Pergamon

Computers in Human Behavior, Vol. 14, No. 3, pp. 429–448, 1998

Printed in Great Britain
0747-5632/98/$19.00 + 0.00

PII: S0747-5632(98)00015-6

Designing Computer Support for Collaborative Visual Learning in the Domain of Computer Programming

Jihn-Chang J. Jehng

Institute of Human Resource Management,
National Central University,
Republic of China

Tak-Wai Chan

Department of Computer Science and Information
Engineering, National Central University,
Republic of China

Abstract — *Collaborative learning of visual information with computers can be particularly beneficial for acquiring complex and abstract knowledge. This article describes a computer-supported collaborative visual learning environment called TurtleGraph that was designed to assist learners in capturing the concept of recursion as well as recursive programming skills. In this distributed learning environment, students were requested to collaborate with their partners to write LISP-LOGO recursive programs in order to solve geometric pattern drawing problems. The instructional aim of the TurtleGraph collaborative visual learning environment was to foster active knowledge processes through collaborative work*

Requests for reprints should be addressed to Jihn-Chang J. Jehng, Institute of Human Resource Management, National Central University, Chung-Li, Taiwan, People's Republic of China. E-mail: jehng@src.ncu.edu.tw

that helps learners make their strategic thinking more explicit and induce more reflective thoughts, and also helps them be more critical in evaluating and interpreting the adequacy of their knowledge.

Keywords — collaborative, learning, visual information

The use of computers in education has been dominated by a view that the computer is an ideal medium for individualized instruction because of its capacity to vary instructional presentations and alter instructional decisions to satisfy individual requirements. This interest in developing individualized instruction also parallels a long-term emphasis in learning research involving issues of individual cognition (e.g., representation, reasoning, and decision making). Recent research has suggested there are limitations regarding learning as an individual activity because most of our learning occurs in the context of social activity (Brown, Collins, & Duguid, 1989; Lave & Wenger, 1991; Resnick, 1991; Rogoff & Lave, 1984). Students usually interact with their teachers or work with their partners to process knowledge. Therefore, instructional systems must be designed from the sociocognitive perspective to enhance learning.

The learning of computer programming is suitable for collaborative work. This is because collaborative learning or problem solving is mainly used for pedagogical reasons with the goal of promoting effective learning of difficult and complex knowledge. Computer programming is a domain containing complex knowledge and many interwoven concepts that often require an individual to expend considerable mental effort to understand. In fact, real-world programming is often collaborative in nature. Nevertheless, exactly how a computer-based learning environment can be designed and used to support the learning of programming knowledge through collaboration, particularly at the initial stage of learning, still remains unclear. This article describes a research project that has developed a distributed visual learning environment called TurtleGraph. The system has been demonstrated to be effective in an introductory computer programming course in assisting students to develop their programming skills through collaborative work (Jehng, in press).

THEORETICAL FOUNDATION OF PEER-BASED COLLABORATIVE LEARNING

The improvement of an individual's cognitive development and academic performance associated with peer-based collaboration is well documented

(Johnson & Johnson, 1990). Piagetian sociocognitive conflict theory is probably the most influential theoretical perspective on the influence of peer-based interaction. The theory emphasizes the sociocognitive conflict as the crucial mechanism responsible for the development of individual conceptual knowledge (Piaget, 1978). Cognitive conflicts arises when there is a perceived contradiction between the learner's existing understanding and what the learner has already experienced (Piaget, 1973). The contradiction and disequilibrating effect it has on the learner lead the learner to question his or her beliefs and to try out new ideas. The learner has the potential to actively construct deeper and more abstract knowledge and develop understanding by reflecting on his or her own activities and the arguments used in social interaction.

An alternative to this view is the internalization of social processes as the mechanism for learning. Cognitive development takes place within the zone of proximal development where external social interaction plays an influential role and provides a cognitive scaffold to help shape an individual's mental structure (Vygotsky, 1978). The development of individual cognitive skills is determined through learning under adult guidance or in collaboration with more capable peers. For example, studies have demonstrated that students' reading comprehension can be improved when they learn to take turns coaching each other while reading a text (Palincsar & Brown, 1984). Students learn not only to control and monitor their own behaviors, but also to develop more effective learning strategies through reciprocal tutoring, evaluation, and criticism. Collaborative learning or problem solving provides a collective zone of proximal development in which learners can cross-fertilize each other's knowledge and help move to a higher level of competence in performing tasks (Scardamalia & Breiter, 1991).

Peer-based collaboration can facilitate the change of individual cognitive structure and lead to convergence of different knowledge held by the collaborative partners. In collaboration, the thinking is distributed among members of the group. The essence of collaboration is convergence—the construction of shared meanings for conversations, concepts, and experiences (Roschelle, 1992). Mutual understanding is achieved through an iterative process of displaying, confirming, rejecting, and repairing the shared meaning. The iterative interactions lead to the joint use of meanings that are progressively constrained. This requires individuals to use communication devices to articulate meaning precisely in relation to each action in a situation.

In sum, peer-based collaborative interaction provides a social context where higher order cognitive behaviors, such as modeling, scaffolding, articulating, and reflecting, are encouraged in order to have a productive learning outcome. Peer-based collaboration can be enhanced if a learning environment can augment the communication process and induce more constructive social and intellectual interactions. A collaborative learning

environment that provides opportunities of effective social modeling and scaffolded guidance and can help learners attend, reflect, articulate, and evaluate what they have learned is an ideal social context for assisting learners to develop complex cognitive skills.

THE DESIGN OF TURTLEGRAPH LEARNING ACTIVITIES

The design of the TurtleGraph collaborative visual learning environment was inspired by the ideas addressed in the previous section. The purpose of the system is to teach the skill of recursive programming. Years of teaching experience in an introductory computer programming course have shown that most students fail to transfer their programming skills from procedural to functional language, such as LISP, in which the program code is mainly comprised of recursive procedures. The experience suggests to us that if we wish to teach our students how to write a computer program with recursive procedures, we ought to be able to tell them what a recursion is in such a way that it will enable them to discover the definition of recursion on their own. We have developed a fairly simple language and its interpreter called LISP-LOGO to make recursive constructs easier to learn. The language was quite similar to its aborigine, LISP, but with a simpler syntax. The language carried a few LISP and several basic LOGO graphic commands, such as Forward and Right. Including LOGO commands into LISP-like codes equipped LISP-LOGO with a capability of generating graphics so that the language itself can explicitly represent the recursive structure externally.

Student Tasks in TurtleGraph

The tasks that we designed were collaborative visual learning tasks. While learning what recursion is in TurtleGraph, students were requested to collaboratively write LISP-LOGO programs in order to solve geometric pattern drawing problems. The use of geometric pattern drawing tasks had its instructional advantages. First, we believed visual learning tasks can make it more effective to acquire abstract knowledge than using those tasks that are purely symbolic or text-based. The task itself could reduce the students' cognitive load by making the knowledge acquisition and interpretation process more perceptual, rather than textual or syntactic. Processing pictorial or visual information is easier for most learners than processing abstract symbolic or verbal information (Ferguson, 1977; Larkin & Simon, 1987). Second, visual learning tasks can make peer-based collaboration more effective. Students can quickly and easily receive feedback from visual information while coinvestigating and coevaluating their collaborative work.

The visual learning tasks also help learners coregulate their collaborative work in a more efficient way. Arguments among participants in a team can easily be resolved through visual comparisons than through logical inferences from purely textually presented syntactic statements.

Finally, the most important advantage of using visual learning tasks is that geometric patterns can explicitly represent the deep structure of recursive procedures. In teaching programming of recursive graphic patterns, one is didactically motivated to approach recursion as a property of the object produced by the program, not merely as a property of the program itself or of the execution process that the program itself refers to. In most computer programming textbooks, the definition of recursion is vaguely referred to as "a procedure defined in terms of itself", and the syntactic definition of a recursive procedure is referred to as "a pattern of process of evolution". However, this fails to stress the point that recursion essentially reflects an execution process that occurs in the computer. To illuminate the deep structure of the concept of recursion, it is suggested that the learning task must be designed in such a way that a certain relationship that holds between a program and a certain manner in which that a program itself can be viewed as emulating a given object. Therefore, the geometric pattern can be the output of the program, or even the process of generating that pattern is created by the program. According to this approach, in thinking about recursion, the beginning students can start from the programming task.

In each pattern drawing task, students were requested to write programs to draw a pattern by following a predefined drawing sequence with a fixed starting and finishing point. For our learning task, the recursive function was taught, if not defined, as a type of regularity of the structure of an object. In particular, the recursion may be partially defined as a high level of regularity that can be attributed to the object by viewing it as an instance of an infinite sequence of similar objects. The process of visual composition of a pattern can easily reflect the structure of the corresponding recursive procedure. The tasks were designed on the basis of three basic forms of recursive construct: headcall, middlecall, and tail recursion. The classification was based on the location of the recursive function call inside a recursive procedure as reflected by the process that a recursive graphic pattern is generated. Figure 1 provides examples of the three basic recursive constructs associated with their corresponding geometric patterns.

As Figure 1 shows, the headcall recursive procedure usually includes its recursive function call at the beginning of the program, while the tail recursion has its recursive function call at the end of the program. Both recursive constructs generate the same geometric pattern but with opposite starting and finishing points. The middlecall recursion usually generates a geometric pattern with a symmetrical form. Different pattern design tasks can help

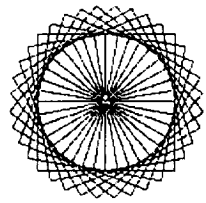
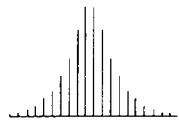
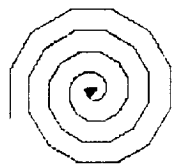
Tail Recursion	Pattern: Whirl
(defun **whirl** (counter length angle) (cond ((/= counter 0) (rectangle length) (RT angle) **(whirl (- counter 1) length angle)**)))	
Middlecall Recursion	**Pattern: Symgrowthbar**
(defun **symgrowthbar** (counter length) (cond ((/= counter 0) (FD length) (BK length) (RT 90) (FD 10) (LT 90) **(symgrowthbar (- counter 1) (* length 1.5))** (FD length) (BK length) (RT 90) (FD 10) (LT 90))))	
Headcall recursion	**Pattern: Complexspiral**
(defun **complexspiral** (counter length angle) (cond ((/= counter 0) **(complexspiral (- counter 1) (+ length 1) angle)** (FD length) (RT angle))))	

Figure 1. Three basic forms of the LISP-LOGO recursive program along with their associated geometric figures.

students construct, analyze, and structure their own understanding about recursion. While drawing a geometric pattern, learners must analyze and tinker with parts of that pattern and construct the definition from its component pieces. For instance, consider the pattern "Whirl" in Figure 1. It seems that most people have no trouble recognizing the pattern, as characterized by certain regularity. When prodded further, they may realize that what they perceive as a regularity does not conform with their initial and naive notion. As a rule, when asked for their idea of a regularity, they will respond with the first-order definition that the pattern is comprised of equally sized squares at different angles. They then realize that this definition does not apply to other particular patterns. After some in-depth examination, they may

recognize that the regularity of the pattern is based on a recurrence of a method, more than on a recurrence of a part. Let us call this type of regularity, which is based on a repetition of a method of generating the whole pattern, a second-order regularity. Instead of just the definition of recursion as "a procedure defined in terms of itself", learners will have a more concrete idea about recursion and the recursive process that generates a pattern. For example, the pattern "Whirl" in Figure 1 can be characterized as an infinite sequence that (a) represents a recurrence of a fixed method for the construction of the pattern, (b) is monotonically changing, and (c) reveals differences between all successive elements containing a first-order regularity. Such interpretation can also be applied to the pattern "Symgrowthbar" and the pattern "Complexspiral" in Figure 1. Figure 2 shows a step-by-step growing process of the pattern "Whirl" as a sequence of equally sized squares of successive generation.

THE INSTRUCTIONAL PRINCIPLES OF THE TURTLEGRAPH LEARNING ENVIRONMENT

From our teaching experience, we speculate that our students do not lack competence to learn but have not had sufficient opportunities to apply their knowledge and utilize their monitoring and reflecting skills in developing their computer programming knowledge. We also found that learners have difficulty meshing their monitoring skills with their executive procedures. Thus, TurtleGraph provides a collaborative problem-solving context in which our learners can apply their knowledge and develop a self-monitoring ability through both individual interactions with the system and the influence of peer interaction. The design of the TurtleGraph collaborative learning environment featured three instructional principles.

Reflective Learning Principle

Researchers have argued that the computer can be used as a tool for learners to observe their own learning (Collins & Brown, 1988; Dillenbourg, 1992).

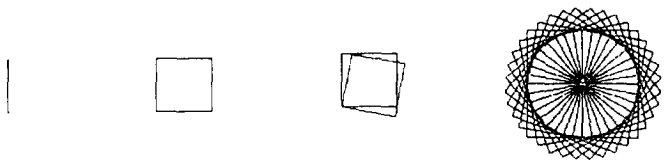


Figure 2. The growing process of the figure "Whirl" as a sequence of squares of successive generation.

Making learning observable means showing learners some representation of how and what they have learned. Owing to the constraint of the capacity of human memory, learners can not memorize all the actions they have made in the process of learning. However, the computer can help learners keep track of how they have behaved and allow them to examine, analyze, compare, and reflect on their prior actions.

An underlying technique employed in TurtleGraph is to present some trace of each individual learner's initiated actions and of the environment's responses. The system has the capability of keeping track of learners' conversations and displaying these conversations on the screen. This is intended to help learners observe and monitor their learning behaviors at any time while working with the system. As Figure 3 shows, all collaborative partners' conversations can be recorded and displayed chronologically inside the Dialogue Recorder. Those dialogues explicitly represent the entire problem-solving process. Learners can utilize their conversations to (a) understand their interactions and evaluate the problem-solving history, (b) reflect on their problem-solving approaches, (c) organize their conversational actions, and (d) develop novel problem-solving strategies by comparing previously developed ideas.

As Figure 4 shows, learners using TurtleGraph can also see and compare their own programming code with their partner's by pressing the Partner's Editor button and opening their partner's Program Editor window at any point. Through comparison, learners cannot only quickly locate faults and weaknesses in their own problem-solving approaches, but can also modify and improve their problem-solving strategies at critical points. Learning through reflection by comparison can help learners elaborate on what has been learned and make the entire learning process more meaningful.

Reactive Learning Principle

The TurtleGraph learning environment was designed in such a way that is sensitive to learners' actions or responses and provides immediate feedback that will extend learners' understanding of their own actions in a context of specific learning situations. In the situated learning model, learning is accomplished through tuning of attention and perception in relation to each action made by learners. According to the model, perception rather than memory is addressed as the means by which we learn. Meaning is not retrieved from previous representation or schema stored in the memory, but is generated on the spot through perceiving and acting. Knowledge is always a novel construction that continues to develop in every interaction that learners make in their environments (Clancey, 1994). To make learning be more

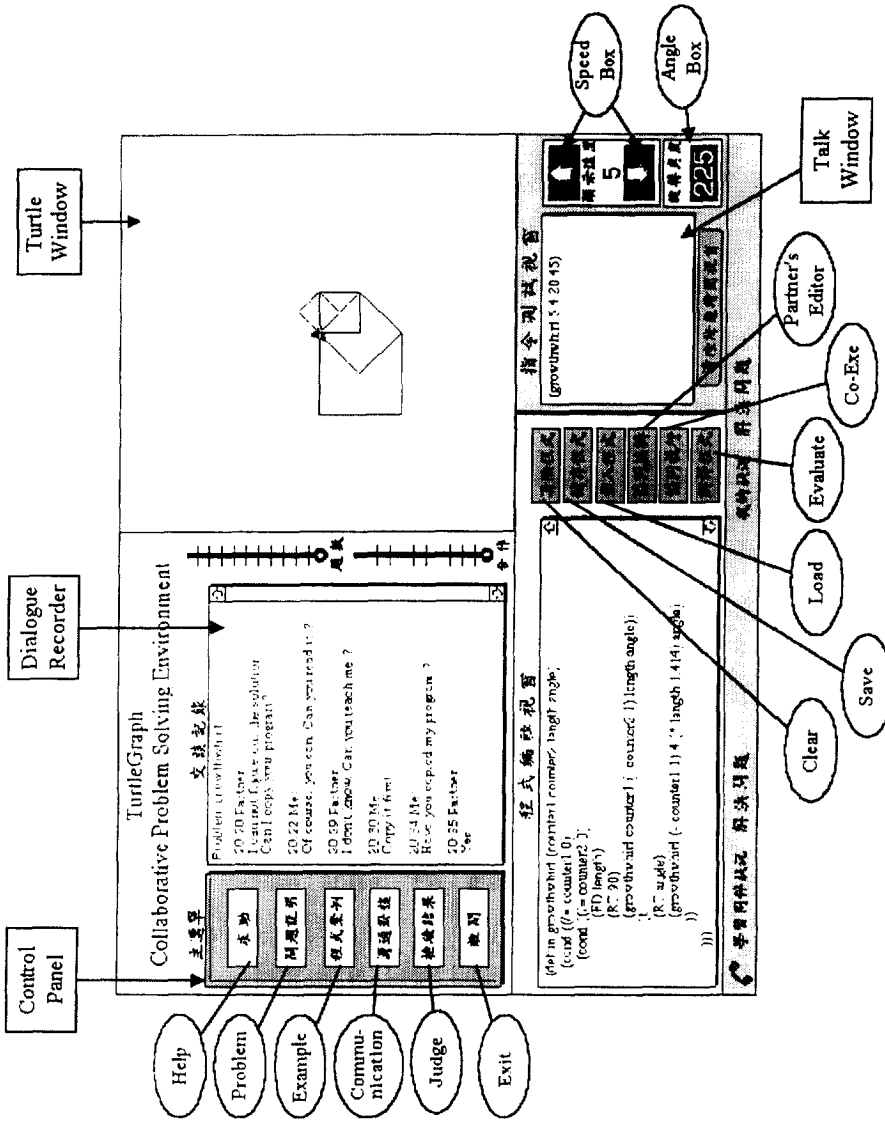


Figure 3. TurtleGraph problem-solving environment.

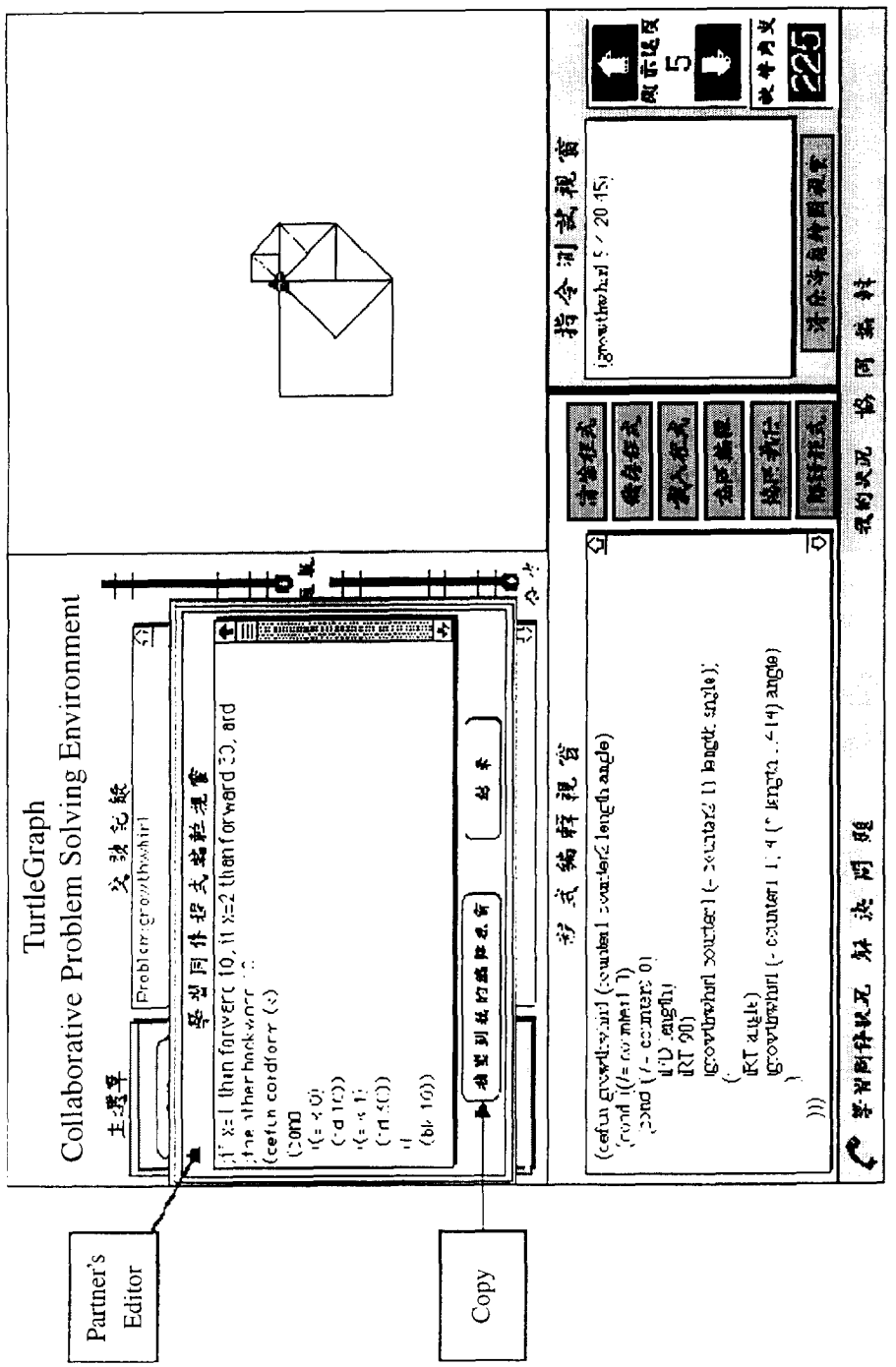


Figure 4. The Partner's Editor.

perceptual, TurtleGraph employs a learning environment that emphasizes dynamic problem representations shown in an external display that is reactive to each action made by learners (Dugdale, 1994; Larkin, 1988). As Figure 3 shows, learners can immediately observe the process of how a geometric pattern was generated while its corresponding recursive program is being executed. Learners can evaluate the accuracy of their programs with corresponding graphical patterns by tracing the movement of the turtle and analyzing the angle with each turn made by the turtle. The dynamic external display is so reactive to learners' problem-solving behaviors that it can help students explicitly envision the state of their knowledge and help them closely monitor, analyze, and modify their knowledge.

Structured Learning Principle

A variety of representative pattern drawing tasks were presented in TurtleGraph with different levels of complexity and difficulty. Each problem starts with a story and learners are assumed to play the role of computer graphic designers to design a geometric pattern. The system incorporates the concept of scaffolding in order to guide learners in the use of recursive procedures to solve the problems. As Figure 5 shows, learners are also allowed to access and utilize relevant example programs to solve the problem. The example programs are provided in a manner that is specific to certain design tasks.

Just as an effective teacher pays close attention to the learner's current level of ability and gives them a task that builds on their prior learning (Johnson, Flesher, Jehng, & Ferej, 1993), TurtleGraph sequences learning experiences based on each individual learner's prior problem-solving performance. The presentation of the task is based on the complexity and difficulty of the recursive constructs. At the initial stage, learners are requested to write a simple tail recursion to simulate a one-layer loop to draw a pattern, such as the "polygon", whose geometric structure can be easily recognized and analyzed (Figure 6a). At a later stage, a much more complex design task, such as the "binary-tree" pattern, is presented with which learners must utilize more complex recursive programming routines to draw the figure (Figure 6b). Each time students accomplish a task, the system keeps updating their performance records based on the teacher's evaluation and makes a decision concerning the difficulty of next task that learners will work on. Learners must demonstrate a certain level of problem-solving skill before advancing to more sophisticated problems. Learning by practicing knowledge with different problem-solving tasks can help learners consolidate their acquired knowledge and gain deep understanding of how the knowledge can be used in a wide variety of problem contexts.

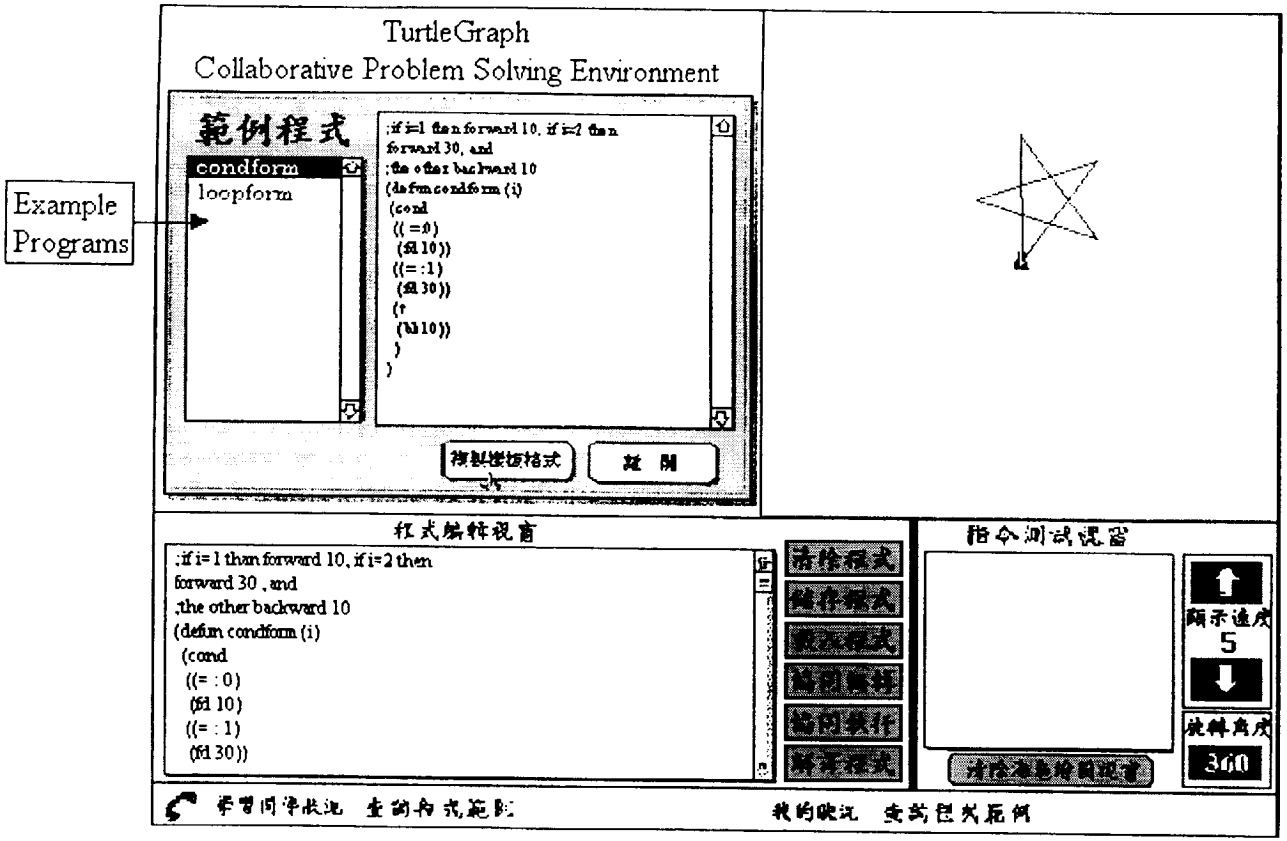
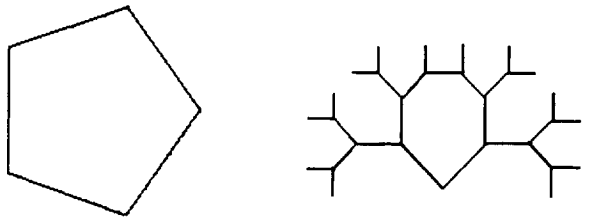


Figure 5. Example programs provided by the system.



a. Geometric pattern of "polygon" b. Geometric pattern of "binary tree"

Figure 6. Example of geometric patterns.

Button-Oriented Interface for Effective Collaborative Interactions

In addition to the three instructional principles, as a distributed learning environment where students share the teamwork at different locations, TurtleGraph provides communication facilities that allow learners to synchronously communicate and exchange their ideas. The design of the communication interface applies and extends the notion of button control theory (Jona, Menachem, Bell, & Birnbaum, 1991; Schank & Jona, 1991). Advocators of button control theory propose that students, in a computer-based learning environment, should have significant control over what they see, hear, and learn through the "button". Rather than using natural language processing, which is difficult to implement, each message corresponds to one button, represented either textually or iconically on the computer screen. TurtleGraph provides a button-oriented communication interface to help students monitor their conversations and control their conversation behaviors. Each time learners press a button initiating a conversation, that button will immediately highlight their partners' previous messages and prompt them to elaborate their ideas to respond to these messages in order to form a coherent conversational interaction (Figure 7). These buttons are designed to assist problem solvers to carefully develop their thoughts and regulate their problem-solving behaviors during collaborative interaction. This, in turn, may help organize ideas and promote more effective and efficient interaction.

THE FUNCTIONALITY OF THE TURTLEGRAPH LEARNING ENVIRONMENT

The system operates on Macintosh computers connected to one another via AppleTalk. A short lesson is provided at the beginning to help students familiarize themselves with the LISP-LOGO language. After finishing the

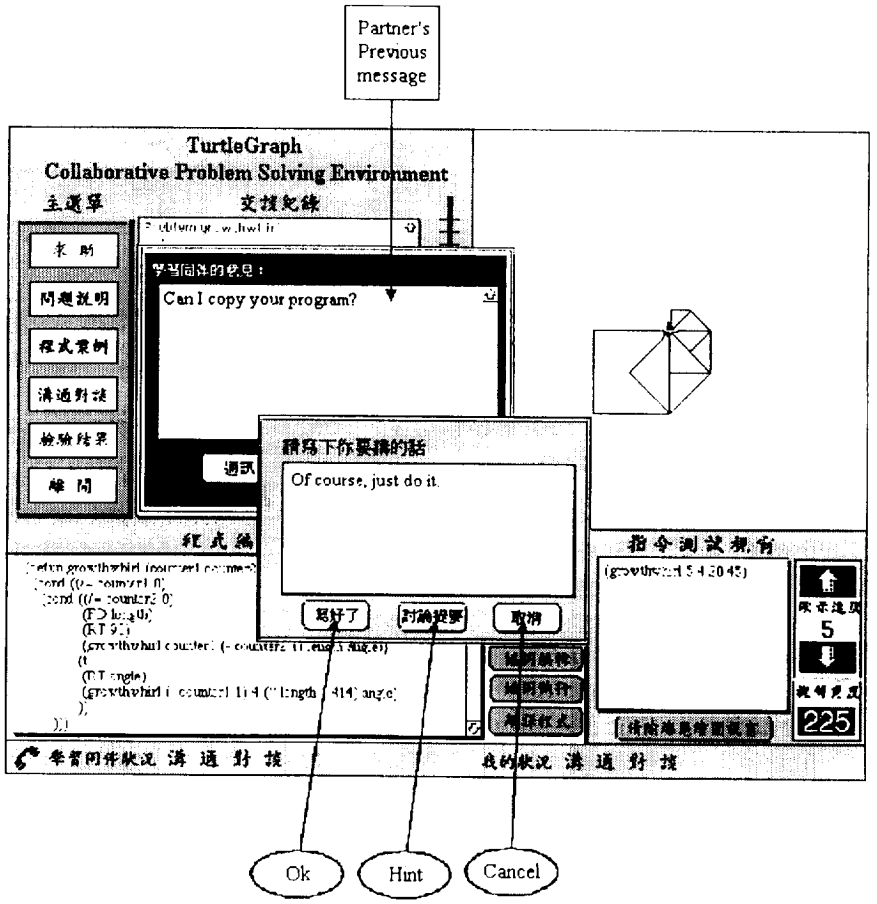


Figure 7. The conversation boxes appear by pressing the Communication button.

tutorial, students will read a story in which they are asked to play the role of a computer graphic designer who needs to choose an on-line partner to form a collaborative team to complete a geometric pattern drawing task. They are then brought into the TurtleGraph Collaborative Problem-Solving Environment to work on the problem as shown in Figure 3.

The environment contains five main functional areas: the Control Panel, the Turtle Window, the Dialogue Recorder, the Program Editor, and the Talk Window. The Control Panel, located on the left side of the screen, contains six buttons. The Help button includes specific instructions that guide students to use the system. The instructions provided are context-sensitive. Learners can click on the Help button at any time to request assistance from the system. Next to the Help button is the Problem button. Each time students click on the Problem button, the system provides a detailed explanation about the task. The system first demonstrates how a geometric pattern is generated and then requests learners to draw exactly the same pattern by following exactly

the same drawing steps with the same starting and finishing point. The Example button contains example programs for reference. The presented examples vary specifically in response to each design task. Information provided by the Example button can help students organize their knowledge and develop more effective problem-solving strategies (Schank, Linn, & Clancey, 1990). Students are allowed to copy the example programs into their Program Editor, or execute them to see how they work and perhaps adapt those program codes for their own purpose. By clicking on the Communication button, students can write text information to exchange and discuss ideas with their partners. A conversation box appears on the screen with their partners' messages highlighted so that they are able to know how to respond. By clicking on the Judge button, the system will automatically send students' solutions to the server. The teacher evaluates students' solutions at the server side to decide how successfully they have accomplished the task, adds points into the their grade books, and provides feedback. Each time a student accomplishes a task the system will maintain a record of his or her performance in the database and update his or her performance history. Therefore, the next time students enters the system, the system will automatically evaluate their previous level of performance and give them appropriate tasks for further learning (Johnson et al., 1993). Students can choose to leave the system through the Exit button at any point.

The Dialogue Recorder, which is next to the Control Panel, keeps track of students' conversations chronologically during each learning session. Students can read and reflect on their own conversations while working on the problem. A turtle is symbolically represented by a dark solid triangle that moves in response to the execution of the LISP-LOGO commands. A pattern is simultaneously generated along the course of its movement. Students can visually evaluate their programs with the corresponding patterns just generated. The Turtle Window and Dialogue Recorder provides an environment where students can closely examine and analyze the state of their knowledge.

The Program Editor is located on the bottom of the computer screen. Students write their LISP-LOGO programs inside the editor window. Editing functions such as copying, cutting, and pasting are all available through shortcut keys. The Save button is used for saving the current program and the Load button is used for retrieving a previously written program. Students must compile the program before it is executed by pressing the Evaluate button. Students can either examine or make a duplicate copy of their partners' programs for their own use by opening their partners' Program Editor through the Partner's Editor button. Another deliberate design of the system is the Co-Exe button. Students can click on the Co-Exe button to execute their programs and the patterns that their programs have just generated will simultaneously appear inside both their and their partners'

Turtle Window. The design of the Partner's Editor and Co-Exe button helps students collaborate to work on ideas effectively. Beside the Program Editor is the Talk Window where a compiled program can be executed. Next to the Talk window are Speed Box and Angle Box. Students can control the speed of the turtle by pressing either the "up arrow" or "down arrow" button inside the Speed Box and can also check the degree of the angle with each turn made by the turtle through the Angle Box. The purpose of the Speed and Angle Boxes is to help students to generate, examine, and analyze their programs.

The entire system represents a self-contained collaborative visual learning environment with rich resources available for students to explore and communicate. The instructional materials and learning facilities provided by the system are closely akin to each other and function as one so that students can seek closure and feel a sense of control of their learning behaviors.

EVALUATING COLLABORATIVE LEARNING IN TURTLEGRAPH

TurtleGraph has been used in an introductory computer programming course which was offered to assist social science students to learn recursive programming. A study has been conducted in this introductory computer programming course to compare whether students learning recursive programming in collaboration with their remote partners could develop better programming skills than those who either collaborate to learn in a face-to-face context or learn individually.

Subjects

A total of 94 social science students who did not receive any computer programming training participated in this course. In the first half of the semester, these students were taught to write simple LOGO and SCHEME programs. They were then divided into high-achievement, medium-achievement, and low-achievement clusters based on their performance on the mid-term test. The high-achievement cluster was comprised of students whose test performances were in the top 33.3% ($n = 31$), the medium-achievement was comprised of students whose test performance were in the middle 33.3% ($n = 32$), and the low-achievement was comprised of students whose test performances were in the bottom 33.3% ($n = 31$) of the class.

Procedures

During the experiment, all students were organized to learn recursive programming in the TurtleGraph learning environment. At the beginning,

all students took a 2-hr session to learn LISP-LOGO and the basic recursive programming concept. They were then randomly selected from the three achievement clusters and assigned into three learning conditions: distributed collaborative ($n=36$), face-to-face collaborative ($n=36$), and individual learning ($n=22$). Students in each of the two collaborative learning conditions were then arranged into 6 high–medium, 6 high–low, and 6 medium–low collaborative pairs. All the students in the three learning conditions were asked to learn to write LISP-LOGO recursive programs to solve four geometric figure drawing tasks at four different learning sessions each of which lasted 1 hr.

Transfer Measures

At the end of the four learning sessions, all students were required to take a test individually. The test contained three debugging, three prediction, three fill-in-blank, and one programming test items. The debugging and prediction test items were designed as multiple-choice questions primarily measuring students' program evaluation ability. The fill-in-blank test was to measure program completion, and the programming test was to measure program generation ability.

Data Analysis and Results

Table 1 shows the results of student performances in the posttest. Students in the three learning conditions performed equally well on the program evaluation and program completion test. Nevertheless, students who learned the recursive programming in collaboration with others in TurtleGraph, either in a distributed or face-to-face collaborative context, developed much

Table 1. Student Performances on the Posttest

Type of programming skill test	Distributed learning ($n=33$)[a]	Face-to-face learning ($n=36$)	Individual learning ($n=18$)[a]	χ^2
Program evaluation	75.8%[b]	78.7%	75.9%	1.04
Program completion	68.7%[b]	63.0%	66.7%	0.26
Program generation	36.4%[c]	44.4%	16.7%	32.89*

[a]Three students in the distributed learning and four students in the individual learning group did not show up to take the posttest so their performances on the posttest were not counted.
[b]All percents listed in rows of program evaluation and completion tests indicate proportions of test items correctly solved by students.
[c]All percents listed in the row of program generation test indicate proportions of students in each learning group who solved the problem successfully.
*$p < .05$.

better program generation skills than those who learned recursive programming individually, $\chi^2 = 32.89$, $p < .05$. Students in the distributed learning context expressed that they were motivated to use the system and felt that the collaborative visual learning really helped them efficiently capture the concept of recursion and acquire recursive programming skills. Based on the results, we believe that sound instructional design is important to make a collaborative visual learning environment educationally effective.

DISCUSSION

Contemporary educational theory increasingly values collaborative learning and group work. Constructive social and intellectual interactions between learners are important in teaching methods in order to produce significant learning. It is generally recognized that significant learning is likely to occur when learners actively participate in building their own knowledge and share the results of their learning experience with others (Resnick, 1991). Modern computer technology can be applied to support and mediate such an interactive process and make it more constructive and productive.

TurtleGraph is a distributed learning environment that utilizes the capability of computer technology to help students acquire recursive programming skills and improve their understanding of the concept of recursion through collaborative visual learning tasks. The use of visual learning tasks is to make the learning process more visible and explorable to the students. Novice learners usually have difficulty in becoming active learners to process their knowledge, since they are not able to envision the state of their knowledge and do not know where and how to detect deficits in their knowledge. Visual learning can resolve such limitations. The geometric pattern drawing tasks, provided by TurtleGraph, explicitly represent crucial characteristics of the concept of recursion and make it easier to learn. Students can immediately know how they have performed on the tasks and get meaningful feedback from the visual information. The dynamic external display can help students be more active in examining, analyzing, and evaluating their knowledge.

Different from the conventional notion regarding learning where becoming expert in a domain requires gradual reception and internalization of already created knowledge and procedures, TurtleGraph provides learning activities that leave extensive latitude for learners to explore their own understanding about recursion. The visual learning tasks become an adventure of exploring, designing, and implementing what learners try to develop in an activity. The first-hand experience replaces conventional second-order learning which involves being aware of the context of behaving and working ways of doing things that are usually taken for granted (Dugdale, 1994).

The visual learning tasks provided by TurtleGraph are accomplished by students through collaborative work. The system is designed to play an active role of agent that mediates interactive learning. TurtleGraph utilizes advanced computer network technology to form a distributed learning environment that allows members of the collaborative team to work at different locations and reduces the social inequality problem that usually occurs in the traditional face-to-face situation (Kiesler, Siegel, & Mcguire, 1984; Ruberg, Moore, & Taylor, 1996). Facilities provided by the system, such as scaffolded guidance and examples, are also changeable in the course of time to support learning. The modifiable instructional support is a requirement for TurtleGraph to sustain the learning activities in which joint adventure of knowledge is encouraged.

Acknowledgments — The research project was funded by the National Science Council, Taiwan, under contract no. NSC82-0111-S-032-005, to the first author of this article. This project was executed under the group project entitled LISA with social learning as the main research focus. The goal of the LISA project was to establish various multi-channel learning environments, where students engaged in learning activities by interacting with various agents who may be actual human learners or computer-simulated agents. The authors are grateful to Steven Liang and Yu-Fen Shih for their assistance and constructive ideas in getting started on developing the system, and are also grateful to Scott D. Johnson and Michael Jacobson for their valuable comments on the early draft of this article.

REFERENCES

Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *Educational Researcher, 18*, 32–42.

Clancey, W. J. (1994). Situated cognition: How representations are created and given meaning. In R. Lewis & P. Mendelson (Eds.), *Lessons from learning*, IFIP Transactions A-46 (pp. 231–242). Amsterdam: North-Holland.

Collins, A., & Brown, J. S. (1988). The computer as a tool for learning through reflection. In H. Mandl & A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems* (pp. 1–18). Berlin: Springer-Verlag.

Dillenbourg, P. (1992). The computer as a constructorium: Tools for observing one's own learning. In R. Moyse & M. Elsom-Cook (Eds.), *Knowledge negotiation* (pp. 185–198). London: Academic Press.

Dugdale, A. (1994). Using students' mathematical inventiveness as a foundation for software design: Toward a tempered constructivism. *Educational Technology: Research and Development, 42*, 57–73.

Ferguson, E. S. (1977). The mind's eye: Nonverbal thought in technology. *Science, 197*(4306), 827–836.

Jehng, J. C. (in press). The psycho-social processes and cognitive effects of peer-based collaborative interactions with computers. *Journal of Educational Computing Research.*

Johnson, S. D., Flesher, J. W., Jehng, J. C., & Ferej, A. (1993). Enhancing electrical troubleshooting skills in a computer-coached practice environment. *Interactive Learning Environment, 3*, 199–214.

Johnson, D. W., & Johnson, R. T. (1990). Cooperative learning and achievement. In S. Sharan (Ed.), *Cooperative learning* (pp. 23–37). New York: Praeger.

Jona, Menachem, Bell, B., & Birnbaum (1991). *Button theory: A taxonomic framework for student–teacher interaction in computer-based learning environments* (Technical report No. 12). Evanston, IL: Northwestern University, The Institute for the Learning Science.

Kiesler, S., Siegel, J., & Mcguire, T. W. (1984). Social psychological aspects of computer-mediated communication. *American Psychologist, 39*, 1123–1134.

Larkin, J. H. (1988). Display-based problem solving. In D. Klahr & Kotovsky (Ed.), *Complex information processing: The impact of Hebert A. Simon* (pp. 1–39). Hillsdale, NJ: Erlbaum.

Larkin, J. H., & Simon, H. A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science, 11*, 65–99.

Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation.* Cambridge: Cambridge University Press.

Palincsar, A. S., & Brown, A. L. (1984). Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and Instruction, 1*, 117–175.

Piaget, J. (1973). *To understand is to invent: The future of education.* New York: Grossman.

Piaget, J. (1978). *Success and understanding.* London: Routledge & Kegan Paul.

Resnick, L. B. (1991). Shared cognition: Thinking as social practice. In L. B. Resnick, J. M. Levine & S. D. Teasley (Eds.), *Perspectives on socially shared cognition* (pp. 1–20). Washington, DC: American Psychological Association.

Rogoff, B., & Lave, J. (1984). *Everyday cognition: Its development in social context.* Cambridge, MA: Harvard University Press.

Roschelle, J. (1992). Learning by collaborating: Convergent conceptual change. *The Journal of Learning Science, 2*, 235–276.

Ruberg, L. F., Moore, D. M., & Taylor, C. D. (1996). Student participation, interaction, and regulation in a computer-mediated communication environment: A qualitative study. *Journal of Educational Computing Research, 14*, 243–268.

Scardamalia, M., & Breiter, C. (1991). High level of agency for children in knowledge building: A challenge for the design of new knowledge media. *The Journal of Learning Science, 1*, 37–68.

Schank, R., & Jona, M. Y. (1991). Empowering the student: New perspectives on the design of teaching systems. *The Journal of Learning Science, 1*, 7–35.

Schank, P., Linn, M., & Clancey, M. (1990). *How does an on-line template library help students learn PASCAL?* Paper presented at the 1990 Annual Meeting of the American Educational Research Association, Boston.

Vygotsky, L. (1978). *Mind in society: The development of higher psychological processes.* Cambridge, MA: Harvard University Press. (Original work published in 1935)