task: given a set of boxes, sort them in order of increasing height.

ThingLab [Borning 86], and SmallStar [Halbert 84] use iconic representations that are nonetheless essentially abstract; hence the programmer must learn how to interpret the conventions of the programming interface. Andreae's robot programming scheme [Andreae 85] infers loops and conditional branches from traces, which must be carefully ordered by the teacher.

Since an induction system capable of inferring an arbitrary program from input and output pictures is practically unattainable [Angluin 83], we focus on the constructive methods traditionally employed in drafting and whose performance by users we have studied. Our learning system induces binary relations between elements of constructions, and extends established techniques of action matching and generalization [Andreae 85]. Because of the user / programmer's enormous computational advantage in the graphical domain (consider the relative ease with which a person can induce a convex hull function), we require that certain conditions of a teaching protocol [Van Lehn 83] govern the user's input. To do so without asking the programmer to undergo special training, we make the system participate very actively in its own instruction. We represent this enthusiastic learner by a multi-function cursor called "Metamouse". [Myers 86b] describes a graphical method of programming responses to mouse events in spatial context by manipulating a mouse icon. Metamouse is similar, but the goal is to program the application rather than the user interface. Moreover, Metamouse has a more general representation of graphical knowledge (spatial relations, constraints and procedures). Metamouse is reactive: it predicts the user's actions, asks for and assists in building constructions, and calls for input parameters as needed. We describe a particular Metamouse for a simple drafting domain — a graphical turtle called "Basil".

II. Box-to-Line: A Worked Example

Consider the "box-to-line" procedure as taught by example (Figure 4). The teaching process consists of leading Basil through a trace of the task. Basil, or, to be precise, the learning system, expects the parameters of actions to be constant, input by the user at run-time, or constrained by events (such as a corner coming into contact with the guide-line). The teacher begins by selecting the boxes; the learning system classifies these as the input set. When she places the guide-line's two end-points (Figure 4c), Basil observes the absence of a contact constraint, classifies the event as underconstrained and interrupts to ask (through a dialogue box, see Figure 8) whether the location is constant or a run-time input. The teacher indicates

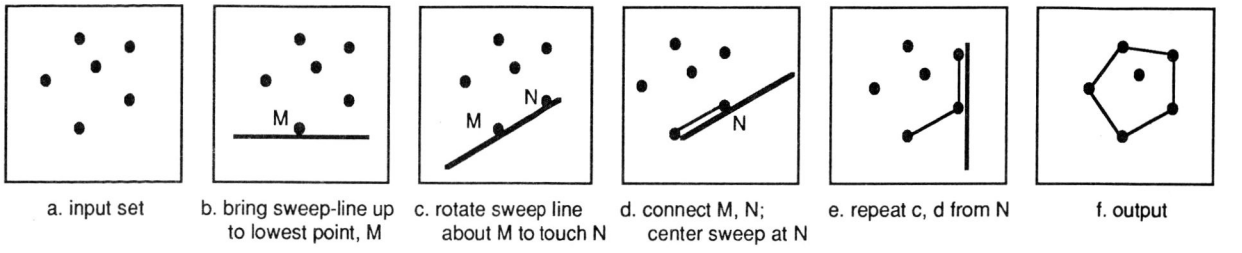


Figure 3. Task: given a set of points, find their convex hull.

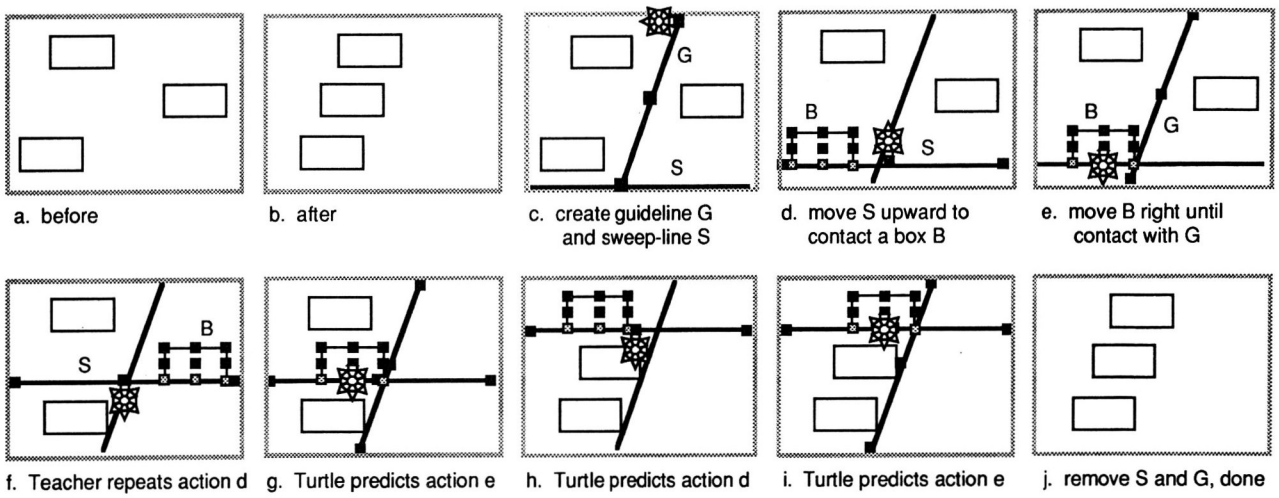


Figure 4. Teaching trace of box-to-line procedure.

that both points are to be specified at run-time.

The teacher then leads Basil through the main iterative sequence (Figure 4d-i). It is easy for Basil to observe that the objects transformed are members of the input set, and that iteration terminates when every member has been processed. In general, however, selection and iteration depend on any number of properties of objects or situations. Thus iteration should be ordered and conditioned on events that Basil can sense by touch. A horizontal sweep-line serves this purpose, and also constrains the boxes' path of translation. The teacher draws the sweep-line near the bottom of the screen and indicates (through a dialogue box) that its initial placement is constant. She then grabs the sweep-line at its mid-point handle and moves it upwards until it touches the bottom edge of some box (Figure 4d).

Observe that this program achieves a series of constraints expressed by visible contacts. A suitable description, say "lower left corner of box in grasp is coincident with some point on guide-line," is invariant over iteration on the input set. The learning system distinguishes this contact event and induces its invariance.

When the sweep-line touches the first box, the teacher grasps and moves it rightward until its lower right corner touches the guide-line, its bottom edge still on the sweep-line (Figure 4e). The teacher then re-grasps the sweep-line and proceeds to the next box (Figure 4f). When Basil sees her select this second box, he notes that the action patently repeats that of selecting the first box. Consequently he conjectures a loop and predicts the translation that is to follow (Figure 4g). Up to this point Basil has been following the teacher like a studious apprentice; now he will attempt to take the lead. The second box, however, must be moved to the left. Basil is biased towards easily generalizing directions of movement, so this does not faze him; he leaps into action and moves the box on his own. The teacher does not object; Basil has now learned the body of the loop and operates on the next box by himself (Figure 4h-i).

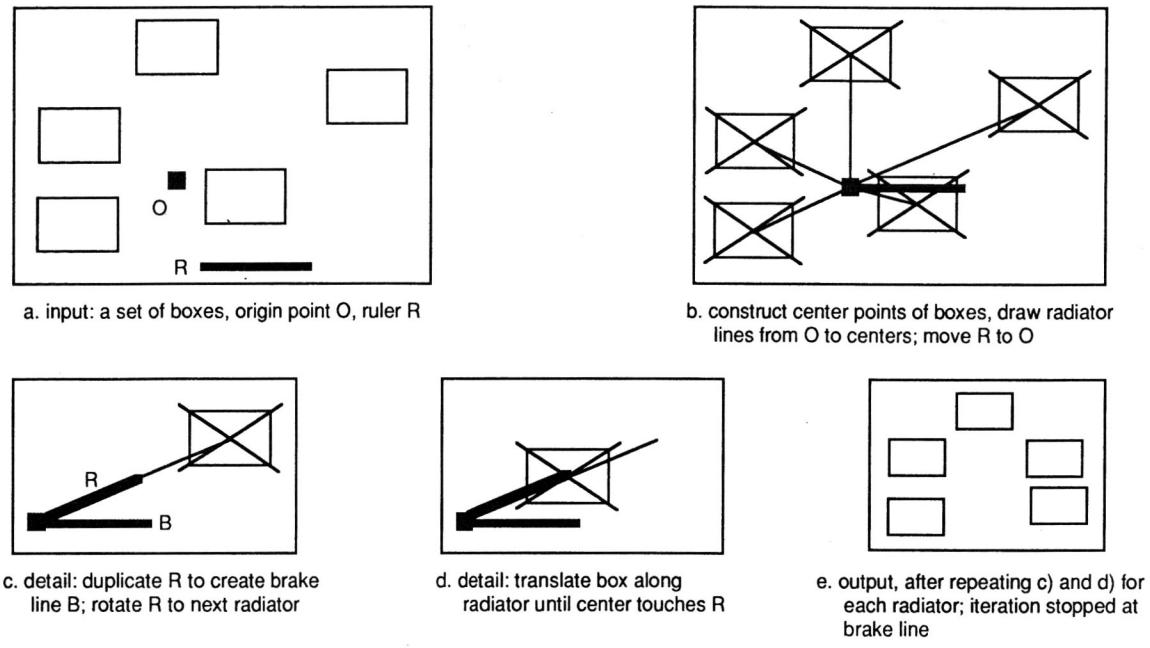
After processing the third and final box, Basil recognizes that he cannot complete the action of moving the sweep-line as he has learned it. Hence he terminates the loop on the condition of being unable to perform its first step, and calls upon the teacher to demonstrate what to do. At this point, she removes the sweep- and guide-lines (Figure 4j), and then announces that the lesson is over.

III. Metamouse as Mediator

We asked a group of people to perform 7 or 8 graphical tasks in 1 hour using MacDraw. Their actions were recorded with a commercial programming-by-example system [Tempo 86]. We studied their use of constructive methods either on their own initiative or with prompting and assistance. Subjects familiar with MacDraw were more likely to use constructions without prompting. Novices who were coached began using constructions on their own initiative after 3 or 4 tasks, but required advice throughout. With one notable exception, novices who were not coached used few if any visual constructions, but instead performed measurements by eye. As a result of this study we conclude that traditional constructive methods, recently and widely proposed for use by non-specialists working with graphics editors (see [Bier 86], [Fuller 86], [White 88], [Noma 88]), are not natural to most users, although they are suitable as a means of performance optimization for those with the required experience or geometric insight.

We also observed an alarming amount of extraneous activity, which could be classified as: 1) missteps (actions quickly retracted); 2) experiments (action sequences retracted, often performed on duplicates of input data); 3) bustle (useless actions due to confusion or to pass time while thinking); and 4) changes in method (actions not retracted). Equally impressive were the variety of algorithms used and the tendencies to abandon one algorithm for another, to switch back and forth without returning to a pristine initial state, and to interleave work on different parts of the problem. Of course, some of this hyper-activity is surely due to the experimental situation; the tasks were unfamiliar, and subjects had to invent algorithms on the spot.

The Metamouse interface to our learning system helps to suppress such extraneous activity in several ways. First, Basil — as an icon — distinguishes programming from other activities, while maintaining context (cursor location). Second, the user can put Basil to sleep at any time in order to experiment unobserved. Third, Basil embodies the teaching metaphor [MacDonald 87] and the learning system's model of procedures and constraints, in the form of touch-sensitive a turtle like those used in Turtle Graphics [Papert 81]; this metaphor is readily



a. input: a set of boxes, origin point O, ruler R

b. construct center points of boxes, draw radiator lines from O to centers; move R to O

c. detail: duplicate R to create brake line B; rotate R to next radiator

d. detail: translate box along radiator until center touches R

e. output, after repeating c) and d) for each radiator; iteration stopped at brake line

Figure 5. Task: translate a group of objects such that all lie at equal distance from a reference point.

understood and helps the teacher meet the felicity conditions elaborated below. Fourth, Basil generalizes coordinate positions as contacts with named parts of objects and hence can be used to precisely manipulate construction tools under programmer control.

Basil enforces the felicity conditions that render a rather naive learning system instructible. Felicity conditions [Van Lehn 83] are conventions of behavior that facilitate communication in a particular social situation. Van Lehn describes four that apply to our system.

"Show work" obliges the teacher to present sample executions of a procedure, rather than ask the pupil to induce it from inputs and outputs. This motivates our use of constructive methods. Basil is a graphical robot that records actions; he does not analyze static pictures. He performs a construction as a sequence of drawing operations, each step beginning and ending with a tactile event or ad hoc input from the user.

"Invisible objects" (eg. the portal of an arch) must be indicated, so that the pupil can distinguish them. Measuring in graphics (distance, alignment, etc.) is often done by visual inspection. Basil inhabits Flatland [Abbott 1884], where touch is more useful than vision. Basil requires tactile events to bound actions, thereby enforcing the use of visible constructors, such as a ruler line to measure off distance. If no constructor is possible, as when placing the guide-line in the "box-to-line" procedure (see Figure 4), Basil confirms that the action is to be performed by the user at run-time (see Figure 8).

Basil generalizes actions by relaxing the tactile constraints that govern them. For example, "any part of A touching any part of B" is a generalization of "leftmost vertex of A touching any part of B".

The "show work" and "no invisible objects" conditions actually help the teacher construct correct examples. Consider the Star problem (Figure 5). The teacher first places the origin point on the display. Basil queries and the teacher replies that this is a parameter. She marks the centers of the objects to be moved, using a built-in operator or a constructive technique, then draws radiator lines from these center points to the origin. She creates a measuring line (ruler), another input parameter, and places one of its end-points at the origin. She rotates the ruler until it coincides with a radiator, then translates the attached object along the radiator until its center point lies at the ruler's end. The teacher repeats this until the ruler returns to its original orientation (marked by a copy of itself).

From this execution trace the learning system need only induce the regularities that govern the movement of ruler and objects, and detect iteration by sequence matching. Now, suppose we eliminate the ruler. The learning system must discover the invariability of distance from the origin point along a radiator to an object's center after it is moved. This requires function induction, which in general is very difficult [Andreae 85]. But the teacher's effort also increases; she must judge the distance by eye. Obviously, the chance of error grows. Now suppose we eliminate the radiators as well. This need not increase the difficulty of function induction for the learning system, but the teacher will have considerably greater trouble judging distances.

"One disjunct" means that any lesson introduces at most one branch of a program. Since our pupil has perfect memory, we propose a "one method" felicity condition, banning irrelevant activity and permutation on the order of actions. Basil keeps his teacher on track by predicting the next action whenever possible, using a multiple-context prediction technique similar to that of PURR-PUSS [Andreae 77]. If the teacher rejects a prediction, a disjunct (conditional branch) is implied.

"Correctness" is an obvious felicity condition but the most difficult to enforce. Graphical traces are full of noise produced by vagaries in the movement of the user's hand. Basil filters out minor positioning errors by a variant of the familiar "gravity" function. Basil is drawn towards the most specific tactile event

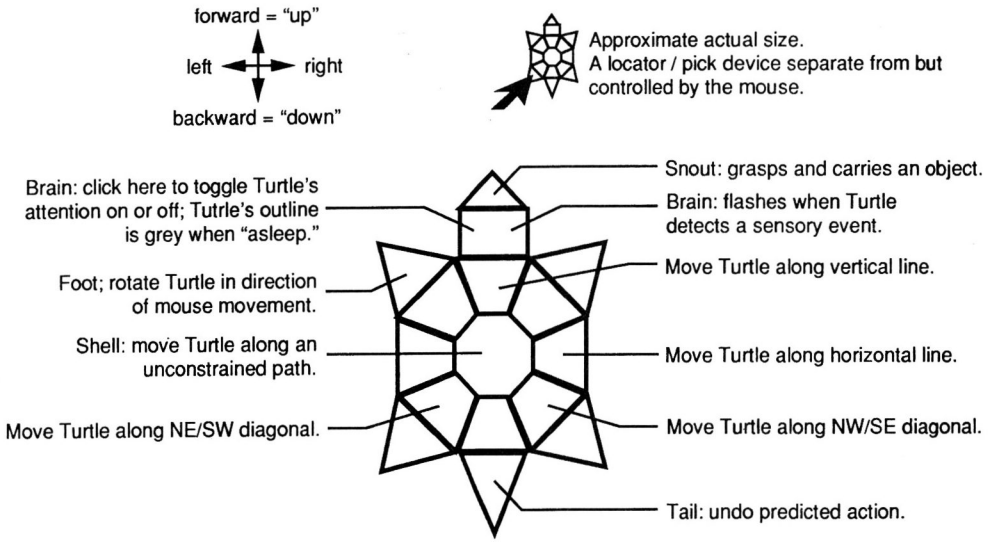


Figure 6. Anatomy of Metamouse, a moveable menu that sometimes runs away.

within a certain distance. Larger errors in measurement are filtered out by the use of constructions. Potential errors in a trace are filtered out by prediction. Errors in the algorithm are not dealt with.

IV. Anatomy and Psychology of a Metamouse

Our prototype Metamouse programming system has for its domain a drawing program called A.Square which, although very simple, is nonetheless capable of representing problems such as positioning labels, maintaining connectivity, evenly distributing objects across a window, etc. The drawing program has two primitive objects, box and line, and two transforms, scaling and translation. Its user interface resembles MacDraw.

Basil is designed to permit two modes of instruction: teaching by demonstration and teaching by leading. In the former the teacher manipulates objects herself and Basil follows (see Section II). In the latter, the teacher manipulates Basil who in turn draws, grasps and moves objects at her direction. A design for Basil is illustrated in Figure 6. With respect to tradition and squeamishness, the turtle was chosen over other short-sighted creatures such as mice and spiders. Note that it adds rotation and constrained-path translation to the operators in A.Square; these help the user communicate her intentions (consider the difficulty of rotating a line, without also scaling it, by translating one of its end points) — in leading mode. Thus Metamouse augments the representational power of A.Square so that the user can teach the system procedures involving rotation, such as the Star transform or finding a convex hull. To date, only teaching by demonstration has been implemented. Rotation is not yet supported. Path constraints on translation are induced from observations.

Basil has the following initial knowledge: 1) there are two types of primitive object, box and line; 2) an object has distinguished components, namely its handles and the line segments between them; 3) picking (grabbing) an object activates its handles; 4) grabbing a handle (other than the center) scales a box and rotates and scales a line; 5) grabbing an object's center or mid-point handle translates it; 6) a spatial relation between two objects can be described with reference to the touching of their components, or to lines that connect them; 7) the display window is enclosed by an invisible box object. Users of drawing programs understand most of this already. Items 2, 6, and 7 may require explanation but have strong intuitive appeal.

As Basil follows the teacher through a task, he records his perceptions. These comprise: 1) the magnitude and direction of his motion; 2) tactile events (begin touching and cease touching) occurring at his snout or at any component of an object in his grasp; 3) types of objects touched; 4) location of all of their components; 5) spatial relations established by tactile events. Our hierarchy of spatial relations may be compared to that given by [Geller 87] and is illustrated in Figure 7.

Basil's working memory contains: 1) his own current position and orientation; 2) the sensory feedback described above; 3) the steps of the current trace, stored as (precondition, action, postcondition) triples, where conditions are derived from 1 and 2 above; 4) the program he is executing and learning, represented as a directed graph of step tuples, each of which may have several predecessors and successors; 5) the individual identity of objects created or transformed during the trace.

As the teacher demonstrates the algorithm, Basil predicts actions from the program, and observes and generalizes the tactile pre- and postconditions that govern actions (see Figures 7 and 8). He also predicts the need to create a particular constructor object and knows whether it should be parameterized by the user. When predicting actions, Basil must instantiate object and position variables. This is done using a simple constraint solver that generates a translation vector to be applied to Basil and any handle he is grasping [Maulsby 89b]. If the postconditions fail to hold after applying the vector, alternative vectors are attempted.

V. Dialog with Basil

A teaching session is one execution of the procedure on sample data. The program elements to be learned are the inputs, outputs, constants, variables, operators, conditional branches and loops. This section describes how these are induced during a demonstration of the Box-to-Line task shown in Figure 4 and

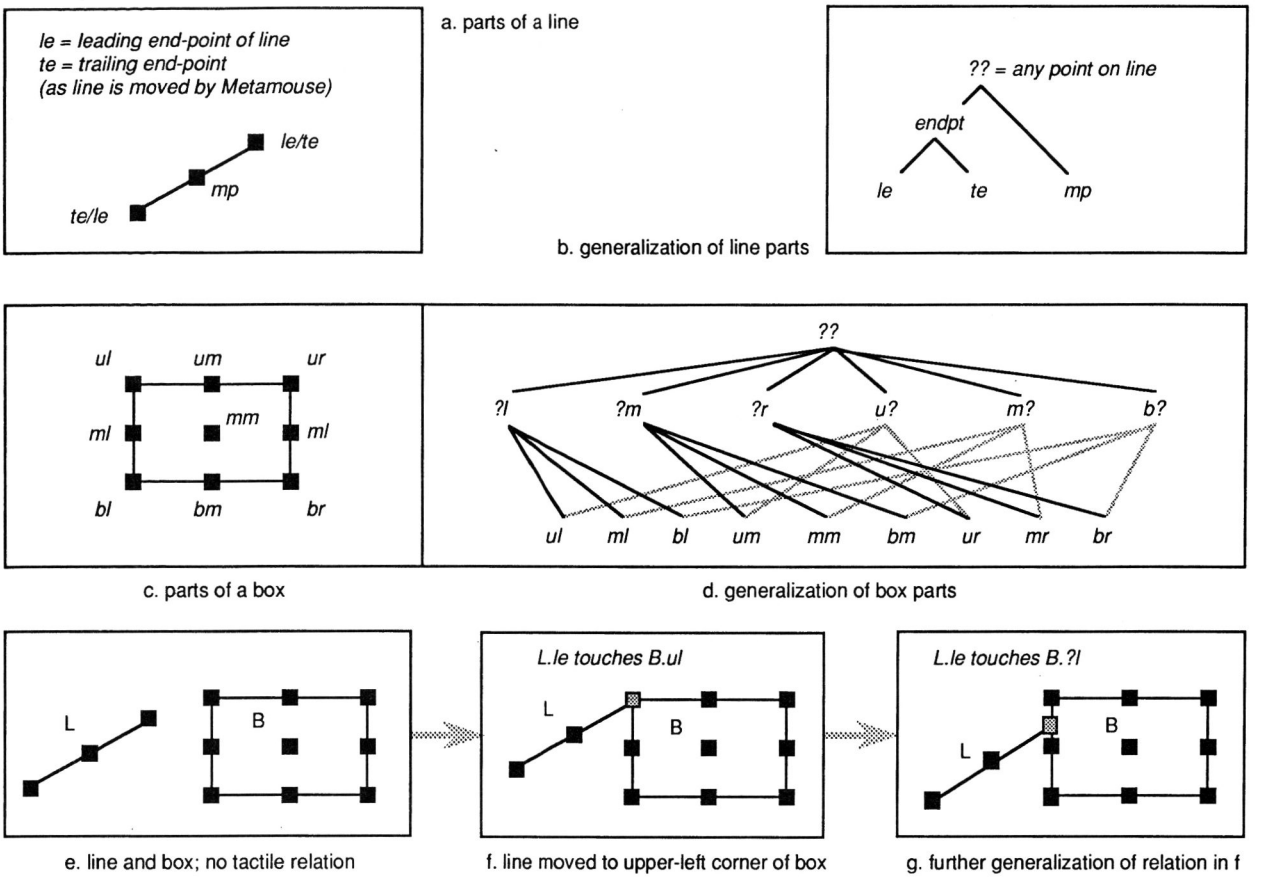


Figure 7. Tactile relations and their generalizations.

described in Section II.

Inputs are those objects already on the display when execution begins, and those objects created during the course of execution with the assistance of the end user. When the position or size of a constructed object is not constrained by tactile events, Basil asks the teacher for the parameter(s) required, as illustrated in Figure 8.

Outputs are those objects remaining on the display after execution. Construction tools, eg. the sweep- and guide-lines, are removed in the course of the training trace; Basil distinguishes these objects and will remove them at run-time.

Constants are constraints on position attained or distance moved, classified by the teacher in reply to Basil's questions. The end-points of the sweep-line are positional constants. **Variables** refer to objects in touch relations, eg. "the line currently grasped" or "the box whose bottom edge was touched by the guide-line in the previous step." The learning system replaces object addresses with variables when it generalizes trace actions into program steps. It searches back through the trace for previous occurrences of the object to ensure that the same variable is used. A variable is typed according to object class (line, box). A variable is instantiated by a drawing operation or by the constraint solver, which finds an object satisfying all the currently-sensed touch relations in which the variable occurs.

Operators are Metamouse and A.Square commands. In the prototype, the only commands are those to create boxes and lines and move their handles. Instead of explicit operands, an action has pre- and postconditions; the constraint solver derives the operands. Conditions are tactile constraints or numeric inputs or constants. Normally, the postcondition of one action is the precondition of the next, but they should be distinguished since their terms of generalization may differ. Only pre- and postconditions are generalized, not the actions themselves. When Basil predicts an action but terminates it incorrectly, the teacher rejects the prediction and then demonstrates the operation herself. Basil generalizes the postcondition to cover the action. Preconditions are generalized when Basil is unable to predict the next action but the teacher insists upon it. At present, we use minimal generalization. We recognize that more promiscuous generalization heuristics might make Basil learn faster, at the cost of having to specialize when creating conditional branches.

Conditional branches are taught in separate training sessions or separate iterations of a loop. Recall that each program step may have a number of successors (ie. branches) following it. The precondition must match Basil's current sensory feedback, the postcondition must be attainable by solving constraints, and the teacher must accept the predicted action. If no option succeeds, a new branch is learned. As the system observes the teacher's actions it tries to match them with steps already in the program to close the branch. In order to test the connection, Basil predicts subsequent actions; if the teacher assents, the connection is confirmed. Technical issues such as ordering the options are discussed elsewhere [Maulsby 88].

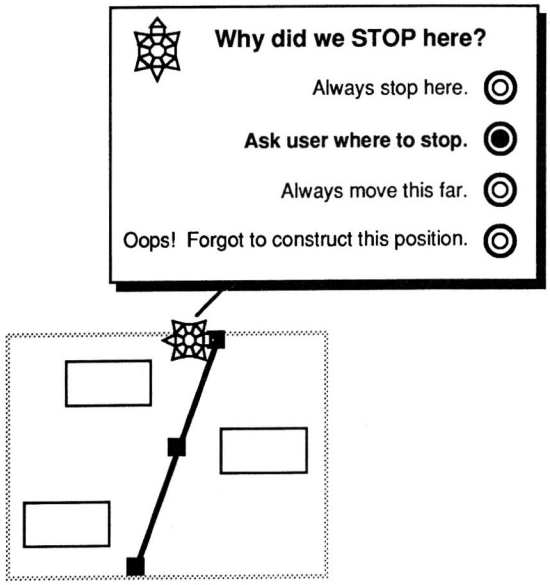


Figure 8. Asking Teacher to explain a seemingly arbitrary decision. Dialog for start of line is similar.

Loops are sequences ending with a jump back to the head. Basil always reviews his most recent actions to match sequences that predict a loop. In the Box-to-line task trace, he observes repetition of the action "move to grasp sweep-line's mid-point". When the teacher accepts his subsequent predictions, the loop is confirmed.

VI. Results and Conclusions

A pilot implementation has been tested on a number of action traces garnered from users of A.Square. Learning performance data for three tasks are given in Table 1 . "Colonnade" is the sub-task of the sort procedure (Figure 2) that distributes boxes at equal spacing along a line. "Connectivity" re-connects the edges of a polygon when one of them is moved. "Box-to-line" is as described in this paper. Basil learned each procedure incrementally from one or more traces. The table shows the total number of steps in each trace, the number predicted correctly by Basil, and the ratio of steps performed by Basil to the total. The complexity of the programs constructed was measured as the number of edges in their state graphs. The data gathered for each task indicate that Basil learns quickly: in these tests he was able to predict all actions from the second trace onwards, except when a variant on Connectivity was introduced. The program graphs were as small as those designed by the researchers.

These results are of course preliminary. We intend to gather more traces and try more complex tasks. The effectiveness of eager prediction to filter out errors, variability and coincidences was tested by comparing programs learned with and without this technique. The results, reported elsewhere, indicate that eager prediction is extremely useful [Maulsby 89a].

The design of Basil has been strongly influenced by the results of our experimental study of user behavior in constructive graphical tasks. We have not completed a proper evaluation of how people interact with the new drawing tool. Nevertheless, we can already draw some conclusions from the work.

First, user interaction can augment or replace domain knowledge in constraining the massive searches incurred by function induction without requiring that the user manipulate or even understand the internal representation. The three-way trade-off between search space, ease of teaching, and built-in knowledge can be readily investigated in the graphical domain. Second, an eager learner can reduce teacher noise and enforce felicity conditions. Basil's actions must be clearly indicated, and it is of course vital that a convenient "undo" facility be provided to control his impetuosity. Third, an appropriate metaphor can quickly and forcefully convey the learning system's limitations.

Finally, we have demonstrated the feasibility of inducing programs from examples in a rich domain — interactive graphics — without placing an excessive burden on the user. It is hoped that programming by example will help ordinary users customize their graphics editors with a minimum of effort.

Acknowledgements

This research is supported by the Natural Sciences and Engineering Research Council of Canada. We gratefully acknowledge the key role Bruce MacDonald has played in helping us to develop these ideas, and the stimulating research environment provided by the Calgary Machine Learning Group.

Task		Steps Performed in Task				Edges in Program Graph	
	Trace #	Total	by Basil	Ratio	Rejected	Total	Growth
Colonnade	1	35	12	0.34	5	22	22
	2	27	27	1	0	22	0
Connectivity	1	6	0	0	0	7	7
	2	6	6	1	0	7	0
	5*	4	1	0.25	2	11	4
	6	4	4	1	0	11	0
	7	6	6	1	0	11	0
	8	6	6	1	0	11	0
Box-to-Line	1	20	8	0.4	0	13	13
	2	24	24	1	0	13	0
	3	20	20	1	0	13	0

* variant of task: move one end-point rather than entire line

Table 1. System performance in learning three tasks.

References

[Abbott 1884]
Edwin A. Abbott. *Flatland — A Romance of Many Dimensions*. Signet Classics. New York. 1984.

[Andreae 77]
J. H. Andreae. *Thinking with the Teachable Machine*. Academic Press. London. 1977.

[Andreae 85]
P. M. Andreae. "Justified Generalization: Acquiring Procedures from Examples." PhD dissertation. MIT. January 1985.

[Angluin 83]
D. Angluin, C. H. Smith. "Inductive inference: theory and methods," *Computing Surveys 3* (15), p. 237-269. September 1983.

[Bier 86]
E. A. Bier, M. C. Stone. "Snap-Dragging." Computer Graphics: Proc. ACM SIGGRAPH. Dallas. August 1986.

[Borning 86]
A. Borning. "Defining Constraints Graphically." Human Factors in Computing Systems: Proc. ACM SIGCHI '86. Boston. April 1986.

[Cutter 87]
M. Cutter, B. Halpern, J. Spiegel. MacDraw. Apple Computer Inc. 1985, 1987

[Fuller 86]
N. Fuller, P. Prusinkiewicz. "L.E.G.O.—An Interactive Graphics System for Teaching Geometry and Computer Graphics." Proc. CIPS Edmonton. 1986.

[Geller 87]
J. Geller, S. C. Shapiro. "Graphical Deep Knowledge for Intelligent Machine Drafting." Proc. IJCAI 87. Milan. August 1987.

[Halbert 84]
D. C. Halbert. "Programming by Example." Research Report OSD-T8402. Xerox PARC. Palo Alto CA. December 1984.

[MacDonald 87]
B. A. MacDonald, I. H. Witten. "Programming Computer Controlled Systems by Non-Experts." Proc. IEEE SMC Annual conference. Alexandria, VA. October 1987.

[Maulsby 88]
D. L. Maulsby. "Inducing procedures interactively." Masters thesis. Dept. of Computer Science, University of Calgary. December 1988.

[Maulsby 89a]
D. L. Maulsby, I. H. Witten. "Inducing procedures in a direct-manipulation environment." *Proc. CHI '89*. (in press)

[Maulsby 89b]
D. L. Maulsby, K. A. Kittlitz, I. H. Witten. "Constraint-solving in interactive graphics—a user-friendly approach." *Proc. Computer Graphics International 1989*. (in press)

[Myers 86a]
B. A. Myers. "Visual Programming, Programming by Example, and Program Visualization: A Taxonomy." Human Factors in Computing Systems: Proc. SIGCHI '86. Boston. April 1986.

[Myers 86b]
B. A. Myers, W. Buxton. "Creating Highly-Interactive and Graphical User Interfaces by Demonstration." Computer Graphics: Proc. ACM SIGGRAPH '86. August 1986.

[Noma 88]
T. Noma, T. L. Kunii, N. Kin, H. Enomoto, E. Aso, T. Y. Yamamoto. "Drawing input through geometrical constructions: specification and applications." in [Thalmann 88]. pp. 403-415.

[Papert 81]
S. Papert. *Mindstorms*. Basic Books. New York. 1980.

[Smith 75]
D. C. Smith, "Pygmalion: a Creative Programming Environment." Report No. STAN-CS-75-499. Stanford U. 1975.

[Tempo 86]
Tempo. Affinity MicroSystems Ltd. Boulder CO. 1986.

[Thalmann 88]
N. Magnenat-Thalmann, D. Thalmann, eds. New Trends in Computer Graphics. Proc CG International '88. June 1988.

[Van Lehn 83]
K. Van Lehn. "Felicity Conditions for Human Skill Acquisition: Validating an AI-Based Theory." Research Report CIS-21. Xerox PARC. Palo Alto CA. 1983.

[White 88]
R. M. White. "Applying direct manipulation to geometric construction systems." in [Thalmann 88]. pp. 446-455.