A Computer Graphics Aided Lecture Presentation System: Exploring Animated Algorithms with Direct Manipulation

Lam, Ho Cheong 林浩昌
91176070

A thesis submitted to
the Department of Computer Science,
The Chinese University of Hong Kong
in partial fulfillment of the requirements for
the degree of Master of Philosophy
July 1993

A Computer Graphics Aided Lecture Presentation System: Exploring Animated Algorithms with Direct Manipulation

Lam, Ho Cheong
Department of Computer Science
The Chinese University of Hong Kong, Shatin, Hong Kong
E-mail: hclam@se.cuhk.hk

Thesis advisors: Dr. C.S Chang, Dr. K.S. Leung and Prof. T.C. Chen

Abstract

Visual aids are more than just flip-cards, transparencies or slides. Actually, computer graphics can be a competent means to enhance the appeal of a presentation. The current research is to design and develop a Computer Graphics Aided Lecture Presentation System to produce courseware for classrooms. We put the emphasis on developing a system according to the requirements of classroom presenters on different occasions, especially during actual presentation. Features including history, story board, highlight and zooming are proposed. Using the system, we also experiment with a model which integrates both interactive graphics and procedural programming for animating a presentation. For example, we have animated a sorting program by linking the attributes of several rectangles with some variables defined in the program. We have made known three main characteristics of the model. Firstly, users can explore "what-if" properties of algorithms with direct manipulation. For example, how the expected number of swaps varies with the way of arranging the items to be sorted is easily identified by the users. Secondly, the independence of algorithms upon graphical displays helps to produce multiple animation for an algorithm. For example, instead of rectangles, animators can denote the items as numbers with no difficulty. Lastly, algorithms are defined in the object oriented C++ language so that programmers can take advantages of its reusability. This thesis presents various novel features of the prototyped presentation system. Examples are then given to illustrate their usage.

Keywords: Presentation Software, Computer Aided Instruction, Algorithm Animation, Program Visualization and PC Applications

Acknowledgments

Many thanks to my thesis advisors. Initiating the project of "Computer Aided Presentation System", Dr. C.S. Chang has given me invaluable information on his findings from previous work. Prof. T.C. Chen often portrayed the global picture for us and taught me to think in the abstract. Setting high standards for his students, Dr. K.S. Leung has shown me the way to carry out research independently and aggressively. This thesis would not have come to fruition without their opinions and support.

Special thanks to the Department of Systems Engineering. Much of the work reported here was performed while I was using their computing facilities and office. More importantly, I will cherish the friendship for all the people, too numerous to mention here, in the department.

Surely, assistance from other staff in the Department of Computer Science is greatly appreciated. Judging from his experience in Computer Graphics, Dr. S.C. Hsu has given me insightful suggestions on the Animation Production Model.

I am also very thankful to my old folks in the University of Hong Kong. Mr. K.M. Chan often patiently listened to me and had discussions with me about my research.

In addition, I am so grateful to have made friends over the world through the Usenet. Doing research in the same area Algorithm Animation, they have sent me their recent publications and told me the latest news in the field.

A special thanks to all my family and friends, who have been there unconditionally for me forever. With all of my heart, I can never thank you enough.

Contents

1
Introduction

1.1 Prologue

Seeing is believing
proverb

What we ourselves see is the most understandable evidence. Algorithmicians can imagine an algorithm in their mind's eyes; just by reading a Pascal version of the algorithm. Unlike them, novices hardly ever can. To teach novices algorithms, enunciating the algorithms with a series of snapshots showing the operation of the algorithms is desirable. Traditionally, instructors inevitably imitated the working of algorithms using blackboard and chalk or transparencies in the theater. Unfortunately, the errant outcomes often made them puzzled after several steps of computation. With the advance of Computer Graphics, computer is actually competent in presenting algorithms in action. This will be the main theme of this thesis.

Our work is an outgrowth of the project of "A Computer Graphics Aided Lecture Presentation System", which is to investigate how computer can complement instructors in class to provide an innovative and informative medium of communication. Evolved from the presentation system, a model of animation production has been conceived and refined. The model is found to be particularly useful for animating algorithms. In addition, the model has been realized in the Pearl system (exploring animated algorithms with direct manipulation); being implemented in the Microsoft Windows 3.1 [1] using the Turbo C++ for Windows [2] on a PC [3][4].

1.2 Thesis Contributions

The primary contributions of this thesis are its two models: the Lecture Presentation and the Animation Production Models. Being a secondary contribution, the Pearl system has been designed and implemented based upon the models. In addition, numerous animations on data structures and algorithms have been made using the system[1]. The following describes the primary contributions.

1.2.1 Lecture Presentation Model

The Lecture Presentation Model is fined-tuned for classroom presentation. The procedures to give a lecture are subsumed under four categories: Materials Organization, Slide Preparation, Animation Production and Actual Presentation. For each category in turn, various novel features are proposed; including history, story board, highlight and zooming. The model is also remarkable for its improvement in actual presentations, which has seldom been notified in other work.

1.2.2 Animation Production Model

In the Animation Production Model, three steps are taken to produce an animation. A program representing a certain algorithm is firstly implemented for the motion control in the animation. The graphical objects cast in the animation are then made by direct manipulation. The last step is to link the variables declared in the program with the attributes of the graphical objects. For example, we have animated the Bubble sort by binding the attribute x of several rectangles with the variables indicating the item positions in the program.

The model has three special qualities. Firstly, while interacting with the animation, users can acquaint themselves with certain subtle properties of the underlying algorithm. For example, sorting different instances, users can relate the expected number of swaps to the arrangement of the items. Secondly, algorithm specifications are independent of their graphical representations. For instance, the items to be sorted can be easily portrayed either as numbers or as rectangles with the heights denoting the values. Lastly, the program is developed using the object oriented C++, which is conducive to reusability.

---

[1] A diskette containing the executable programs with sample source codes is available upon request.

1.3 Thesis Outline

This thesis is essentially made up of two main chapters: chapters two and three. Chapter two documents our work on the lecture presentation system. The beginning of the chapter describes the motivations leading to the current research. After giving an overview of the entire system, the chapter reports the various features of each component in turn. In addition, chapter two as well as chapter three are both self-contained.

In chapter three, we go into details of Algorithm Animation. Before entering the chapter, readers presumably have gone through the section "Animation Production" in the previous chapter, which briefly describes how someone goes about animating algorithms with the Pearl system. After reviewing previous work on Algorithm Animation, the chapter elaborates each procedure to produce an animation in depth; including, how to specify algorithms, how to represent displays, how to establish links between them and lastly how to definitely state various options. Readers can also find examples of animation created by the Pearl system in the chapter. This thesis is concluded in the last chapter with a discussion on the areas for future research and a description of general problems.

2
Lecture Presentation System

2.1 Introduction

We are still using the lecturing technology of a quarter centuries ago. Transparencies and slides are still widely used in classroom presentations. However, these traditional visual aids are all static, usually only black and white and non-reusable. When an animated presentation is needed, the traditional way is to produce a video, which in general is too expensive for the teaching institutes to afford.

Currently, high-speed light-weight laptop / notebook computers together with a color LCD display panel, which can be projected onto a large screen, make computer presentations possible in classrooms. The display can achieve up to 640 x 480 pixels, 185,000 true colors and 50 ms response time, which enables high-quality multi-media images with realism and full motion in animation. In addition, some models of the LCD panels can even allow pen-input [5], which means that a presenter can write on a tablet, and in turn directly onto the screen. In spite of the hardware advances, available software is inadequate to meet the requirements of a lecture presentation.

Authoring systems [6] are developed to let students do interactive exercises [7]. Undoubtedly, an interactive exercise can involve student participation. However, each exercise will require a great effort of the instructors to prepare so as to handle various student responses.

Commercial presentation software [8][9] is often tailored to business people. For example, they often provide users with a huge library of ready-made clip art and templates. This certainly makes a deep impression on the public audiences but they are not suitable for academic lectures.

Animation software [10] is usually compelled to be easy to use. An example is the motion control, which is often pre-defined by the software manufacturers instead of being programmable. However, demonstrations of phenomena in classes frequently demand high flexibility and accuracy. The software is thus insufficient.

This chapter presents our study on how computer can facilitate classroom presentation. In particular, a user-friendly Computer Graphics Aided Lecture Presentation System has been designed and developed to help instructors prepare, arrange and present courseware. In the future, instructors can bring along with them only the disks containing the courseware to the lecture theater.

The early prototype developed at the Chinese University of Hong Kong in 1991 [11][12] has demonstrated to us the feasibility of such a presentation system. The prototype runs on the OS/2 environment using the Presentation Manager interface. The presentation materials are treated as objects of different classes. Novice users can use pre-defined objects and operations that act on those objects to prepare a dynamic presentation. Experienced users can further define their own objects and operations.

The current system has two main characteristics. Firstly, the system is particularly adapted to the needs of classroom presenters [13]. Pin-pointing each of the procedures to deliver a lecture, we have examined novel applications of computer; including history, story board, highlight and zooming. Especially, the system assists instructors in the actual presentation, which is generally overlooked by other software. Secondly, we have introduced an animation production model, which integrates both interactive graphics and procedural programming. In short, attributes of graphical objects can be driven by the variables declared in a general program and vice versa. Hence, the tasks of modeling graphical objects and motion control can be clearly separated and in turn simplified.

Next section gives an overview of the entire presentation system. Each of the succeeding sections then fully describes respectively one of the four components: Materials Organization, Slide Preparation, Animation Production and Actual Presentation. A summary of this chapter is made in the last section.

2.2 System Overview

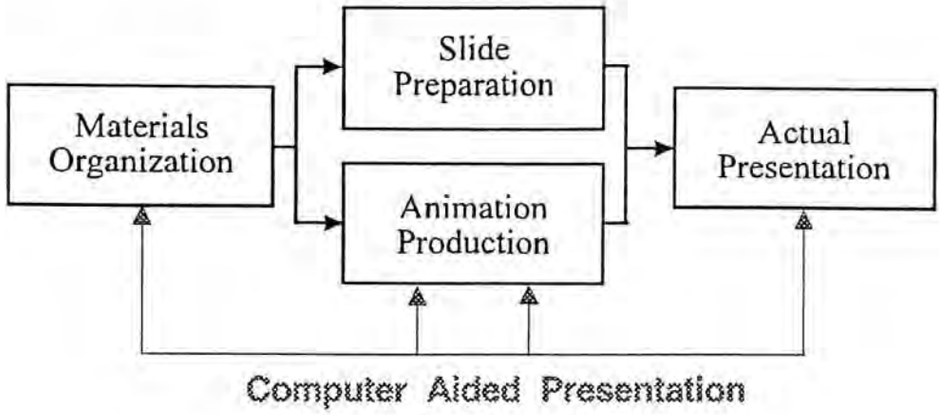


Figure 1 System Overview

The procedures to give a lecture are depicted in figure 1. Gathering materials from books and journals, instructors can first arrange the acquired knowledge into a well-organized lecture. They can then prepare the slides[1], which contain their ideas added with appropriate supporting illustrations. To make the lecture more interesting and simulating, the instructors can also produce an animation, revealing their ideas dynamically. Finally, using appropriate visual aids, the instructors can put their ideas across to the students in the theater.

The presentation system is correspondingly made up of four components; namely, Materials Organization, Slide Preparation, Animation Production and Actual Presentation, which are described in detail one by one in the following sections.

[1]Materials are organized and presented in a slide-by-slide basis in the system. Each slide is framed in a window on the projected computer screen.

2.3 Materials Organization

The Materials Organization subsystem of the presentation system helps users to organize pieces of information into a structured presentation. Moreover, the related information can be accessed in a convenient manner. Reminder Note, Idea Outliner, Hypertext, Index, History and Hand-out are features essential to good materials organization.

2.3.1 Reminder Note

For each presentation, remarks such as course title, lecturer details and reference materials can be made in the Reminder Note. The remarks are going to assist other persons in re-using the presentation in the future.

2.3.2 Idea Outliner

In the Idea Outliner, the whole organization of the presentation can be reviewed as a hierarchy tree with the current slide highlighted. Users are able to create and re-arrange this contents tree by adding and deleting nodes in order to structure the presentation. Afterwards, they can go into details by linking each node with a slide.

2.3.3 Hypertext

The hypertext concept [14][15] is adopted in the system, which means that users can associate a keyword, which is called a button, with a particular slide. During actual presentation, when the mouse cursor moves over the button, the button becomes heightened. When clicked, the button leads the users to the associated slide.

Having traversed several slides, the users may lose the place of the slide which they are currently presenting. To avoid this, they can put a bookmark on the slide which they wish to refer to later. Subsequently, upon invoking a particular function, the system will retrieve the most recent slide with the bookmark. Hence, the users can always keep track of their presenting order.

2.3.4 Index

Like indices of a book, Index helps users locate certain information embodied in a large number of slides. An index link can be attached between a topic and several slides. Through the links, the users can access all the related materials under a specific topic.

Furthermore, Index can also be used to search for words on certain slides without the knowledge of their locations. To look for a word, all the slides are automatically gone through. Words with similar spelling will be reported as well. But surely this kind of searching is less efficient than that with a link. Figure 2 shows the relationship of the Idea Outliner, Hypertext and Index.

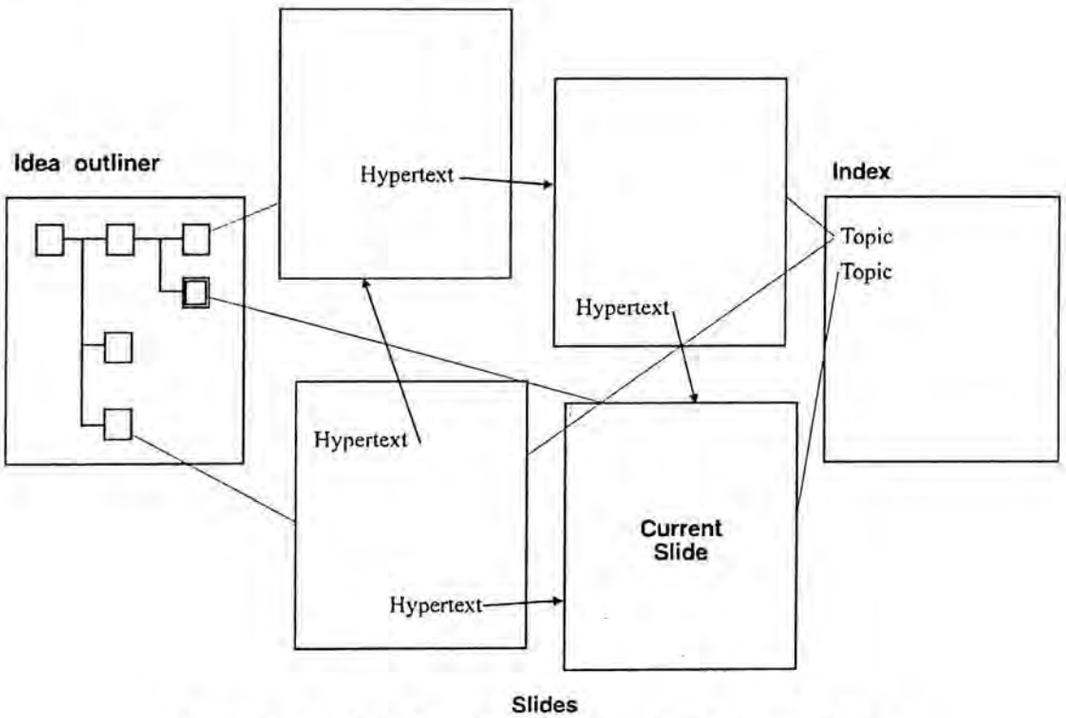


Figure 2 Idea Outliner, Hypertext and Index

2.3.5 History

A drawback of using blackboard is that contents on the blackboard will be erased shortly after mentioned. Instead, the system stores the sequence of the covered slides as a history during the lecture. History assists users in referring to a slide shown before. The users can play back the sequence hereinafter. History is especially useful when a student raises a question and wants to refer to a previous slide for discussion.

2.3.6 Hand-out

Hand-outs of each lecture are necessary for students to study after the lecture. A copy of the slides, which are probably annotated with additional descriptions, can be easily produced through a printer.[1]

[1]We have also experimented with a heuristic to improve the printing of PostScript programs. The results are included in appendix A "PostScript Optimization".

2.4 Slide Preparation

To simplify the process of preparing slides, the Slide Preparation of the presentation system provides tools for handling some frequently used graphical objects. Hence, the time for preparing these graphical objects can be greatly reduced.

2.4.1 Text

As text is extensively used in lectures, the ability to handle text is important for the system. Users are able to edit text directly on the screen as if they were using a word processor (See figure 3). Changing fonts, point sizes and typefaces are possible, and so are phrase search and paragraph alignment. Furthermore, hypertext buttons, which are also text, can be defined.

2.4.2 Shape

Drawing tools for various geometric shapes such as rectangle, ellipse, line and bezier curve are provided by the system (See figure 3). Certain attributes are associated with them; namely, line width, brush pattern and color. Bitmap is another way to represent shapes. Digitized images like those produced by scanners are usually stored as bitmaps.

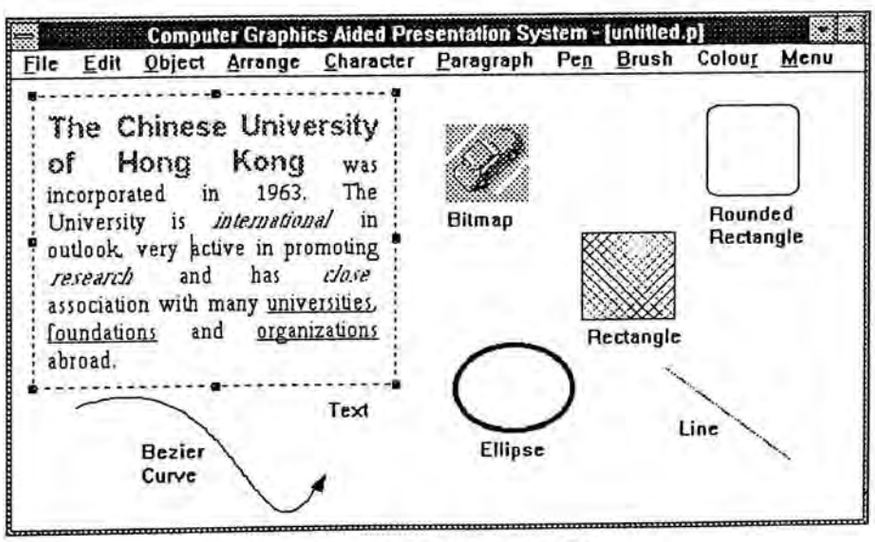


Figure 3 Text and Shape

2.4.3 Graphical Object Library

Simple graphical objects can be grouped into a complicated graphical object. They can be further stored in a graphical object library. For example, certain button shapes are frequently used; like Go, Quit and others. A copy of them can then be kept as a Button library so that later they are ready for use in instances as shown in figure 4.

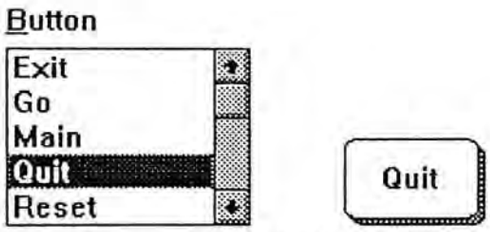


Figure 4 Graphical Object Library

2.5 Animation Production

Using Pearl, an algorithm is animated in three steps. Firstly, a program representing the algorithm is implemented in C++. Secondly, the animator then models the graphical display intended for the animation. Finally, the program and the display are linked together, thus, producing the animation. Users[1] can interact with the algorithm through direct manipulating the graphical objects in the display (See figure 5).

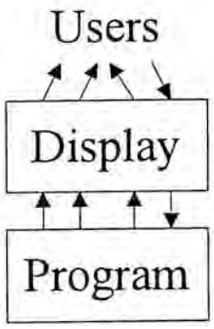


Figure 5 Pearl

A simple example is to animate sorting algorithms. In the *Sorting* animation, several rectangles, which represent a list of items, are to be arranged into order according to their heights. In addition, the number of swap required by each algorithm is to be shown. The process to develop the animation using Pearl will be covered in this section.

2.5.1 Algorithm

An algorithm is specified as a number of object-oriented C++ classes. The classes are templates for the objects in the program. Each class includes both data structure definitions *Data* and the processing code for instances of those data structures *Methods*.

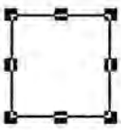
Two classes, *Item* and *SortQueue*, are used to define the sorting algorithms. The class *SortQueue* is the main class, which includes ten objects of the class *Item* and a variable *count* as its data. It also contains several methods including the *Bubble()* for the Bubble sort. On the other hand, the class *Item* comprises two variables *Pos* and *Val*, which represent the position and value respectively, and various operations of them. (See figure 6)

[1] *User* refers to the person watching the animation.

The methods are annotated with additional markers such as the "`Write count;`" in figure 7 to indicate changes of values in the program.

2.5.2 Display

Each of the above classes is represented by a number of graphical objects. The animator draws a rectangle for the class *Item* as well as the text "Number of Swaps :" and "0" for the class *SortQueue*. A rectangle is associated with the attributes *x* and *Length*, which are the horizontal distance to origin and the height respectively. In addition, a text possesses an attribute *Content*, which represents the content of the text. (See figure 6)



Number of Swaps : 0

```
class Item {
   int Pos,Val;
   Domain Pos(0..9),Val(0..100);
public:
   void SetPos(int);
   int GetPos(void);
   void SetVal(int);
   int GetVal(void);
};
```

```
class SortQueue {
   class Item item[10];
   int count,i;
public:
   SortQueue(void);
   void Swap(int,int);
   void Swap(void);
   int Compare(int,int);
   void Arrange(void);
   void BubbleUp(void);
   void Randomize(void);
   void Bubble(void);
   void Insertion(void);
   void Selection(void);
};
```

Figure 6 the classes Item and SortQueue

```
void SortQueue::Bubble(void)
{
   ...
   count = 0;
   Write count;
   for (i = n-1; i > 0; --i)
      for (j = 0; j < i; j++)
         if (compare(j,j+1)) {
            swap(j,j+1);
            count++;
            Write count;
         }
   ...
}
```

Figure 7 the method Bubble

2.5.3 Link

The attributes of the graphical objects are then linked with some variables pertaining to the class. For example, a link is built between the attributes *x* and *Length* of the rectangle and the variables *Pos* and *Val* of the class *Item* respectively (See figure 8). Moreover, the attribute *Content* of the text is bound[2] to the variable count of the class *SortQueue*.

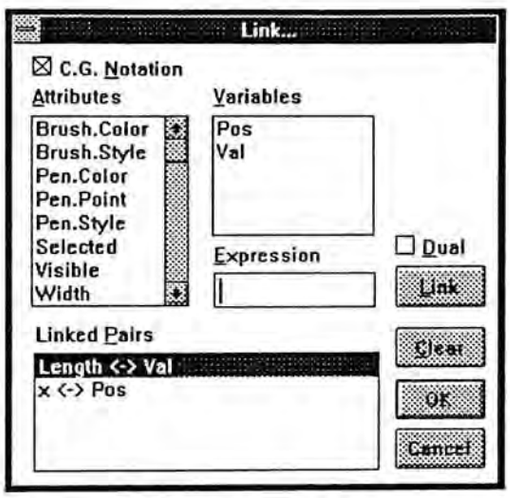


Figure 8 Link

2.5.4 In Action

When the animation is put in action, a user can interactively modify the lengths of the rectangle, which will also automatically change the values of the objects in program. Then, the process to sort the rectangles is shown in sequence as in figure 9, with the longest (shortest) rectangle going to the leftmost (rightmost) side. Meanwhile, the number at the bottom indicates the number of swaps used by the algorithm at that moment.

[2]The terms *link* and *bind* are used interchangeably.

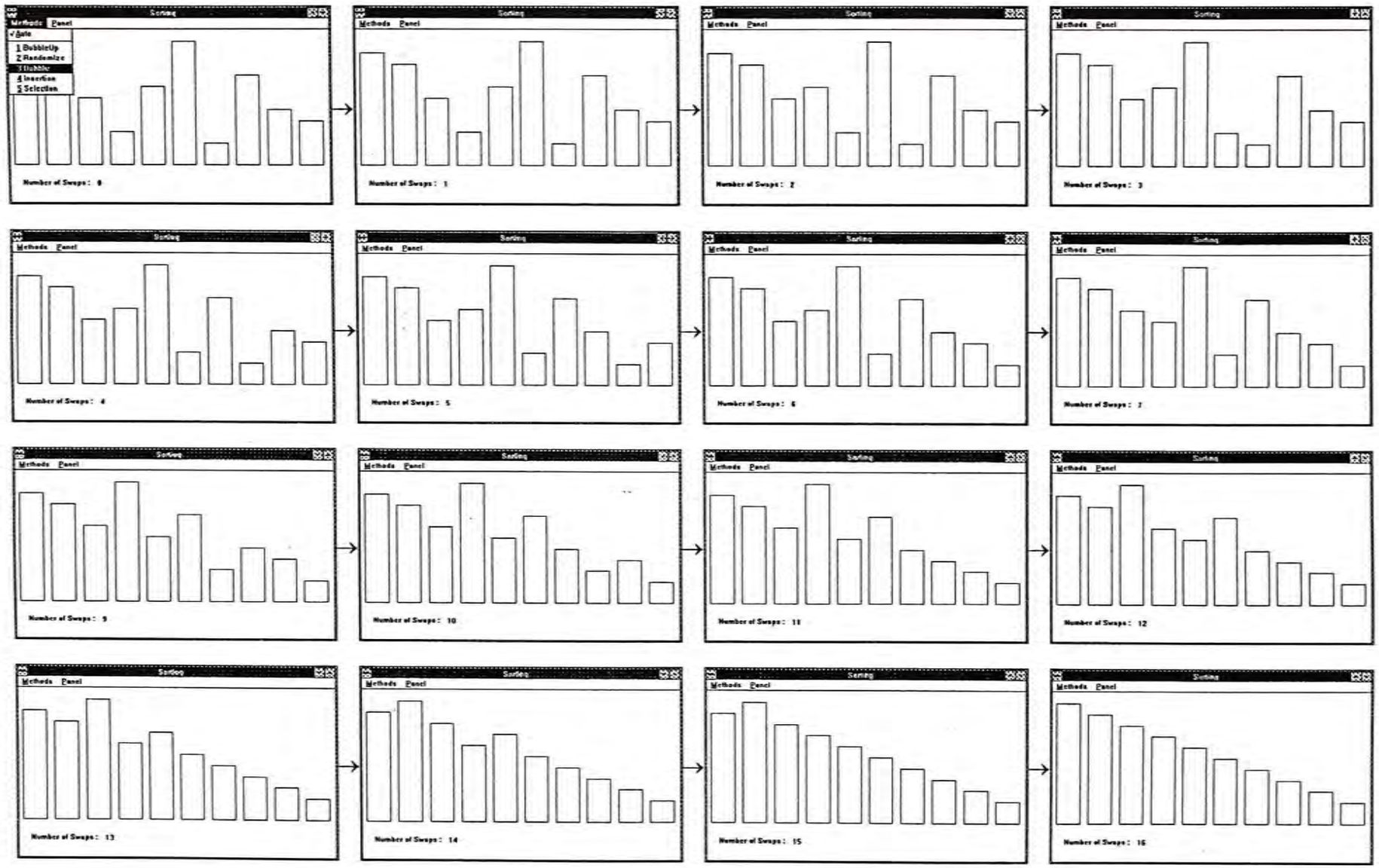


Figure 9 in action

2.6 Actual Presentation

A system, which can only handle graphical objects, is merely a graphics system. A presentation system should not only facilitate preparing the slides but also help the actual presentation. Six features for the actual presentation module are proposed: Story Board, Multi-Windows, Highlight, Zooming, Scribbling Pad and Speaker Reference Notes. They assist presenters in conveying their meaning to the listeners through the screen.

2.6.1 Story Board

Story Board is very useful in presenting. Using it, an idea can be clearly shown in a step by step manner. It works like sequentially stacking several transparencies together. Features on a slide are displayed according to a particular time order. Each time presenters press the mouse button, more (less) information will appear on the screen; just like having flipped one more transparency onto the current transparency. On the contrary, a double click causes the system to go back to the previous instance.

The following are two examples using the technique. The first reveals a list of characteristics of an algorithm. The list is displayed one point after another (See figure 10(a)). The second illustrates how a scan-line algorithm works. In each step, one more dot is drawn (See figure 10(b)).

Furthermore, the system allows another flipping mode, which is to show two consecutive instances concurrently. In other words, the first instance is displayed accompanied with the second instance; and then, the second accompanies the third and so on so forth. This mode is used for comparing the changes between two instances. An example is to show the steps to optimize a query tree. Each time, the trees before and after a certain change are displayed side by side (See figure 10(c)).

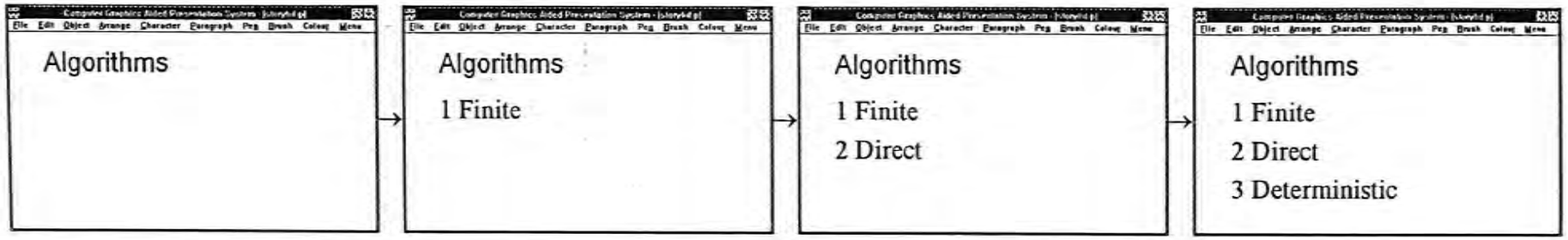


Figure 10(a) Story Board

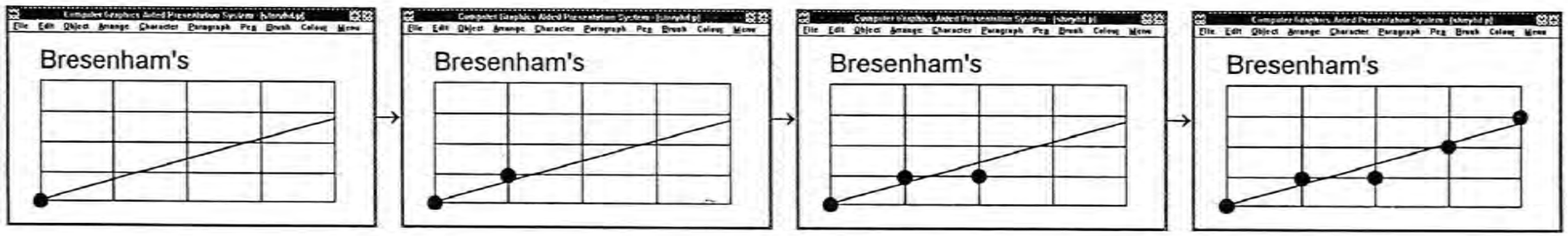


Figure 10(b) Story Board

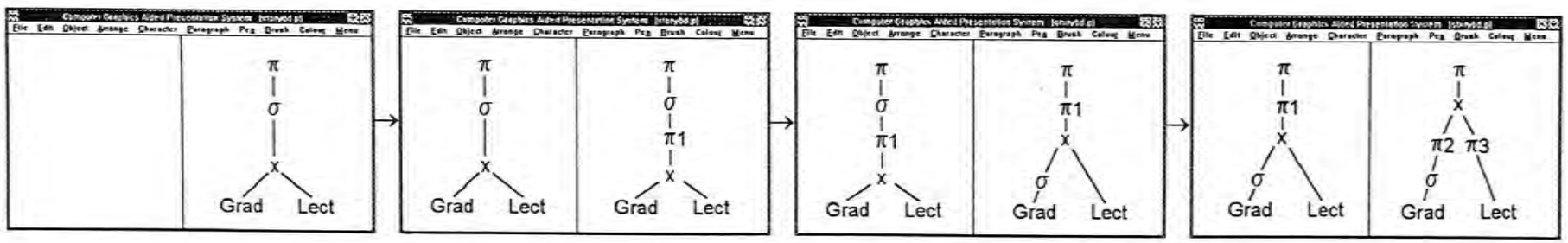


Figure 10(c) Story Board

2.6.2 Multi-Windows

Sometimes, presenters need to display more than one slide at the same time. This can be easily achieved because each window can represent the contents of a different slide. An example to show the results of a computation in form of a graph on one slide while the mathematics on another as in figure 11.

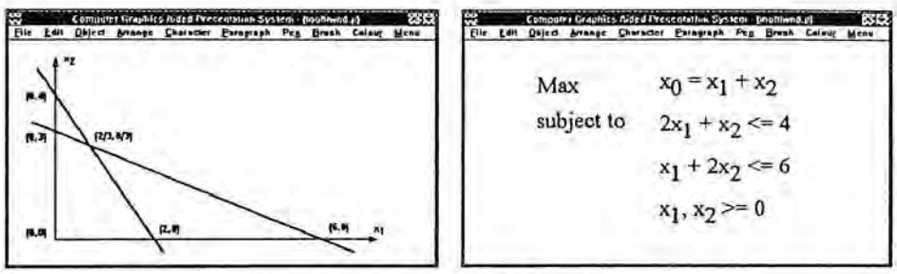


Figure 11 Multi-Windows

However, too many windows opened will mess up the screen. There are functions to arrange the windows, like the *cascade* and *tile* supported by the Windows system. In addition, the total number of windows for slides is limited.

2.6.3 Highlight

It is sometimes necessary to highlight certain area of a slide. It is just like many presenters cover part of the transparency with a paper. The system achieves this by maintaining an area of interest bright, while the region outside the area will be grayed, thus leaving only the most important feature to be seen. For example, the current point can be given prominence (See figure 12).

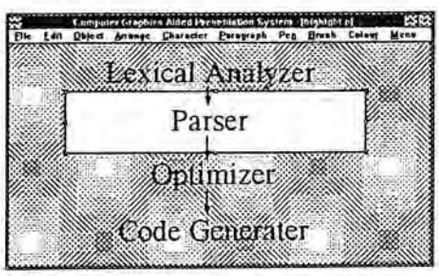


Figure 12 Highlight

2.6.4 Zooming

Zooming is especially useful when presenters have to discuss a very large diagram in the class. Each slide can be zoomed in (out) to cause the objects to appear nearer (further). Presenters can also open two windows for the same slide. One shows the complete picture and the other reveals the region zoomed in. Displaying each small portion of a large data flow diagram in detail is an example (See figure 13).

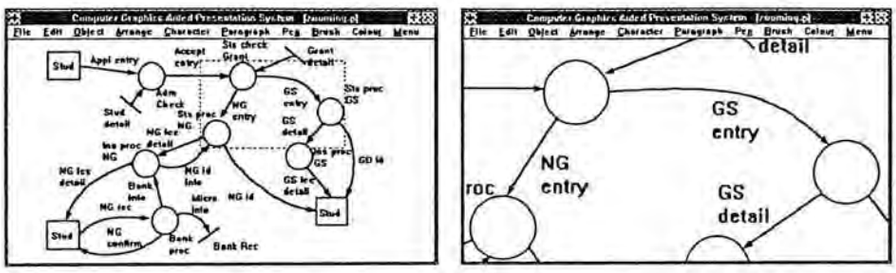


Figure 13 Zooming

Moreover, supposing that the text on the slide is found being too small when projected on the screen, zooming can sometimes alleviate the problem. The presenters can then take a close-up, which will show the subject on a larger scale.

2.6.5 Scribbling Pad

Scribbling Pad is an area, where presenters can draft their rough works. The presenters are able to draw or erase any figures using the tools provided for preparing slides. The scribbling pad window can also duplicate the features on the current slide so that the presenters can update these features in the scribbling pad without interfering the original slide. It works like people write on a blank transparency placed on the current transparency.

2.6.6 Speaker Reference Notes

Presenters often use clue cards to jot down annotations for reference. The system also allows the use of Speaker Reference Notes. The Notes appear as a small button on the slide. When it is pressed during presenting, a small text box will pop up revealing the remarks. Once the button is released, the box vanishes at once.

2.7 Conclusion

A Computer Graphics Aided Lecture Presentation System has been designed and prototyped. Above all, the system is tailor-made for classroom presentation. From preparing a lecture to actual presentation, various features of the system are mentioned. Besides, a novel animation production model has been put forward and implemented for evaluation. To animate a presentation, users model graphical objects using interactive graphics and control the motions of the objects with procedural programming. The details of the animation subsystem will be given in the following chapter.

3

Algorithm Animation Subsystem

3.1 Introduction

Program Visualization is the use of graphics to illustrate some aspects of the program or its run-time execution [16]. Program Visualization systems are then classified according to whether they illustrate the code, data or algorithm of the program, and whether they are dynamic or static. The form of Program Visualization showing the operation of an algorithm in action is called Algorithm Animation.

Picturing abstractly how algorithms operate, Algorithm Animation makes the algorithms more understandable. Especially, Algorithm Animation is worthy of helping students gain insight into the intricacies of the algorithms.

Being presented in this chapter, the most important work of this thesis is to develop a model which facilitates the production of animation; particularly for animating the working of algorithms. To justify the conceptual model, an interactive algorithm animation system Pearl is designed and implemented. In addition, numerous well-known algorithms are animated; serving as examples of using the system.

The Pearl system can be characterized by three special qualities. Firstly, the system let users interact with algorithms so that they can explore the properties of the algorithms. Interactive animation is distinguishable from other algorithm animation, in which users tend to participate passively. Secondly, applying the Direct Link Library technique separates the algorithms from their graphical displays. Hence, changing an algorithm will be unnecessary even though more than one representations of the algorithm are built. Lastly, the use of self-contained C++ class makes commonly-used data classes reusable. Animating different algorithms on the data classes can then be reused by instantiation.

Next section gives an review on related work. Succeeding sections describe the four components: Algorithm, Display, Link and Options respectively. Section 3.7 describes the examples of animated algorithm. This chapter is briefly summarized in the last section.

3.2 Related Work

Previous work on Algorithm Animation has a direct influence upon the present research. To use animation to illustrate algorithms, a number of films were produced at the end of '70s. With the advance of Computer Graphics, researchers have built several innovative systems [17-41]; namely, BALSA, Animus, Stills and Movie, ALADDIN, TANGO and Pavane. Having been formally evaluated, some of the systems have tentatively realized the great potential of Algorithm Animation. Their features are summerized in the following sub-sections.

3.2.1 BALSA, BALSA II and Zeus

BALSA [17-21] from the Brown University is one of the best known algorithm animation systems. It has been extensively used for teaching in laboratory. The platform for BALSA is an Apollo personal workstation; whereas an updated version, BALSA II, runs on Macintosh. Using BALSA, the algorithm designer first identifies the *interesting events* in an algorithm, which should lead to changes in the image being displayed. The animator then implements the graphical views, which maintain the image and change it in response to the interesting events. Input generators are implemented to provide data for the algorithm to manipulate. Finally, sequence of commands to present the algorithm in action are stated as scripts. In addition, the latest system, Zeus [22][23], focuses on the use of color and sound to convey how an algorithm operates.

3.2.2 Animus and Gestural system

Animus [24][25] has been built upon the ThingLab, which is a constraint-oriented simulation system based on Smalltalk. *Temporal constraints* are declared by the users; specifying that certain events in the underlying program should trigger the occurrence of some complex sequence of graphical responses. The dynamics of users' gestures are captured as the desired graphical responses [26]. In addition, relative timing of the gestures is controlled by a music-like score editor.

3.2.3 Stills and Movie

Stills and Movie [27] is a simple algorithm animation system. A C program is first augmented with several `printf` statements to generate a *script* file describing the results of the execution of the program. The script file is then processed by the program Stills to produce static pictures, which can be included in troff documents. Furthermore, the same script file can be dynamically displayed by the program Movie. Using the script file, the animation can be proceeded forward or backward at different speed.

3.2.4 ALADDIN

ALADDIN [28] allows a graphical specification of the animation. The specification is composed of three steps. A *graphical type* is first interactively defined using a catalog of graphical objects. A *graphical variable* belonging to a certain graphical type is then declared for each occurrence of the graphical objects. Finally, an *animation statement* changes the appearance of the graphical object with the execution of the program. The three components can be compared with the type definition, variable declaration and statement in conventional programming.

3.2.5 TANGO and DANCE

TANGO [29][30][31] has introduced a path-transition paradigm for Algorithm Animation. Using the paradigm, users describe an animation in a program through manipulating four abstract data types; namely, the graphical *images* on the screen, the *locations* that images and other objects occupy, the *transitions* that the images make, and the *paths* that modify the images' transitions. The program representing the animation can also be generated by DANCE [32], which is a direct manipulation style graphical editor for designers to sketch out animation scenarios.

3.2.6 Pavane

Pavane [33][34] treats visualization as a mapping from the state of a program to the graphical representations. In this way, five levels of abstraction as applied to Algorithm Animation have been identified. Firstly, *direct* representations map some aspect of a program directly to a picture. Secondly, *structural* representations encapsulate certain extraneous data so as to simplify the view of the program. Thirdly, *synthesized* representations derive information that is not explicitly present in the program. Fourthy, *analytical* representations attempt to capture more abstract properties of the structure or behavior of the program. Lastly, *explanatory* representations enhance the viewer's understanding through the provision of visual hints.

3.2.7 Other Systems

Apart from the above systems, some systems have traded the flexibility in producing animation for their particular purposes. Keeping the source code of the animated program unmodified [35][36][37], several systems are developed to assist in program debugging and tuning. In addition, certain systems usually referred as Parallel Algorithm Animation systems [38][39] seek to depict the communication among processors for analysis of performance. Other related work can be found in [40][41].

Early algorithm animation systems are programming-intensive for use. To specify the display, the animators have to make low-level calls to the graphics primitives. Not only rapid prototyping is not allowed but also learning to operate the systems takes much more time.

Employing interactive graphics techniques, newer or latest versions of the systems begin to emphasize the ease of use in the specification of the display. There are systems using direct manipulation for animators to build the desired animations by demonstration. However, we are unaware of any other work, which allows user actions such as click and drag to influence the underlying computation of the algorithms. This has motivated the development of the Pearl system.

3.3 Algorithm

As mentioned in section 2.5, an algorithm is specified as various C++ classes. Each class is composed of data and methods. The methods are designated with additional markers to signal changes of values in the program. The methodology described in this chapter can easily be adapted to any high level language used for specifying the algorithms to be animated.

3.3.1 Markers

It was intended to keep the animated version of the program as intact as possible. However, our experience is that eliminating fundamental operations such as read and write will, on the contrary, degrade the readability of the program. Eventually, four kinds of markers are chosen -- *Domain*, *Read*, *PreWrite* and *Write* as follows:

3.3.1.1 Domain

The use of *Domain* (See Figure 14) is to define the range of a variable, which will be required to establish a link. The animator can either specify the extremes of the range or explicitly state each values in the range; namely, "`Domain Pos(0..9);`" and "`Domain Prime(2,3,5,7);`" respectively.

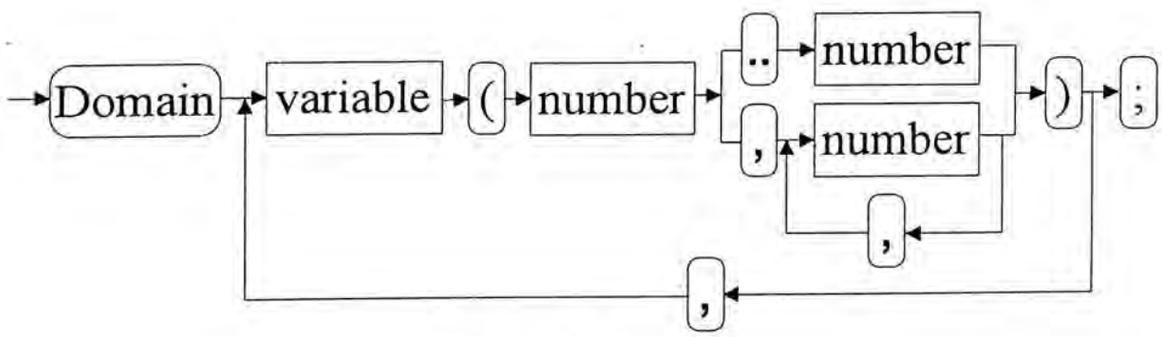


Figure 14 Domain

3.3.1.2 Read, PreWrite and Write

Read, *PreWrite* and *Write* markers (See Figure 15) are self-explanatory. A *Read* in a program feeds in the values of some variables from the attributes of the bound graphical objects. On the contrary, a *Write* sends out the values of the variables to the attributes of the graphical objects, thus, causing an update on the display.

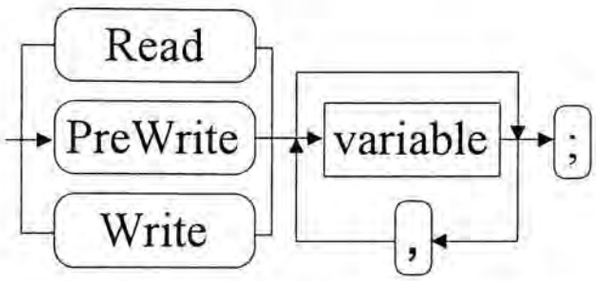


Figure 15 Read, PreWrite and Write

A *PreWrite* also gives out values of the variables. Contrary to a *Write*, a *PreWrite* only dispatches values to a buffer before modifying the display. Ultimately, the values, which may result from several *PreWrite*'s, will be simultaneously delivered to the display when a *Write* is performed.

An animator can make use of the *PreWrite* and *Write* to arrange the relative timing of different motions. An example is the swapping of two rectangles in the animation *Sorting*. For the case of using *Write*'s, the two rectangles will move one by one. However, when *PreWrite*'s are used, the rectangles will be in motion concurrently. (See figure 16)

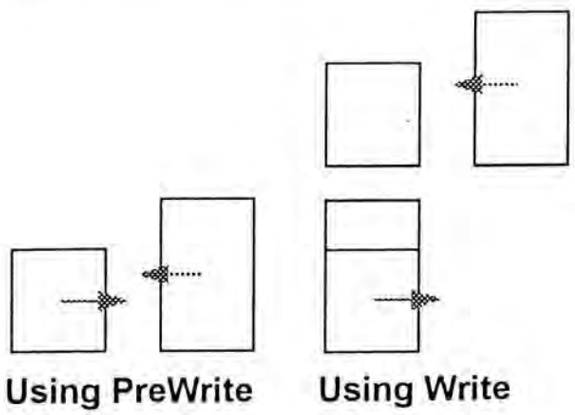


Figure 16 relative timing of motions

3.3.2 Dynamic Linked Library

The program is to be pre-processed, which converts the markers into actual C++ codes. The codes are then compiled by an external compiler into a Dynamic Link Library DLL, which will be eventually loaded by the system during run time. (See figure 17)

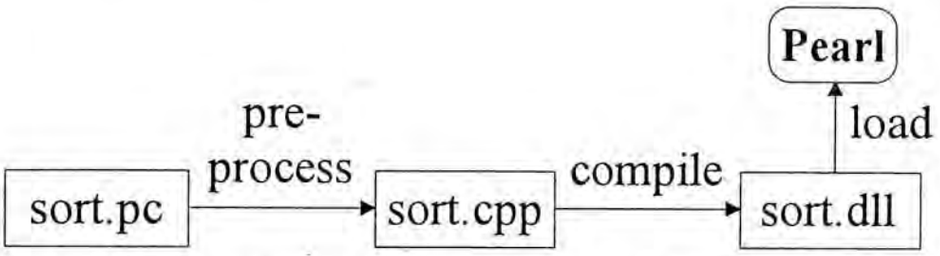


Figure 17 producing a DLL

3.3.3 Animation Independence of Algorithm

The algorithm specified in Pearl is animation independent, which means that the program is not specific to a particular display. Modifying the graphical objects, as well as the links, will not even require re-compiling the source program.

For example, in animation *Sorting*, the rectangle representing the class *Item* is now replaced by a text. The attributes *x* and *Content* of the text are then linked with the variables *Pos* and *Val*. Therefore, another representation with the items expressed as numbers is produced as shown in figure 18.

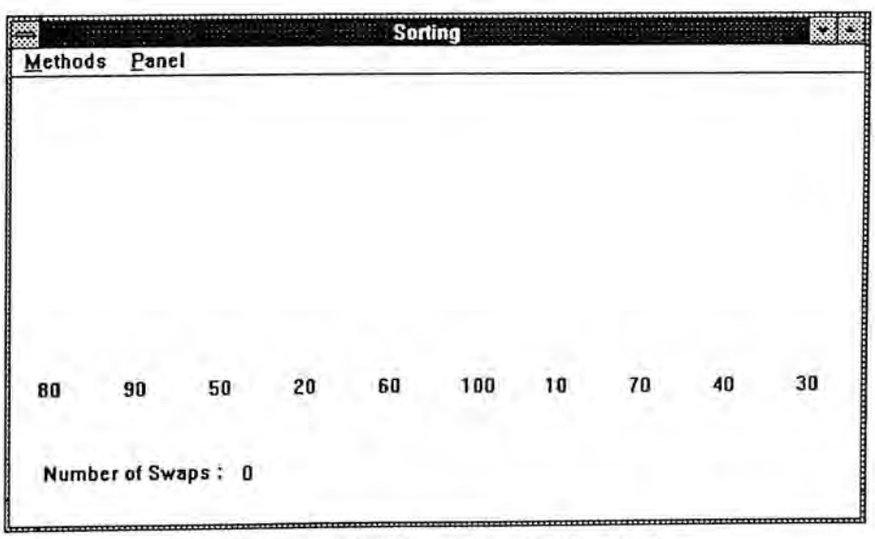


Figure 18 Sorting with text

3.3.4 Reusability

Object-oriented C++ programming encourages reusability. We have implemented several widely-used data classes; namely, *Graph*, *Stack* and *Queue*, which are also graphically represented in Pearl. Other programmers can later make use of these programs to produce animation at ease.

3.4 Display

Each class in the program is represented by various graphical objects. When an instance of the class is created (destroyed), the construction (destruction) of the graphical objects will take place on the display. The attributes of the graphical objects in display are described below.

3.4.1 Attributes

Available graphical objects include text, rectangle, rounded rectangle, ellipse, bitmap, line and bezier curve as shown in figure 3. They possess certain attributes, which are categorized as the *Position*, *Appearance*, *State* and *Content* groups. Each group is summarized in the following sections.

3.4.1.1 Position

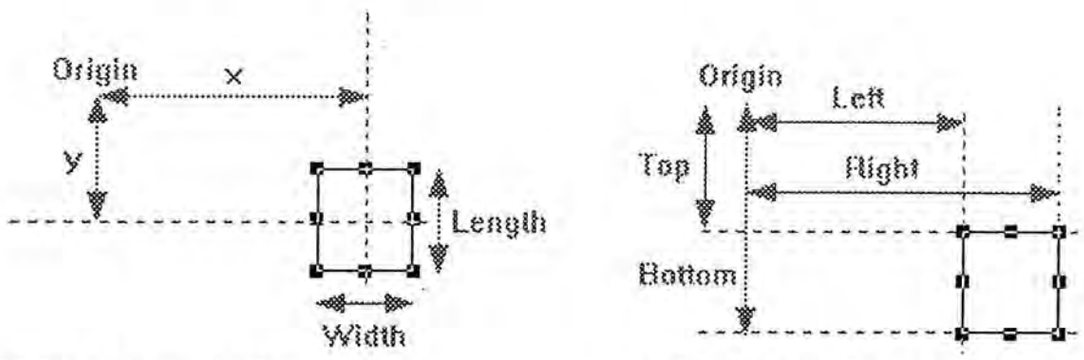
Availability of the position attributes for each graphical object is tabulated in table 1.

	Graphical Objects			
Attributes	Text	Rectangle Rounded Rect Ellipse	Bitmap	Line Curve
x	†	†	†	†
y	†	†	†	†
Width	†	†	†	†
Length	†	†	†	†
Left	√	√	√	
Top	√	√	√	
Right	√	√	√	
Bottom	√	√	√	
Head.x				√
Head.y				√
Tail.x				√
Tail.y				√

A mark indicates those attributes possessed by the corresponding graphical objects.
† available for Center-of-Gravity notation only, √ otherwise.

Table 1 Position

An animator can describe a position using the Center-of-Gravity (CG) notation or not. Using the CG notation, the animator will specify the bounding rectangle of a graphical object with four attributes *x*, *y* (which are the coordinates of the center of gravity of the rectangle), *Width* and *Length*. On the contrary, not using the notation, the animator can state the attributes *Left*, *Top*, *Right* and *Bottom* of the rectangle (See figure 19).



Using CG notation **Using non-CG notation**

Figure 19 CG notation

Both notations are necessary. For example, in figure 20, the animator has created one more rectangle, inside which items are known to be not yet sorted. The rectangle is defined in the class *SortQueue*. The non-CG notation attribute *Right* of the rectangle is bound to a variable, which will be decreased after each pass in the bubble sort algorithm. On the other hand, as a comparison, the attribute *x* of CG notation is instead used for the rectangles representing the class *Item*.

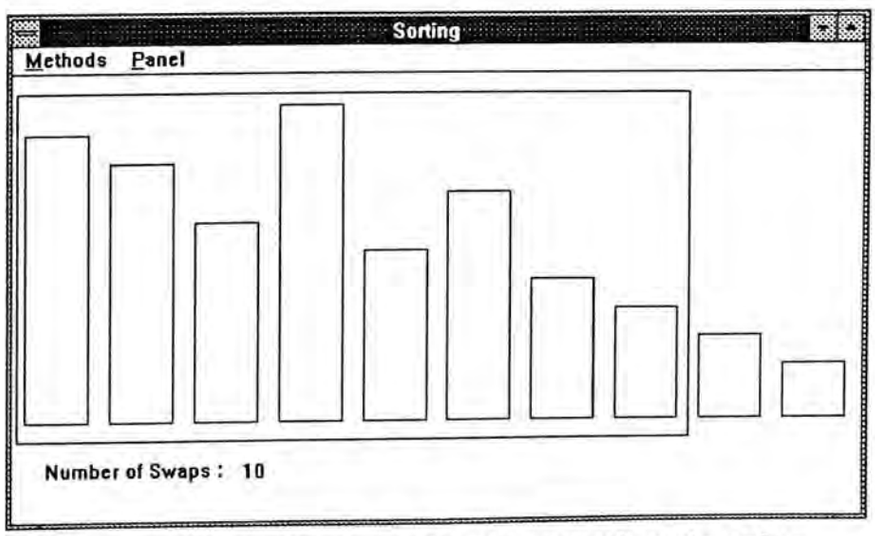


Figure 20 indicating the unsorted rectangles

Besides this, choosing the item *Record* in the menu can reveal the current position of a selected object as shown in figure 21. The range in which the values of the position attributes sketch can further be marked, which will be used later for linking.

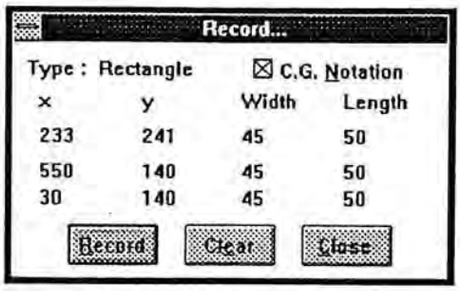


Figure 21 Record

To build a link, it is essential to state the range of an attribute and the range of a variable so as to compensate for the difference between them for precise mapping. For example, the attribute *x* sketches from 30 to 550; while the variable *Pos* lays between 0 and 9 (See figure 22). Moreover, tweening will be carried out to smooth the motion.

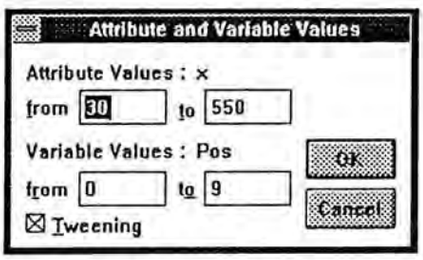


Figure 22 linking Position attribute

3.4.1.2 Appearance

Appearance attributes are summarized in table 2. In addition, the linkings of some of them are displayed in figure 23. Animators can update the values inside the boxes, which are equal to the values of the linked variable in run-time, leading to the particular appearance of the graphical objects. For example, linked as shown in figure 23(a), the line will be dotted if the variable *i* equals 2.

Attributes	Graphical Objects			
	Text	Rectangle Rounded Rect Ellipse	Bitmap	Line Curve
Head.Arrow				√
Tail.Arrow				√
Pen.Style		√		√
Pen.Point		√		√
Pen.Color		√		√
Brush.Style		√		
Brush.Color		√		

Table 2 Appearance

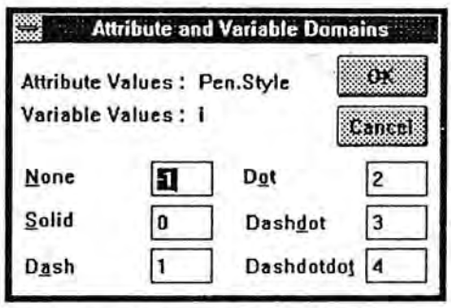


Figure 23(a) Pen Style

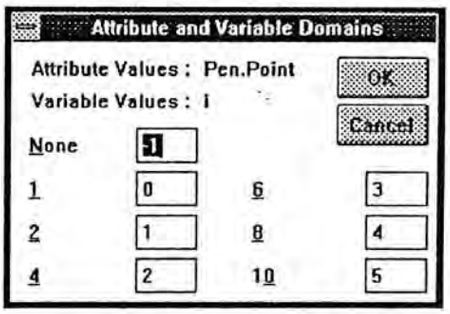


Figure 23(b) Pen Point

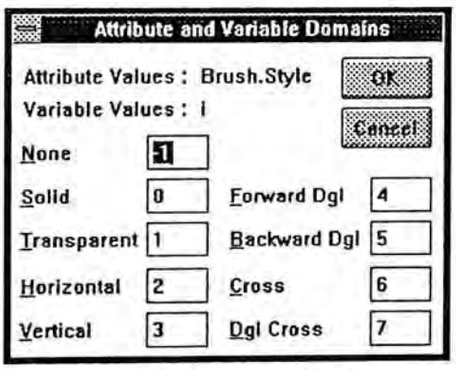


Figure 23(c) Brush Style

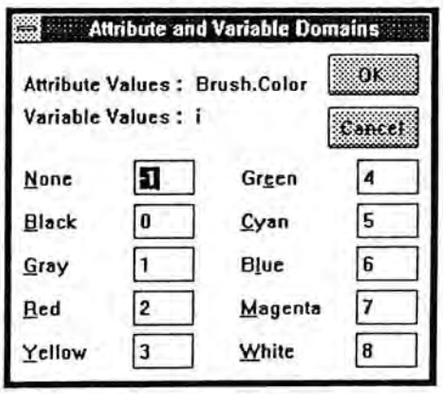


Figure 23(d) Brush Color

Figure 23 linking Appearance attributes

3.4.1.3 State

All graphical objects possess state attributes as shown in table 3.

Attributes	Graphical Objects			
	Text	Rectangle Rounded Rect Ellipse	Bitmap	Line Curve
Selected	√	√	√	√
Visible	√	√	√	√

Table 3 State

Attribute *Selected* refers to the state whether a graphical object is selected by the user. It is frequently used to indicate those graphical objects to be attached in invocation of a certain method. An example is the method *BubbleUp()*, which will bubble up those rectangles selected. The attributes *Selected* of the rectangles have been bound to some variables indicating whether a move should be made to the rectangles (See figures 24 and 25).

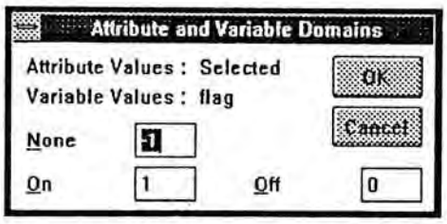


Figure 24 linking State attribute

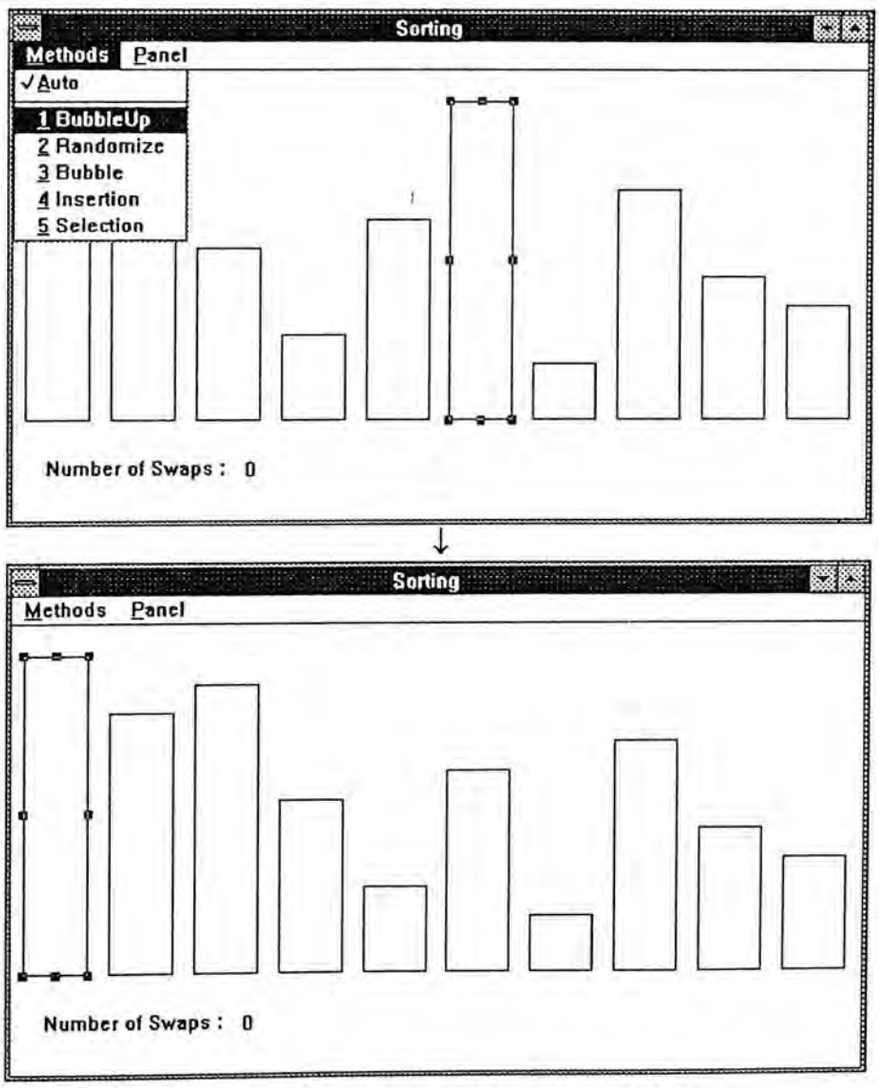


Figure 25 the method BubbleUp

3.4.1.4 Content

Being linked with the attribute *Content* of a text (See table 4), the value of a variable can be depicted; namely, the variable count in the animation *Sorting*. Beside this, the animator can pre-define several strings for a text, which will be displayed depending upon the variable as shown in figure 26.

	Graphical Objects			
Attributes	Text	Rectangle Rounded Rect Ellipse	Bitmap	Line Curve
Content	√			

Table 4 Content

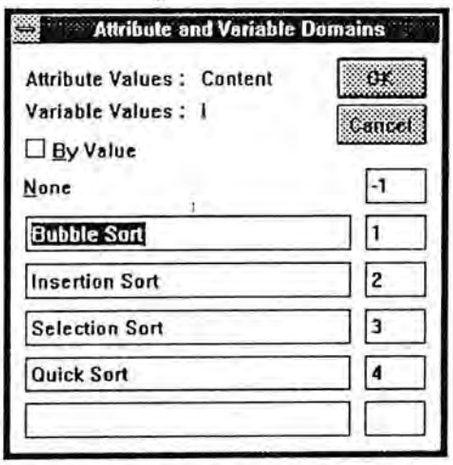


Figure 26 linking Content attribute

4.1.5 Attributes for Group Objects

A graphical object can be a group object, which is composed of several component objects. The attributes of the group object contain all of the attributes of the component objects.

A *Write* applied to a group object is in turn passed on to all of the component objects. A *Read* attempts to retrieve the value but is sometime undefined if the attributes of the component objects are not the same. In this case, the *None* value is used. For example, in figure 23(d), the variable i will obtain the value -1 when the linked group object is of more than one color.

3.4.2 Background

Background classes can be displayed to assist in the layout of the graphical objects in the current class.

3.5 Link

Each object in the program is depicted as one (or more) graphical object. Figure 27 shows the execution of the animation *Sorting*.

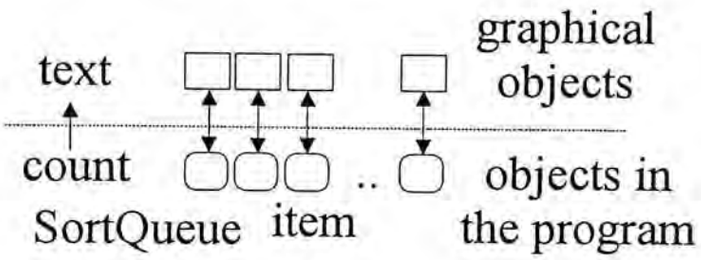


Figure 27 the execution of Sorting

3.5.1 Linking

As shown in figure 8, the attributes of a graphical object are bound to some variables declared in a class with the following characteristics.

3.5.1.1 Duality

As compared with the notions of call-by-reference and call-by-value, a link can be either dual or non-dual. Both *Read* and *Write* are allowed for a dual link; while only *Write* can have a non-dual link.

To ensure correctness, no two attributes of the graphical objects can be dual-linked to a single variable. Otherwise, a *Read* for the variable is indeterminable when the two attributes have different values.

3.5.1.2 Expression

Not only a variable but also an expression can take part in a non-dual link. For example, the attribute *y* of the rectangle is bound to the expression *100-Val/2* as illustrated in figure 28. After being re-sized by the user, the rectangles are to be arranged neatly again (See figure 29).

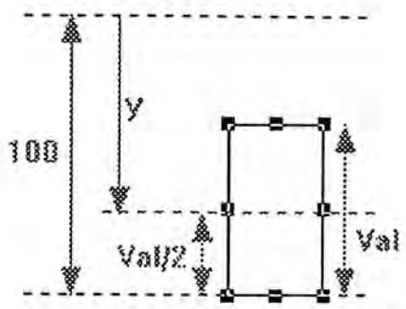


Figure 28 the expression 100-Val/2

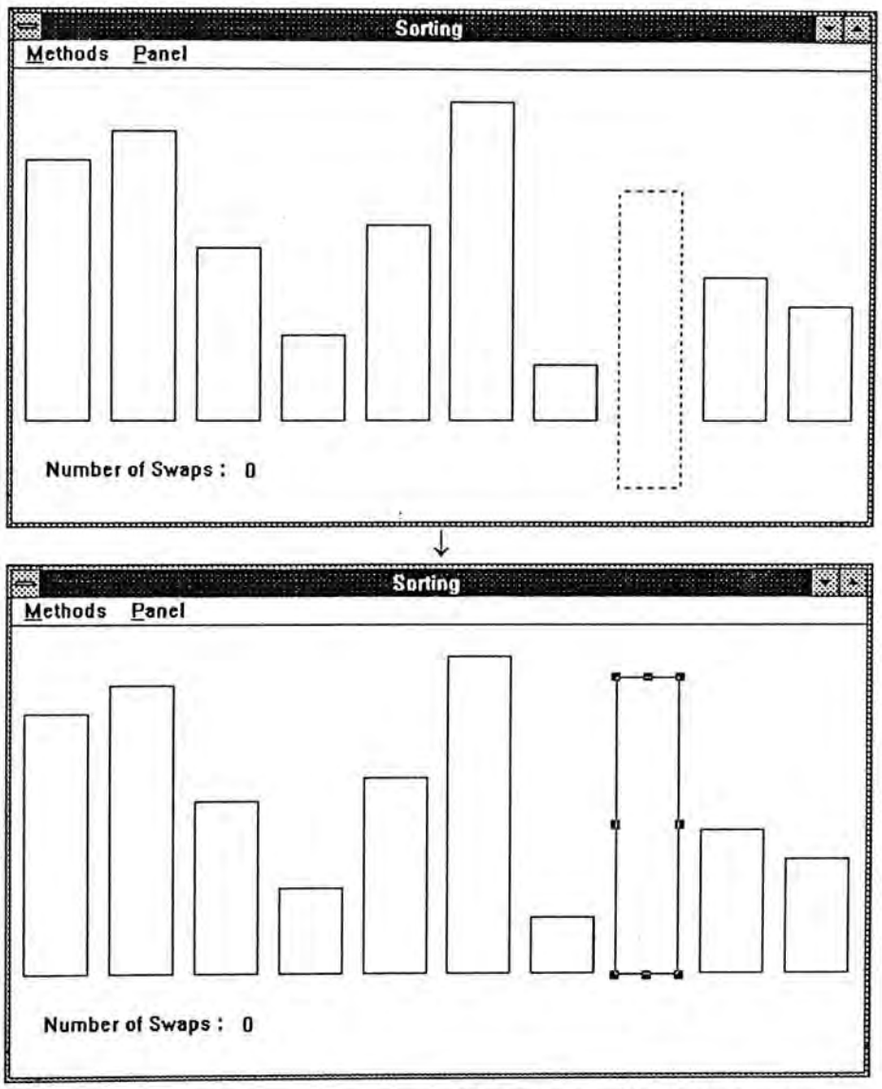


Figure 29 rectangles aligned to the bottom

To make linking an expression easy, the system will automatically compute the range of the expression from the ranges of the variables involved in the expression.

3.5.1.3 Constant

Certain attributes cannot be changed by the users, which can be accomplished by linking the attributes to a constant, usually zero. For example, in figure 30, the animator keeps all the attributes y of the rectangles the same. Even when dragged out of place, a rectangle will be brought back to the right position.

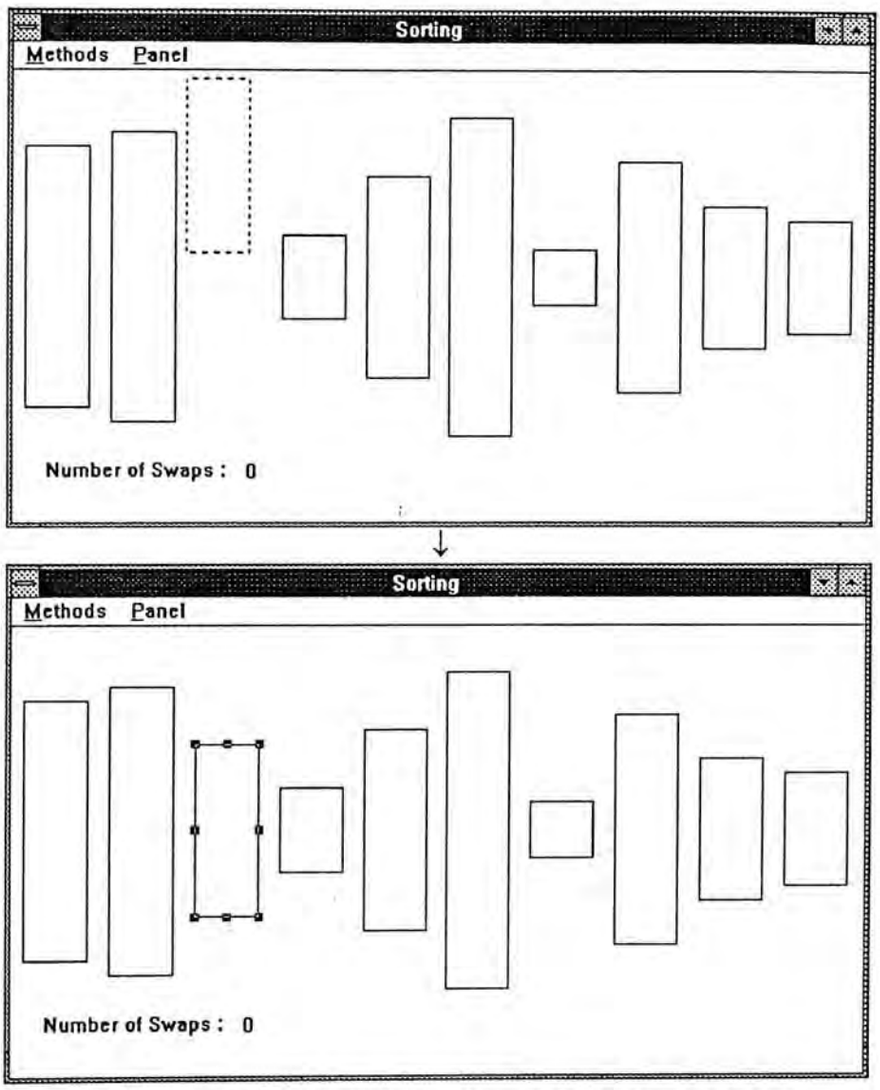


Figure 30 rectangles aligned to the center

3.5.2 Interactive Animation

The role of the users in most algorithm animation is passive. Instead, Pearl enables users to explore certain properties of an algorithm on their own through trying different input data upon the algorithm. For example, in the animation *Sorting*, it is not hard to find out that the number of swaps Bubble Sort requires will be exactly equal to the number of out-of-order pairs of items to be sorted[1].

3.5.3 An Alternative Approach for Sorting

The current approach to produce the animation *Sorting* does not allow users to save the execution state of the animation; with each rectangle being re-sized to a particular length. An alternative approach makes a difference.

The approach is to create ten rectangles in the class *SortQueue* instead of one general rectangle in the class *Item*. The attributes x of the rectangles are linked one by one to each element of an array *PosArray[10]*. The rectangles can then be saved like other graphical objects modeled for a class.

However, a drawback to the approach is that the animator must make changes to the rectangles one after another if he needs to modify them.

3.5.3.1 Duplicate

To help linking to an array of elements, a graphical object can be duplicated with the indices of the linked variables advanced each time. For example, a rectangle is first created with attribute x bound to the variable *PosArray[0]*. The rectangle is then duplicated as shown in figure 31; yielding nine more rectangles with attributes x bound to *PosArray[1]*, *PosArray[2]* and so on so forth.

[1]More interestingly, this leads to the proof that the average number of swaps used by Bubble Sort is n(n-1)/4 where n is the number of items.

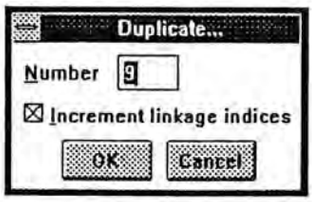


Figure 31 Duplicate

3.5.4 Exercise

To practice the links built in a class, the animator can invoke the *Exercise* as in figure 32. The *Exercise* lists out all the variables declared in the class. The animator can assign values to the variables so as to observe the effects upon the graphical objects. In the other way round, the animator can directly manipulate the graphical objects to examine the changes in the variables.

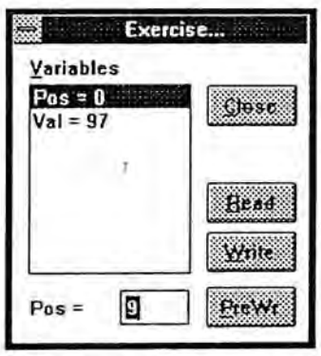


Figure 32 Exercise

3.6 Options

Before putting an animation in action, the animator has to specify various options as shown in figure 33. A class in the program is designated as the entry point of the program. The methods of the class can be either appended to the menu or selected as *Auto* methods. Users can also pause the animation in action. In addition, the smoothness for tweening can be precisely controlled.

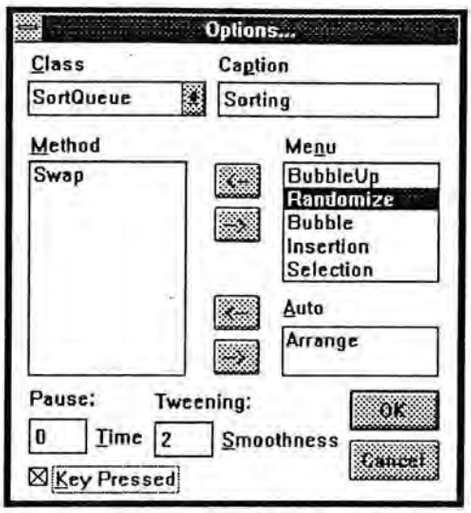


Figure 33 Options

3.6.1 Class

One of the classes defined in the program should be chosen as the *Main* class, for example, *SortQueue* in figure 33. An instance of the class will be created to start an animation.

3.6.2 Method

Methods (without parameter) of the *Main* class can be either inserted into the animation menu or chosen as *Auto* methods. Selecting the corresponding item, the user can invoke the methods added into the menu. For example, choosing the item *Bubble* starts to sort the items using Bubble sort. On the other hand, *Auto* methods like *Arrange()* are activated succeeding each user interaction with the display.

Being enabled, *Auto* methods are used to recover the state of the objects in the program from inconsistency due to user interaction. An example is the *Arrange()*, which prevents users from placing two rectangles to an identical position as shown in figure 34.

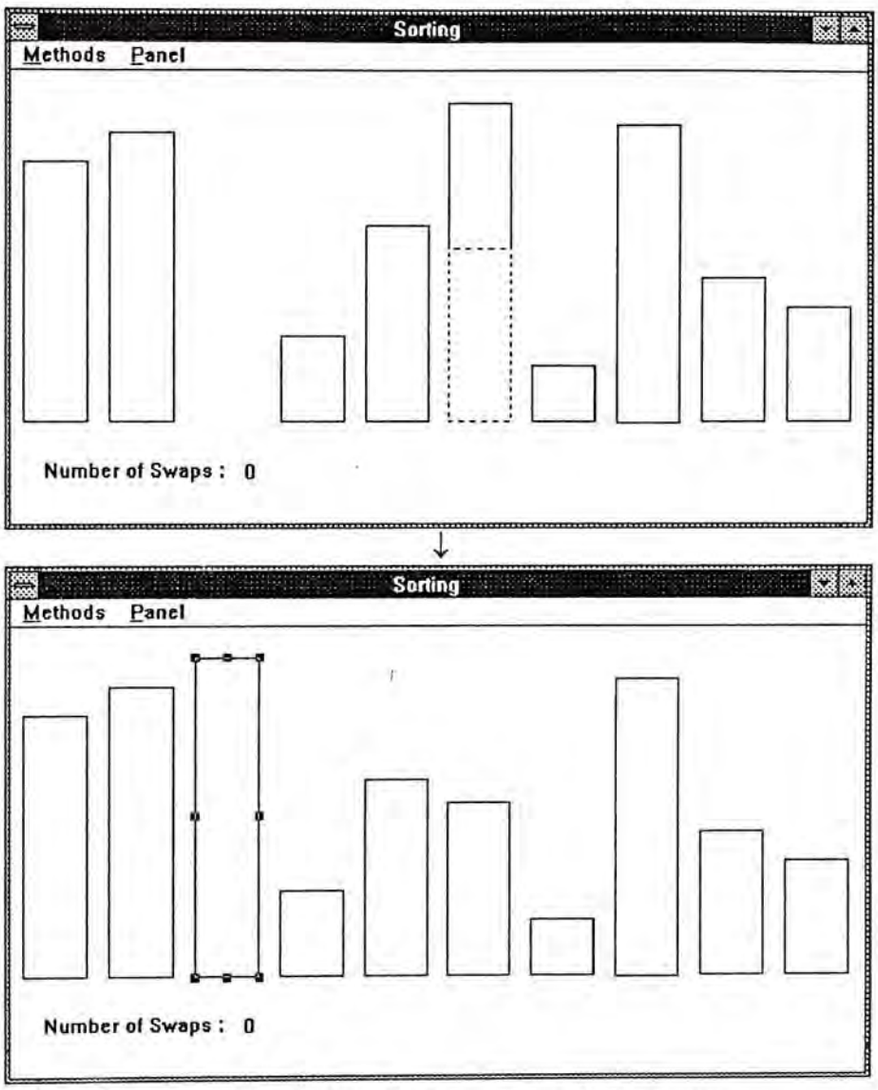


Figure 34 the Auto method Arrange

3.6.3 Pause

An algorithm can be paused after each display update. It can be delayed for a few seconds or suspended until the user choose the item *StepOver* as shown in figure 35. The pause can be enabled or disabled during the animation so that users have full discretion to control the flow of the animation.

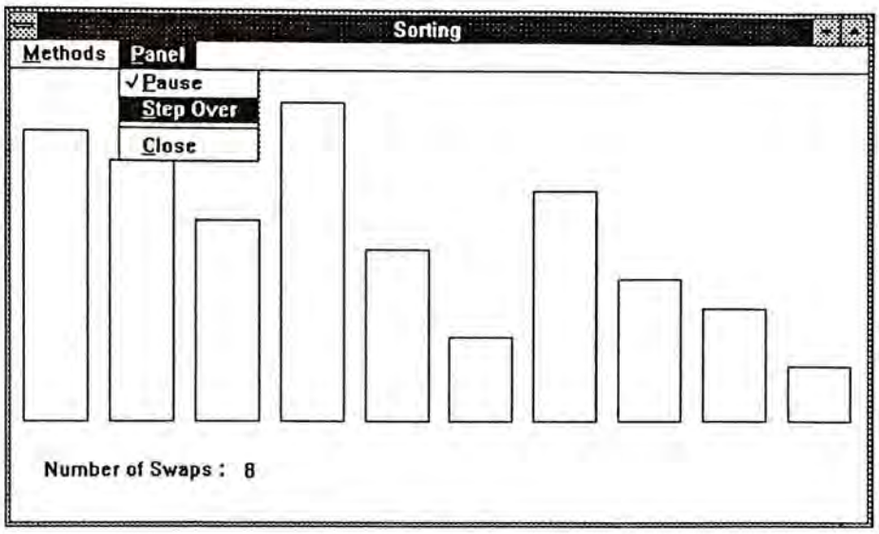


Figure 35 Pause

3.6.4 Tweening

The smoothness for tweening is used to reflect the number of in-between frames required; as illustrated in figure 36. The lower the smoothness (with 1 as the lowest) is used, the smoother the motion can be achieved but the slower is the performance.

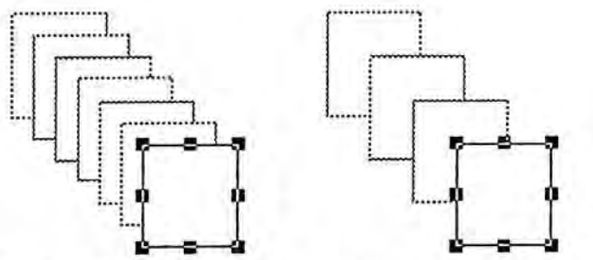


Figure 36 Tweening

3.7 Examples

Six more examples made using Pearl are described below; namely, Stack, Tic-tac-toe, Knapsack Problem, Turing Machine, N Queens Problem and Maximal Matching. Our experience in implementing the examples shows that developing each example takes on average one day to complete.

3.7.1 Stack

On the right hand side of figure 37 is a stack, in which insertions and deletions are always made at the top. To push a selected object, the user chooses the item *Push* from the menu. The object is then projected into the stack. To pop the stack, the top entry is removed.

3.7.2 Tic-tac-toe

The user and the computer alternate in the moves for the game Tic-tac-toe. While looking for the most favorable move for the computer, the system depicts each of the expected move in gray (See figure 38), on the analogy of a max-min game tree. After several level of searching, the board position is associated with a score, which is measured by subtracting the number of open rows, columns and diagonal for the computer by that open for the user. The move leading to the highest score will be made.

3.7.3 Knapsack Problem

Several items are to be packed into a knapsack. Each of the items is given a utility, which as well as the size are modified by the user. The total utility is to be maximized, subject to the size constraint. The optimal solution is found using exhaustive search since the Knapsack Problem is known to be NP-complete. (See figure 39)

3.7.4 Turing Machine

A Turing Machine is consisted of a finite control, a tape and a head that can be used for reading or writing on the tape. The finite control is specified in the form of a table. Each entry of the table denotes respectively the next state, the symbol to write and the move (left or right) with respect to the current state and the tape symbol. In figure 40, the user has stated explicitly the finite control to accept the language $\{ 0^n1^n \mid n >= 1 \}$ and an input of 000111. After several steps of computation, the Turing Machine has left an answer Y (Yes).

3.7.5 N Queens Problem

The N Queens Problem is to place N queens on an N × N board such that no two queens will attack one another. After each queen Q has been placed, improper squares are crossed out as shown in figure 41. Backtracking is used when no available square remains in one of the unoccupied rows, which is indicated as a list of f's (Fail). This method is called Forward Checking.

3.7.6 Maximal Matching (of Bipartite Graph)

A matching is a subset of edges with no two edges incident upon the same node. Started with an empty matching, a matching is improved using several augmenting paths. The augmenting paths alternates with edges included and excluded in the matching and end with excluded edges. To enlarge the matching, the edges in the augmenting path are to be accordingly removed and added to the matching. In figure 42, the matching is represented by heavy edges and the augmenting paths are displayed in gray.

Figure 37 Stack

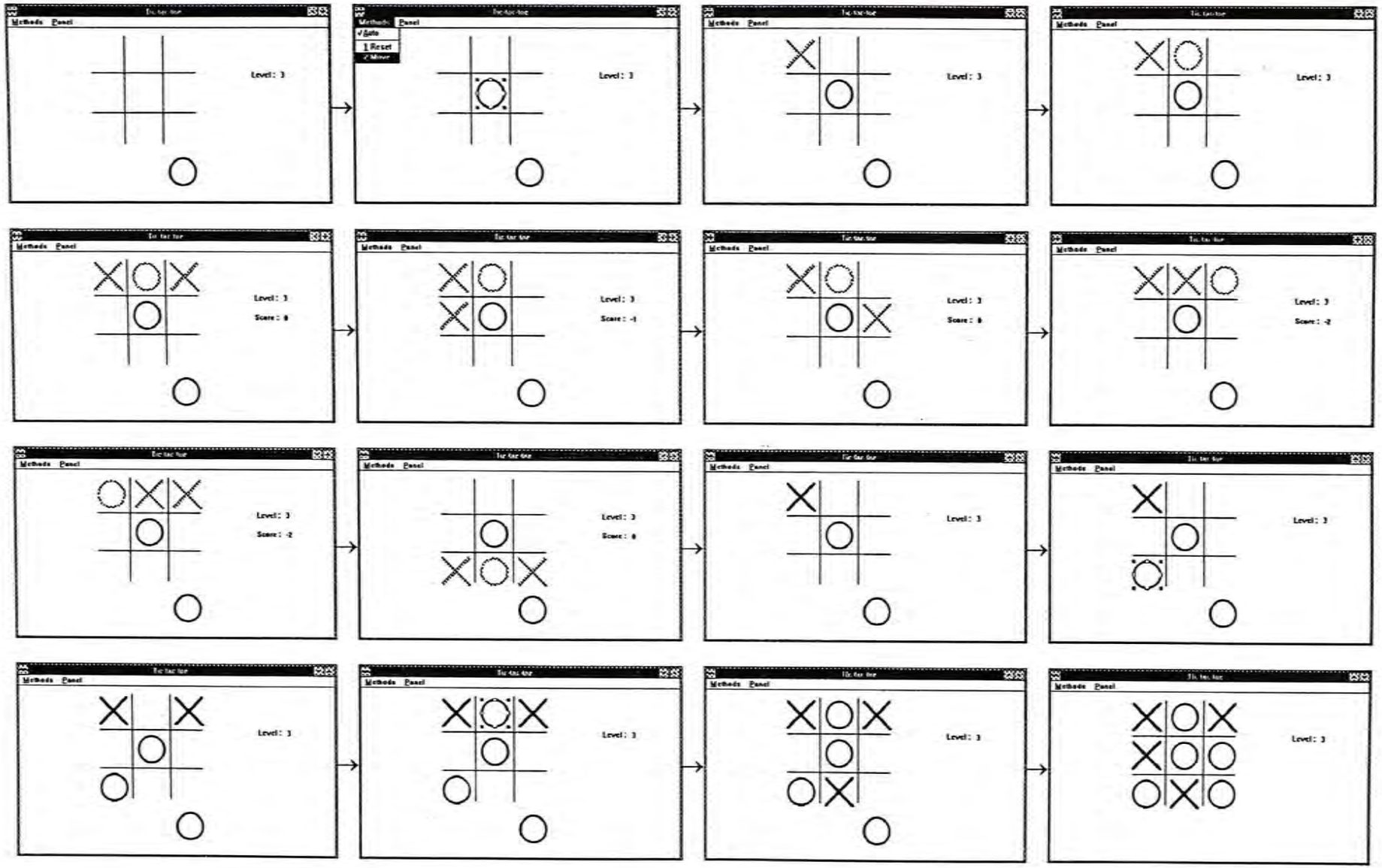


Figure 38 Tic-tac-toe

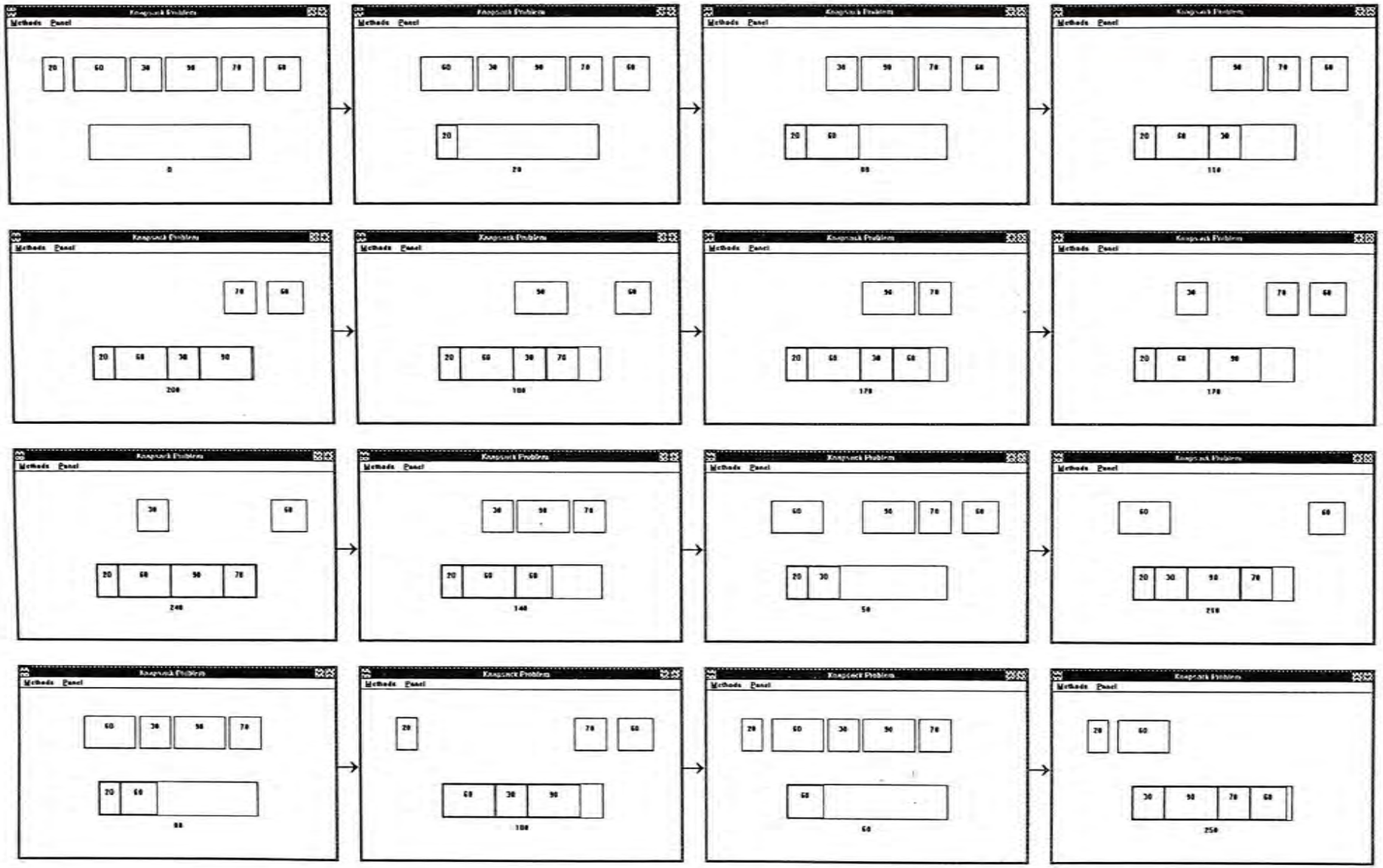


Figure 39 Knapsack Problem

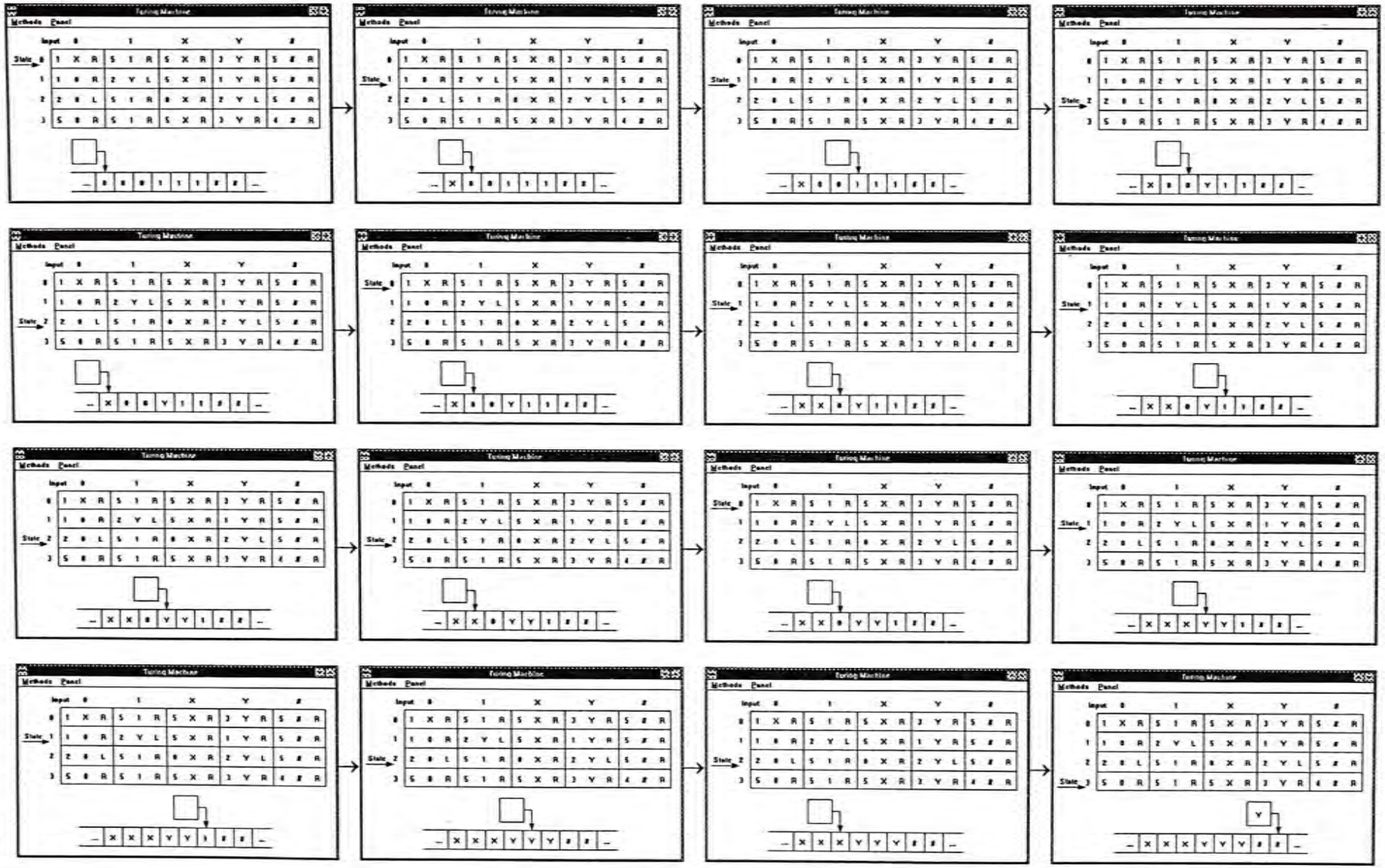


Figure 40 Turing Machine

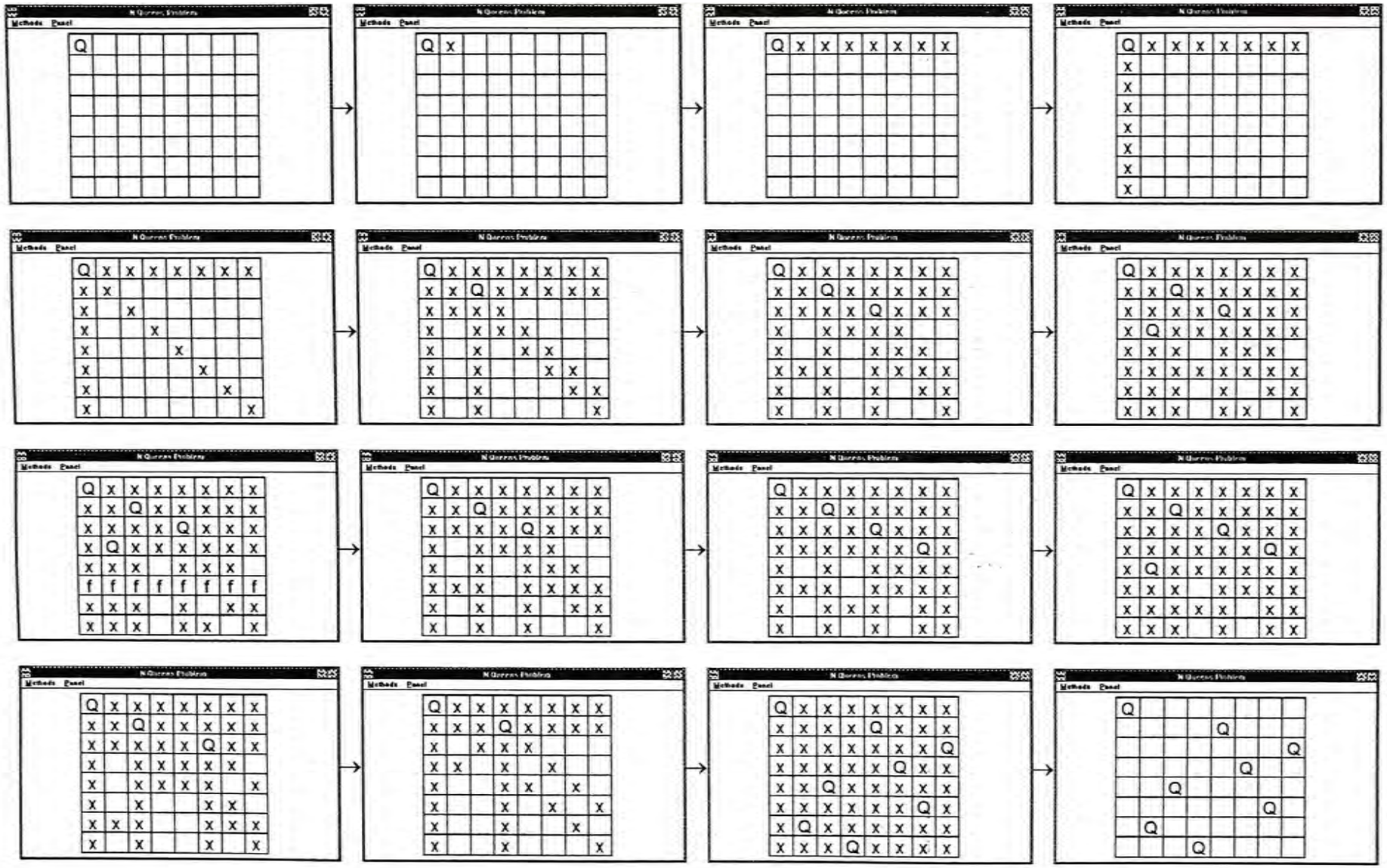


Figure 41 N Queens Problem

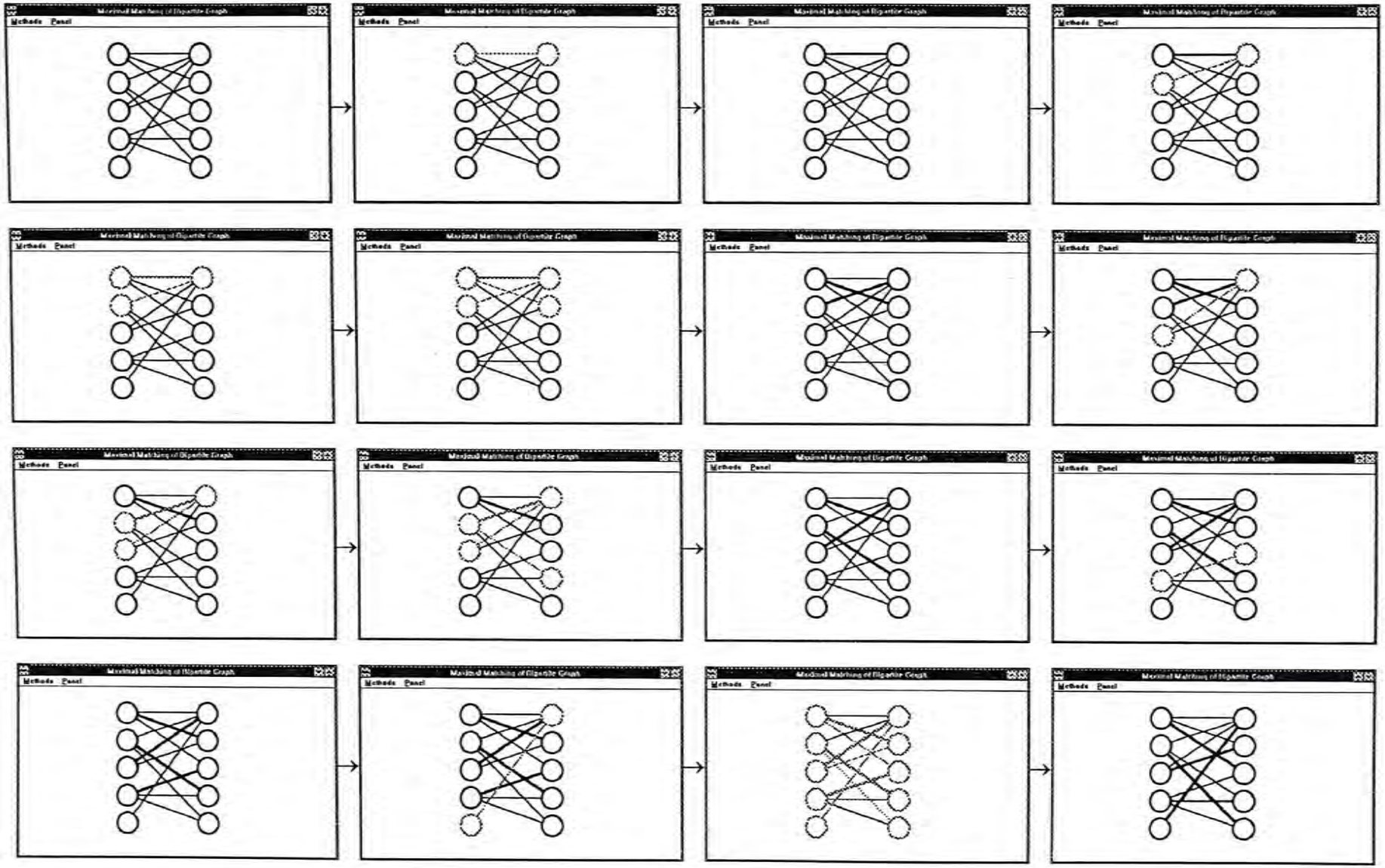


Figure 42 Maximal Matching of Bipartite Graph

3.8 Conclusion

An algorithm animation system Pearl has been designed and developed. In essence, Pearl makes the exploration of animated algorithms to be at one's own pace. Users can acquire a thorough understanding of the algorithms by experimenting with a variety of input upon the algorithms. Moreover, algorithms are explicitly stated in a way independent of the animation, which simplifies the production of multiple animation for users to look at an algorithm from different points of view. Furthermore, the C++ encapsulation enables the reuse of common data classes. Hence, the time for animating algorithms on the data classes can be significantly reduced.

4
Conclusion

4.1 Future Directions

Four directions in future research are identified; namely, Algorithm Debugging, General Algorithm Language, Parallel Algorithm and Algorithm User Interface Generator. The first two sticks to the current prototype; whereas the last two should deserve an entire new model.

4.1.1 Algorithm Debugging

At present, an algorithm being animated is assumed to be error-free. However, the algorithm should be refined from time to time in practice. Not until the completion of the entire animation, current prototype provides no means to obtain feedback from the algorithm so as to justify the correctness, which in other words obstructs the development of the algorithm. Hence, we are going to build an algorithm debugger, which helps programmers examine the data in each C++ class. The C++ classes can then be verified one after another; like performing a bottom-up testing. Before animated, the algorithm can be sure of being trouble-shot. In a sense, this direction approaches the notion of Code Visualization, which aims at facilitating software development.

4.1.2 General Algorithm Language

Current prototype is bound to the C++ language. However, algorithms are frequently written in language specific to the problem; for example, using Prolog for problems in AI. Hence, linking up current graphical display with algorithms in different languages is to be investigated. An intuitive idea is to use a general script as Stills and Movie[1] does. `Printf` statements are annotated into a C program in order to generate a text file of script; describing the program execution. Undoubtedly, the script file can be produced regardless of the language in use. However, the algorithm is shown in action usually after the generation of the script file has been completed, which does not allow users to interact with the algorithm.

[1] mentioned in section 3.2.3.

4.1.3 Parallel Algorithm

Our model binds graphical objects to a single execution of the program, which can be extended to enable executing programs concurrently, thus, animating parallel algorithms. A typical example is parallel sorting, in which each object compares itself to its neighbors and moves itself if necessary. Specifying the motion of one object can in turn define the behavior of the entire animation. In addition, current prototype is running on the Microsoft Windows 3.1, which adopts a co-operative multitasking scheme. Being restricted by the scheme, the prototype can never run programs in parallel, thus, disallowing parallel algorithm animation. Nevertheless, the latest Windows NT (New Technology) will soon be shipped, which should embrace a true preemptive multitasking.

4.1.4 Algorithm User Interface Generator

Enhancing the notion of interactive animation has given rise to the building of a user interface upon algorithms. Graphical objects on display can include some Windows Controls as well; including button and menu. They should be similarly bound to some data declared in a class. This direction goes toward research in UIMS (User Interface Management Systems).

4.2 Summary

The main contributions of this thesis are the two models: the Lecture Presentation and the Animation Production Models. The models have been realized through the design and implementation of the Pearl system. Using the system, we have also animated a wide range of data structures and algorithms; including, Sorting, Stack, Tic-tac-toe, Knapsack Problem, Turing Machine, N Queens Problem and Maximal Matching.

The Lecture Presentation Model is well-suited for classroom presentation. In the model, delivering a lecture is decomposed into four procedures; namely, Materials Organization, Slide Preparation, Animation Production and Actual Presentation. Various novel features have been proposed for each procedure respectively; including, history, story board, highlight and zooming. Examples of using the features to assist in teaching have also been given. More importantly, helping actual presentation is generally unnoticed elsewhere.

In the Animation Production Model, an animation is made in three steps. Firstly, an algorithm is implemented as a C++ program. Secondly, direct manipulation is used to create the graphical objects in display. Thirdly, the variables in the program and the attributes of the graphical objects are bound together to form an animation. Additional options can also be specified so as to adjust the controls of the animation. Throughout the previous sections, animating the sorting algorithms has demonstrated how to produce an animation with the model.

The Animation Production Model has three main characteristics. Firstly, users are allowed to have interactions with the underlying algorithms. For example, users can interactively arrange the items to be sorted at will. Not accepting user interactions is also one of the shortcomings of existing systems. Secondly, being independent of the display, algorithms can be easily represented in different ways. For example, the items can be denoted as a list of numbers as well as several rectangles. Thirdly, the implemented programs are highly reusable since object oriented C++ classes are used.

Furthermore, future directions in research have been made known; including Algorithm Debugging, General Algorithm Language, Parallel Algorithm and Algorithm User Interface Generator.

4.3 Epilogue

This thesis compiles technical issues on using computer to assist in lecture presentation and to produce animation for showing algorithms in action. In spite of its success, there are actually non-technical problems involved, which have been previously ignored.

Lecturing technology is going forward faster than its popularity. Although these teaching aids have been available for years, few educators are now aware of their advantages. Moreover, some teachers, especially non-technical people, are having phobia in using computer. Their inertia in retaining current teaching methodologies gets in the way of applying computer in teaching.

Although preliminary trials of Algorithm Animation are "promising", we are short of formal evidence to prove its usefulness. Experiments should be conducted to contrast the learning rates of two groups of students; one with the use of the algorithm animation while the other without. Actually, we are interested in delivering the courses on data structures and algorithms with the Pearl system provided that we have a wide assortment of animated algorithms.

Furthermore, making an animation is indeed an art, which cannot be directly improved through technology. Producing an attractive and impressive animation requires knowledge in aesthetic. Teaching effectively with animation further demands expertise in education. Perhaps, much more work crossing various disciplines has to be continued to make teaching with animated algorithms happen.

Appendix A
PostScript Optimization

A.1 Introduction

Most software generates PostScript programs [42] by rules so that the PostScript programs are frequently not concisely written. For example, lengthy function definitions are always included as header no matter whether the functions will be used in the program or not. Clumsiness of the programs not only takes up much space but also slows down the execution time; especially, when the PostScript programs are to be printed in a large quantity.

This appendix documents our study of designing a heuristic to produce efficient PostScript programs through using an optimizer to simplify the style in writing of the programs, which is analogous to performing code optimization during conventional program compilation. The current approach is to schedule the objects to be printed, and in turn get together objects with adjacent attributes. In this way, number of necessary changes in attributes is reduced to a minimum.

In addition, a tentative optimizer has been designed and developed to arrange text objects of similar fonts. Using the optimizer, we have also conducted an experiment to measure the performance of the programs against the optimized one.

Next section fully describes the current approach; followed by the implementation details of the optimizer. The succeeding section gives an example showing the operation of the optimizer. The findings of the experiment are then reported. This appendix is concluded in the last section with a discussion of problems encountered.

A.2 The Current Approach

As mentioned before, the current approach is to sort out the objects to be printed such that the total number of changes in attributes between the objects is minimized. Preliminary experiments have been carried out, in which text objects of identical font are gathered to be printed at the same time.

Actually, modifying the font in use of a printer is indeed time-consuming since a new set of fonts has to be loaded onto the memory of the printer; moreover, each coordinate of all characters has to be computed for the particular point size.

Intuitively, text objects are put in the order of occurrence in the PostScript program as most word processors do. For example, "14 th February" is separated and arranged into "14 ", "th" and " February" in turn. As such, the PostScript interpreter has first to be set to font Times Roman with point size 12 for the foremost token "14 ". The token "th" in between then causes the interpreter to be modified to point size 10. Finally, the interpreter is required to be re-set to the original point size 12 so as to print the last token " February". A total of three changes in point size is then involved. However, obviously, putting the token "th" in the first place followed by the tokens "14 " and " February" together will cut down the number of changes to two.

A.3 Implementation

To justify the notional approach, a PostScript optimizer has been designed and implemented for a restricted domain of PostScript commands. The optimizer has been written using C in the Ultrix environment. The following enters into the details of the data structure and algorithm used in the optimizer.

- *FontTable* maintains all the various fonts which have been made use in the PostScript program so far. Associated with each font is a list of copies of text to be printed in that particular font.
- *FontDefault* indicates the current font by default.

Using FontTable and FontDefault, the algorithm to sort out the text objects is definitely stated in figure A.1.

```
Set FontTable empty;
Set FontDefault NULL;
while not EOF do
    begin
        Read a line;
        if the line sets to a font then
            begin
                Append the font to FontTable;
                Set FontDefault to the font
            end
        else if the line prints a text then
            Append the text to the entry pointed to by FontDefault
        else
            begin
                for each entry of the FontTable do
                    begin
                        Output the Set statement of the font of the entry;
                        Output the Print statement(s) of the text of the entry
                    end;
                Output the line;
            end
    end
```

Figure A.1 Algorithm of the current approach

A.4 An Example

To concretely illustrate the algorithm, an example is made as follows. The PostScript program to print "θ2-5θ1+6" is similar to codes in figure A.2.

```
...
/Symbol findfont 12 scalefont setfont          (1)
315 267 moveto (q) show                        (2)
/Times-Roman findfont 10 scalefont setfont     (3)
341 280 moveto (2) show                        (4)
/Times-Roman findfont 12 scalefont setfont     (5)
362 273 moveto (-5) show                       (6)
/Symbol findfont 12 scalefont setfont          (7)
404 267 moveto (q) show                        (8)
/Times-Roman findfont 10 scalefont setfont     (9)
430 280 moveto (1) show                        (10)
/Times-Roman findfont 12 scalefont setfont     (11)
451 273 moveto (+6) show                       (12)
showpage                                       (13)
...
```

Figure A.2 Original Program

After line 12 has been read, the FontTable has grown into the stage depicted in figure A.3.

FontTable

Symbol	12	→	θ	315, 267	→	θ	404, 267
Times Roman	10	→	2	341, 280	→	1	430, 280
Times Roman*	12	→	-5	362, 273	→	+6	451, 273

*FontDefault

Figure A.3 FontTable and FontDefault

Line 13 triggers off the re-generation of the PostScript program (See figure A.4). The length of the program has been shortened to 10 lines; whereas the number of setfont operation has been significantly reduced from 6 to 3.

```
...
/Symbol findfont 12 scalefont setfont          (1)
315 267 moveto (q) show                        (2)
404 267 moveto (q) show                        (3)
/Times-Roman findfont 10 scalefont setfont     (4)
341 280 moveto (2) show                        (5)
430 280 moveto (1) show                        (6)
/Times-Roman findfont 12 scalefont setfont     (7)
362 273 moveto (-5) show                       (8)
451 273 moveto (+6) show                       (9)
showpage                                       (10)
...
```

Figure A.4 Optimized Program

A.5 Observations

A number of arbitrary PostScript programs generated by the printer driver in Microsoft Windows are chosen to be examined. The time to print the programs and the size of the programs are measured before and after optimization (See table A.1). The findings show on average a reduction of 20% in printing time and 17% in program size.

Trial	Before		After		Speed Up	
	Time	Size	Time	Size	Time	Size
1	25s	15267 Bytes	20s	13788 Bytes	20.0%	9.7%
2	26	17900	20	14252	23.1	20.3
3	26	17988	21	14683	19.2	18.4
4	25	17687	21	14379	16.0	18.7

Table A.1 Results

The time to optimize the PostScript programs is negligible since all the programs take up less than one second to be optimized. In addition, the experiment has been performed using the DEC personal station DEC5025 connected to the Apple Laser Writer IIg.

A.6 Conclusion

A heuristic bringing objects of adjacent attributes together has been designed to enhance the printing of PostScript programs. In practice, an experimental optimizer has been implemented to improve on printing text objects with various fonts. Using the optimizer upon several PostScript programs has shown a decrease in both printing time and program size.

However, tuning general PostScript programs raises much difficulty. For example, one can never avoid program undecidibility, which means that the state of a program is non-determinable except really putting the program in execution. Being likely to alleviate the problem, data flow analysis unfortunately demands enormous effort.

Appendix B
Thesis Publications

1. H.C. Lam, C.S. Chang, K.S. Leung and T.C. Chen, "A Computer Graphics Aided Lecture Presentation System", to appear in proceedings of 1993 International Conference on Computers in Education, Taiwan, Dec. 1993.
2. H.C. Lam, K.S. Leung and C.S. Chang, "Exploring Animated Algorithms with Direct Manipulation", in preparation.

References

1. Microsoft Windows 3.1, Microsoft Corporation, Washington, 1992.
2. Turbo C++ 3.0 for Windows, Borland International Inc., California, 1991.
3. Microsoft Windows Software Development Kit, Microsoft Corporation, Washington, 1990.
4. Peter Norton and Paul Yao, Windows 3.0 Power Programming Techniques, Bantam Computer Books, New York, 1990.
5. Carol S. Holzberg, "LCD Panels", Electronic Learning, Mar. 1991, pp.46-49.
6. Authorware Professional, Authorware, Inc., Minnesota, 1989.
7. Carol B. Macknight, Santosh Balagopalan, "Authoring Systems: Some Instructional Implications", Journal of Educational Technology Systems, Vol. 17, No. 2, 1988-89, pp.123-134.
8. "Presenting with the PC", Personal Computer World, Nov. 1990, pp.228-236.
9. "The Macintosh presents...", Personal Computer World, Nov. 1990, pp.250-256.
10. Luisa Simone, "2-D Animation Software: The Motion Is the Message", PC Magazine, Aug. 1992, pp.435-467.
11. Anton S. Y. Lam and C. S. Chang, "Prototype of a Courseware Production and Presentation System, Educational Technology", Vol. 32, No. 4, Apr. 1992, pp.20-28.
12. Lam Shing-yung, "Visual Interaction Techniques for Courseware Production and Presentation", M.Phil. Thesis, The Chinese University of Hong Kong, Shatin, Hong Kong, 1991.
13. Cristina Stuart, Effective Speaking, Gower, Hants, England, 1989.
14. Ivan Tomek, Saleem Khan, Tomasz Muldner, Mostata Wassar, George Novak and Piotv Proszynski, "Hypermedia -- Introduction and Survey", Journal of Microcomputer Applications, Vol. 14, No. 12, Apr. 1991, pp.63-103.
15. HyperCard Reference, Claris Corporation, California, 1990.
16. Brad A. Myers, "Taxonomies of Visual Programming and Program Visualization", Journal of Visual Languages and Computing, Vol. 1, No. 1, 1990, pp.97-123.
17. Marc H. Brown, Algorithm Animation, The MIT Press, Massachusetts, 1988.
18. Marc H. Brown and Robert Sedgewick, "A System for Algorithm Animation", Computer Graphics, Vol. 18, No. 3, Jul. 1984, pp.177-186.
19. Marc H. Brown and Robert Sedgewick, "Techniques for Algorithm Animation", IEEE Software, Vol. 2, No. 1, Jan. 1985, pp.28-39.

20. Marc H. Brown, "Exploring Algorithms Using Balsa-II", Computer, Vol. 21, No. 5, May 1988, pp.14-36.
21. Marc H. Brown, "Perspectives on Algorithm Animation", Proceedings of CHI'88 conference on Human Factors in Computing Systems, Washington, DC, May 1988, pp.33-38.
22. Marc H. Brown and John Hershberger, "Color and Sound in Algorithm Animation", Computer, Vol. 25, No. 12, Dec. 1992, pp.52-63.
23. Marc H. Brown and John Hershberger, "Zeus: A System for Algorithm Animation and Multi-View Editing", Proceedings of IEEE Workshop on Visual Languages, Kobe, Japan, Sept. 1991, pp. 4-9.
24. Ralph L. London and Robert A. Duisberg, "Animating Programs Using Smalltalk", Computer, Vol. 18, No. 8, Aug. 1985, pp.61-71.
25. Robert Adámy Duisberg, "Animation Using Temporal Constraints: An Overview of the Animus System", Human-Computer Interaction, Vol. 3, No. 3, 1987-88, pp.275-307.
26. Robert A. Duisberg, "Visual Programming of Program Visualizations", Proceedings of IEEE Workshop on Visual Languages, Linköping, Sweden, Aug. 1987, pp.55-66.
27. Jon L. Bentley and Brian W. Kernighan, "A System for Algorithm Animation: Tutorial and User Manual", Computing Science Technical Report No. 132, AT&T Bell Laboratories, Murray Hill, New Jersey, Jan. 1987.
28. Esa Helttula, Aulikki Hyrskykari and Kari-Jouko Räihä, "Graphical Specification of Algorithm Animations with ALADDIN", Proceedings of 22nd Hawaï International Conference on System Sciences, Kailua-Kona, Hawaï, Jan. 1989, IEEE Computer Society Press, pp.892-901.
29. John T. Stasko, "TANGO: A Framework and System for Algorithm Animation", Ph.D. thesis, Technical Report No. CS-89-30, Department of Computer Science, Brown University, Providence, RI, May 1989.
30. John T. Stasko, "Simplifying Algorithm Animation with TANGO", Proceedings of IEEE Workshop on Visual Languages, Skokie, IL, 1990, pp.1-6.
31. John T. Stasko, "TANGO: A Framework and System for Algorithm Animation", Computer, Vol. 23, No. 9, Sept. 1990, pp.27-39.
32. John T. Stasko, "Using Direct Manipulation to Build Algorithm Animations by Demonstration", Proceedings of ACM CHI Conference on Human Factors in Computing Systems, pp.307-314.

33. Kenneth C. Cox and Gruia-Catalin Roman, "Abstraction in Algorithm Animation", Proceedings of IEEE Workshop on Visual Languages, Seattle, WA, Sept. 1992, pp.18-24. Also available as Technical Report No. WUCS-92-14, Department of Computer Science, Washington University, Campus Box 1045, One Brookings Drive, Saint Louis, MO.
34. Kenneth C. Cox and Gruia-Catalin Roman, "Experiences with the Pavane Program Visualization Environment", Technical Report No. WUCS-92-40, Department of Computer Science, Washington University, Campus Box 1045, One Brookings Drive, Saint Louis, MO.
35. Robert R. Henry, Kenneth M. Whaley and Bruce Forstall, "The University of Washington Illustrating Compiler", Technical Report No. 90-07-01, Department of Computer Science and Engineering, FR-35, University of Washington, Seattle, WA.
36. Heinrich Müller, Jörg Winckler, Stefan Grzybek, Matthias Otte, Bertram Stoll, Frederic Equoy and Nicolas Higelin, "The Program Animation System PASTIS", The Journal of Visualization and Computer Animation, Vol. 2, 1991, pp.26-33.
37. Ricardo A. Baeza-Yates, Luis Jara and Gastón Quezada, "VCC: Automatic Animation of C Programs", Proceedings of COMPUGRAPHICS'92, Lisboa, Portugal, Dec. 1992.
38. Michael T. Heath, Jennifer A. Etheridge, "Visualizing the Performance of Parallel Programs", IEEE Software, Vol. 8, No. 5, Sept. 1991, pp.29-39.
39. Ulla Solin, "Parallel Algorithm Animation", Technical Report Series A, No. 50, Institutionen för Informationsbehandling, Åbo Akademi, Fänriksgatan 3, SF-20500 Åbo, Finland, 1986.
40. Konstantinos Konstantinides, "Algorithm Visualization using Tree Graphs", the Visual Computer, Springer-Verlag, No. 7, 1991, pp.220-228.
41. Blaine A. Price, "A Principled Taxonomy of Software Visualization", to appear in Journal of Visual Languages and Computing, Vol. 4, No. 3, Sept. 1993.
42. PostScript Language Reference Manual, Tutorial and Cookbook and Program Design, Adobe Systems Incorporated, Addison Wesley, Massachusetts, 1985.