An overview of visualization: its use and design
Report of the Working Group on Visualization

Joe Bergin
Pace University, USA
Ken Brodlie
University of Leeds, UK
Michael Goldweber
Beloit College, USA
Ricardo Jiménez-Peris
Universitat Politecnica de Madrid, Spain
Sami Khuri
San Jose State University, USA
Marta Patiño-Martínez
Universitat Politecnica de Madrid, Spain
Myles McNally
Alma College, USA
Tom Naps (chair)
Lawrence University, USA
Susan Rodger
Duke University, USA
Judith Wilson
Temple University, USA

Abstract

This paper presents an overview of visualization in Computer Science instruction. It is broken down in the following fashion. First, we present the motivation for using visualization and visual techniques in instruction. This is followed by a discussion of when the use of visualization is most appropriate.

We then consider a broad spectrum of uses of visualization in Computer Science instruction. This spectrum is organized from passive to active in terms of a student's involvement with the visualization tools. Types of visualizations are then categorized.

The remainder of the paper focuses more on design issues for instructional visualization tools. These design issues are first presented from the perspective of the instructor who is constructing the visualization tool for students and then from the perspective of the programmer who is creating visualization software. We close the paper with some suggestions on organizing and maintaining a Web-based repository of visualization tools for Computer Science instruction.

1 Motivation

Visualization tools can help Computer Science instructors in a variety of ways, ranging from merely attracting students' attention to constructing in-depth exercises that require substantial student involvement. In this section we present some motivating factors for instructional use of visualization tools.

- Clarification of complex concepts through the use of pictures: the teaching of complex concepts through language alone often requires a formalism that intimidates students. Although it is certainly important for our students to eventually acquire a facility for understanding such formal definitions, we can often aid them with appropriate visualizations of the concepts.

- Alternative presentation modality: research by Tom West [17, 18] has indicated that some students think best in visual terms. Such students would be disadvantaged if not presented with visualizations of material. It is reasonable to maintain that all students learn certain concepts better by thinking visually about them, that is, by building mental images of the concepts. The visualization tools can facilitate the building of such mental models.

- A hook by which we can grab students' attention: increasingly, instructors have discerned the need to interrupt their students' level of (in)attention in order to focus students' thinking on a given problem. Visualization tools, in part by virtue of their similarity to popular forms of entertainment are often successful at this grabbing and maintaining of students' interest.

- Visualization tools allow instructors to cover more material in less time: a frequent use of visualization tools, when combined with large-screen projection devices, is to automate the demonstrations and examples we present in lectures. There has been little, if any, formal research that would prove that students learn concepts better when they are presented in this fashion in lectures. But, let us assume that such concepts are at least learned as well as they would be when instructors use manual techniques. This seems a reasonable assumption since the automatic production and display of such demonstrations at least removes the errors that inevitably occur when instructors manually draw structures as B-trees on the board. Under this assumption, Eric Roberts provides an interesting rationale for using visualization tools in [1]. His argument is that students learn concepts as least as well as they did without the tools and that using the tools enables us to cover material faster in our lectures. Therefore, we cover more material with at least the same level of understanding as before, so there is a net gain.

- Good visualizations can increase students' understanding: anecdotal evidence indicates that good interactive visualizations can increase students' understanding. In particular, an interactive visualization allows students to receive feedback on correctness, to step slowly through an algorithm at their own pace, to view changes pictorially in an algorithm, and to view the outcomes when different data sets are input to an algorithm.

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

Integrating Tech. into C.S.E. 6/96 Barcelona, Spain
© 1996 ACM 0-89791-844-4/96/0009...$3.50

- Visualization encourages modes of learning we want to see in students: 'Because students resist theory, preferring practical, hands-on computer problem-solving', [8] the use of highly interactive simulation software packages that support visualization, stimulates and clarifies most of the concepts for them. Such packages can help students better comprehend the theoretical concepts underlying applications. Given a problem instance, students can experiment with different inputs and investigate various aspects of problems, concepts and algorithms at their own pace.

- Handling students with different backgrounds: educators are often faced with the challenging task of motivating the less well-prepared students while challenging the more advanced ones. One way of motivating all students is to include optional problems in assignments so as to challenge the more advanced students. Visualization tools that include theoretical explanations and challenging problems can sustain student interest and allow self-paced student exploration.

- Visual debugging: debugging is not a pleasant endeavor. Visual debugging can simplify the debugging process and remove some of the frustrations, thus rendering the whole process more tractable. For example, one can simultaneously see the individual lines of code executing, monitor the state of variables, and visually ascertain whether or not a linked list is connected.

- Visualization allows us to capture effective manual classroom techniques: the illustration of many procedures, especially those which involve two or higher dimensional structures, profit from visual animations. It is time-consuming and difficult to draw such structures on the board manually. The automation of visual images saves time, and it also reduces errors and increases the clarity of classroom presentations.

- Broadening visualization to perceptualization: it is important to realize that visualization is by no means limited to 'visual images'. The latter can be enhanced by including other perceptual components, such as audio.

2 When to use visualization

Some form of visualization, either the simplest form of a picture or the more complex form of an animation, should be used in the teaching of Computer Science concepts and algorithms. Only for very simple examples is effort wasted on creating a visualization. If computer tools are used to generate the visualization, these tools should be of the highest quality and must have a high degree of interactivity.

The choice of the type of visualization used, a picture or an animation, depends on the topic and the cost effectiveness. A visualization in the form of a picture is used when the amount of data is small, the data structure is very simple, or when the relationship of objects is important, but movement is not needed. When a small amount of data is shown, the steps of an algorithm can be traced with snapshots and it may not be worth the time and investment of generating an animation. For example, the illustration of insertion sort with 4 elements is enough to illustrate the main idea in the algorithm, and can be shown on a one page trace. When the data structure is very simple, a picture of the data structure is much clearer than a textual description of it. For example, a picture of a linked list is much clearer to understand than a textual description of each element, its position in the list, and a description of how elements are connected together. Furthermore, the description of an insertion into a linked list can be easily illustrated with two pictures, the initial list and the final list with old links crossed out. If there is no movement, then a picture can be used to illustrate the relationships between objects. For example, in a mathematical proof, a pictorial model of the steps in the proof can be helpful in remembering the proof.

A visualization in the form of an animation generated on the computer is used when the quantity of data is large, the data structure is complex, or when movement is needed to show how the relationships between objects change over time. When a large quantity of data is used, an animation can show how this data is processed. For example, the illustration of shellsort must use a large amount of data in order to show several partition breakdowns. As another example, illustrating the binary search algorithm and its analysis is more effective using a large amount of data to show how fast one can find a particular element. When a data structure is complex and/or changes of the data structure are to be discussed, an animation can show these complexities and movement much more easily than one could show them on a blackboard. For example, to illustrate the insertion of a new element into a red-black tree, one must show the colors of all the nodes, the recoloring of specific nodes and then possibly one or two rotations, which could involve large subtrees moving. Not only is this difficult to animate on a blackboard with erasures and redrawings, but students have difficulty in trying to take notes on a picture that is changing.

The type of picture or animation created, either hand drawn or computer generated, depends on the cost benefit of developing the picture. Non-technical animations can be just as effective, may involve less time in their creation, and can involve the class with more interaction. We give three examples. First, a phone book can be used to illustrate the quickness of binary search in looking up the phone number for a specific person. Second, students sitting at a table can be used to illustrate the dining philospher's problem. Finally, a linked list can be bought at a toy store: large snap color beads that are composed of 'nodes', conveniently containing a hole on one end and a snub or 'pointer' on the other end for snapping them together. The illustration of building a linked list can be done with student participation. One student can be the 'heap' and hold all the nodes. Whenever a new node is created, the heap is called and throws the builder a new node. This emphasizes that memory must be obtained dynamically when creating a linked list.

3 Using visualization in teaching

Visualization tools can be used in a variety of teaching situations: classroom demonstrations, open or closed laboratories, and traditional assignments. The student can be a passive viewer of information or an active participant in the visualization process, and involved in instructor-led, individual, or collaborative work.

In the classroom, visualizations are often employed in the demonstration of a topic. Given the power of a visualization to illustrate abstractions, visualization tools may be appropriately used before, after and during the discussion of a topic -- before, because of the ability of visualizations to abstract from implementation details, and after, to reinforce lecture material such as algorithm details. It is thus important that a visualization tool allows access to the topic under investigation from different viewpoints and at multiple levels of abstraction.

Active engagement is an important goal, not only in the laboratory, but in the classroom as well. Visualization tools need not be used merely as 'view only' tools in the classroom. For instance, if the topic is balancing criteria for binary trees, and the tool is one in which values can be merely added or deleted, an instructor could begin by demonstrating the addition and deletion of values and resulting balancing actions. But as the presentation proceeds, students can be engaged in a more active fashion by questions such as 'What would be the effect of adding this value?' or 'Name a value whose addition will lead to a double rotation.' Later, collaborative work could be assigned to solve such problems as 'Generate a minimal binary search tree which is red-black but not AVL.'

Evidence indicates that active engagement on the part of students leads to higher motivation and better integration and retention of content. Some visualization tools lend themselves to active student engagement more readily than others. In table 1 we have classified visualization tools along a continuum from maximally passive to maximally active student involvement. At one end of the scale, static pictures and canned demonstrations are 'view only' and require the instructor's intervention to actively engage the student. At the other end, student created or modified visualizations require active involvement. Examples of appropriate classroom and laboratory use for each tool are indicated in the rightmost column.

4 Types of visualization tools

Visualization tools can be characterized in several ways. this characterization is based on the taxonomy by R. Roman & K. Cox [13]. The criteria used from it are:

- Scope: what aspects of the program are visualized? Visualization tools can show code, events, and/or data.

- Abstraction level.: this is the level of the concepts displayed. It can be a direct representation where there is a picture associated with some aspect of the program. In that way the original information can be reconstructed from the picture. Structural representation is more abstract than direct representation. Aspects not important to the viewer are supressed, while those of interest are directly represented. Synthesized representation is the most abstract level of visualization; here the information visualized is derived from program data.

- Specification method: this is the mechanism used by the animator to construct the animation. Some systems have a fixed mapping from the program to the visualization while others allow the definition of this mapping by annotating the program with procedures that draw and modify images.

- User Interaction: the viewer of the visualization can have low interaction with the visualization (passive) or frequently interact by selecting data, defining the speed of the visusalization, and so forth.

These four points can be summarized as what information is visualized (scope and abstraction level), and how the visualization is generated (specification method), the latter also determining the potential users of the tools. If there are predefined visualizations, the tool is easy to use and the useris required to do nothing to visualize a program. These kinds of tools are adequate for students new to visualization. On the other hand, if the specification method of the tool is annotation, the tool will be more difficult to use and thus not adequate for beginners. The specification method also influences what type of use the tool can have. Predefined visualizations are specific tools, they can only show the predefined aspects, while annotation makes tools more flexible thus allowing the user to customize the visualization.

Degree of student engagement	Visualization types and tools	Examples of instructional use
Most Passive	Static pictures	Lectures; ; Textbook & Workbook figures
	Canned (dynamic) Demos	Lecture supplements; Introduction to visual systems (ex. PAIL [1]) Preparation for lab exercisesExercises
	Demos with display adjustments	Lecture supplements; Closed lab exercises; Open lab review
	Simulations	Lecture supplements; Closed lab exercises; Open lab review
	Animations with interactive data input	Lecture supplements; Closed lab exercises; Open lab review
	Games(inclued interactive decision points)	Lab (open or closed) exerecises
	Learning environments(highly interactive systems that support exploration: notations, interactive value adjustments, ...	Lab exercises: exploration and experimentation primarily in open labs (e.t. PAIL, FLAIR [2], Smithtown [3]
	Student modifiable visualizations Student generatable visualizations	Open lab exercises; Term projects;
Most Active	Visual output for programs	Programming assignments

Table 1

1 PAIL = Portable AI Laboratory [1]
2 FLAIR = Flexible Learning with an AI Repository [9]
3 Smithtown, is used to teach microeconomics to middle school students [14]

Tools can also be clasified from the point of view of their flexibility, that is, *how* they can be used. Specific tools visualize an algorithm or a set of them in a predefined way. Visual debuggers can depict any algorithm in a predefined way. On the other hand, visualization generators are the most flexible since they can visualize any algorithm in many different ways.

Existing visualization tools can be clasified into three categories: program visualization, algorithm animation, and data visualization

tools. Each of them has a scope, a level of abstraction and a specification method.

Program visualization (Visual debuggers)

This type of visualization focuses on the graphical representation of an executing program and its data.

- Scope: data, code and events of interest are visualized.

- Abstraction: direct representation of the code and data (low level of abstraction).

- Specification method: data and code have a predefined visualization. Each aspect of the program to be visualized is predefined.

- User interaction: the role of the user can be from passive to active user. Typical interactions are: breakpoints, changes of input data, and so forth. The interactions are predefined.

Algorithm animation

This type of visualization shows operations fundamental to an algoithm, as opposed to just code and data.

- Scope: these are general purpose tools, the user decides what to visualize.

- Abstraction: since the user of the tool decides what to visualize and how, the level of abstraction to be shown can be chosen. Most of these tools have a structural or synthesized representation.

- Specification method: user defines the relationship between the algorithm and the objects displayed by means of annotations. That is, the user chooses the images for the aspects of the program to be visualized and when to visualize them.

- User interaction: this is also defined by the programmer, so these tools range from passive user to a high level of interaction.

Data visualization

This kind of visualization shows operations inherent to an abstract data type, as opposed to program code.

- Scope: only data or other things that can be represented as data. For instance a tree of recursive calls or a stack of pending calls, are visualized.

- Abstraction: these tools have a high level of abstraction, structural or synthesized representation, tending to group or organize data in some way, and do not just showing a collection of values.

- Specification method: the program is annotated in order to produce a data trace.

- User interaction: the user of these tools is a passive user having a low level of interaction.

5 What makes a visualization effective?

An effective visualization combines the skills of a human designer with the capabilities of the software and hardware on which the system will run. In this section we present a set of principles to keep in mind when designing a visualization tool. These principles bridge the gap from our earlier discussion of using instructional visualizations to the ensuing discussion of software design patterns underlying a visualization tool.

- Instruction is paramount: remember that the primary goal in constructing a pedagogical visualization is to instruct, not to entertain. Although we may want to use graphic 'tricks' to grab students' attention, we only keep their attention if the graphics truly lead to better understanding of the material. According to Miller in [11], "It is much easier to draw a visually attractive picture than it is to draw a useful one. You know you have goofed when you hear: 'that really looks neat, what does it mean?' "

- Provide the user with a standard GUI to manipulate within the visualization tool: anecdotal evidence suggests that one reason instructors and students avoid using visualization tools is the learning curve associated with becoming adept at using them. This learning curve can be greatly reduced by presenting the visualization in the context of a GUI with which the student is already comfortable, for example, the standard Windows or Macintosh 'look-and-feel'. Providing online help is also essential.

- Provide the user with different views of the data, program, or algorithm: the structures underlying most sciences are embedded in a physical model. This is not the case with Computer Science in which the underlying structures are purely conceptual. This often results in an inherent complexity that can only be realized through a visualization that can assume a variety of forms. For instance, in depicting the actions of the quicksort algorithm, one could begin by viewing the actual values of the items in the array being sorted. However, this view requires the user to spend considerable time mentally arranging the data items according to relative magnitude. More intuitive views of the data values are the so-called 'sticks' view and the 'dots' view. In the former, the magnitude of each item is reflected by the length of a bar, with a visual effect similar to that of a histogram. In the latter each value is plotted as a dot on a set of axes. The vertical axis represents the relative magnitude of the number while moving to the right along the horizontal axis represents moving into higher array indices. The net visual effect is that of a scattergram. However, quicksort is also a model for recursion. In this context, the user may want to view the stack of activation records ssociated with each recursive call or the tree of recursive calls. Hence we have described five separate views of the quicksort algorithm -- the actual data values in the array structure, the sticks view, the dots view, the run-time stack, and the recursive call tree. An effective visualization of quicksort should allow the user to see each of these views simultaneously each time an interesting event is encountered in the algorithm's execution.

- Provide the user with simultaneously identical views of different algorithms manipulating the same data: this principle is the 'dual' to the preceding one. When comparing different algorithms, it is imperative that the student be able to watch how each would operate using the same data set. Such a comparison should be carried out by presenting the same view of the two algorithms in different windows. For instance, when comparing the heap and quicksort algorithms, we may wish to present a 'sticks' view of each algorithm. By watching the simultaneous animations of the two algorithms, the student will become aware not only of how each algorithm compares and exchanges items in the array but also of differences in efficiencies of the two algorithms.

• Strive to draw the user's attention to the critical area of the visualization: there will typically be an area of central activity in an algorithm. For example, in studying graph search algorithms, this area will be the current node under consideration and those nodes directly adjacent to it. the visualization should focus the user's attention on these nodes. Perhaps the easiest way to do this is to highlight these nodes with different colors. Color used in this fashion is truly informative rather than merely entertaining for the user. Another less obvious technique to focus attention on the area of central activity is the use of a 'fisheye' view. In such a view the central area will appear magnified when compared to other areas of the visualization. One should also consider the use of audio to focus the attention of the user. When combined with audio, the visualization becomes a true perceptualization of the principle being studied.

• Use a text window to make sure the user understands the visualization: in [16], Stasko, Badre, and Lewis point out that one reason algorithm animations are often ineffective is that the instructor (that is, the creator of the visualization) 'already understands the algorithm. Students just learning about an algorithm do not have a foundation of understanding upon which to construct a mapping from the algorithm to its visualization.' To construct this mapping, the instructor must explain it in words -- hence the need for an appropriately sized text window. Although this window may contain portions of the code being visualized, it is not a code window in the sense of a graphical debugger. The window should de-emphasize code and emphasize clearly written instructional material. Ideally this window should be visible throughout the visualization so that the user may requently refer to it without having to switch contexts. Proponents of visualization often claim that 'one picture is worth 1,000 words.' We feel that, by combining visualization with carefully written prose, we can achieve an even greater gain -- one picture plus 100 words is worth 1,000,000 words.

• A high degree of interactive user control is necessary: users will often get lost even when viewing the best of animations and visualizations. Once lost, they must be able to control the visualization to get back on track. Without the ability to exercise such control, they will merely become passive viewers of what has become a sequence of meaningless pictures. Pan and zoom capabilities are critical in this regard. Pause, speed-up, and slow-down controls are essential for viewing animations. An even more important facility that is not presently found in most animation systems is the ability to go back in time. Such a facility might take the form of a rewind control which would allow the user to back up the animation to a previous point in time and then view again a particularly complex portion of the animation. Another alternative that allows the user to answer the question 'How did I reach this point?' is to use three-dimensional graphics in which the third dimension represents time. Thus, one sees previous views drifting back in space. The effective use of three-dimensional graphics in algorithm animation systems has been explored by Brown in [4] but requires much further study.

• Substantial screen real estate will be needed for the most effective visualizations: the ability to present different views of the same algorithm, simultaneous identical views of different algorithms, and textual material explaining the visualization will require a large display area. This means that the best visualizations will often require high-end display devices. Designers of visualizations will be faced with the tradeoff of designing a visualization that can be viewed by many on a wide variety of equipment (for example, over the World Wide Web) at the expense of sacrificing visual information that has more substantial display requirements.

6 Design of visualization software systems

6.1 Introduction

Reference models are useful in identifying the logical operations and interactions in complex systems, and give a basis for discussion. From the basic understanding given by the model, one can devise appropriate implementation models for different computing environments.

The aim of this section is to describe two paradigms, model-view-controller and dataflow, and to show how visualization may be explained in terms of these paradigms. A motivation for this work is to identify strategies that promote system independence and sharing of software among educators; thus we also look at enabling technologies whose use with these architectural models promotes re-usability. Finally we present an existing algorithm visualization system in terms of our models.

6.2 Model-view-controller (MVC) paradigm

The model-view-controller framework emerged relatively early in Smalltalk experience as a paradigm for the creation of interactive applications. It does not depend on Smalltalk or fundamentally on object oriented programming, though it is well supported by object oriented languages. When using MVC, the application is partitioned into three different kinds of classes. Model classes implement the underlying concepts with which the program is concerned. View classes implement what is seen by the user. Controller classes implement the mechanisms with which the user interacts with the program. Typically an application will have a single model and one or more views with one or more controllers.

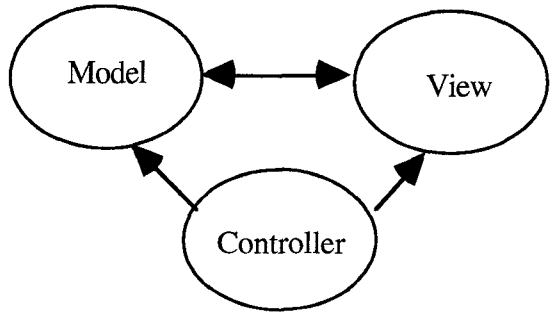

Figure 1 Model View Controller

There is relatively little coupling among the components of the MVC application. In particular, the model has no knowledge of its controllers and very little about its views, except for their existence and support of a standard interface. The decoupling between elements makes it easy to add additional views and controllers to an application, perhaps radically changing its 'look and feel'.

An example of a data-oriented model might be as simple as a queue of pending messages, implemented as a linked list. One view of the queue might be array like, showing the messages in the order in which they occur in the queue.

Another might include cells and links, showing the mplementation details. One controller might be textual, providing a text box into which the user might type commands (insert, remove, ...). Another controller might involve direct manipulation with the mouse, dragging messages into or out of the queue. The textual controller will accept commands from the user and after parsing them and verifying their validity, will pass one or more messages to the model to effect the desired changes. The model will then send 'update' messages to its views so that the displays may be changed to reflect the changes in the model. Upon receiving the update message, the view may query the model for the details of the change. The direct manipulation controller, however, will work directly on one of the views. When the user drags the front element of the queue to another part of the screen, that view may be immediately changed to reflect the user's desires. The view will then send a delete message to the model so that the model is consistent with this view. The model, having been changed, will then alert its views so that all may be made consistent.

The model-view-controller may be used effectively to implement visualizations. The views represent the visualizations themselves and are the relatively passive part of the user interface. The controllers, on the other hand, represent the more active portions, by which the user manipulates and controls the visualization.

6.3 Dataflow paradigm

6.3.1 Introduction

The dataflow paradigm models a computational task as a network of subprocesses, or modules. Data passes through the network, from a source towards a sink; data output from one module acts as input to another. This paradigm is well suited to visual programming where modules can be interactively fitted into a network, in a 'plug and play' fashion. This is illustrated in Figure 2.

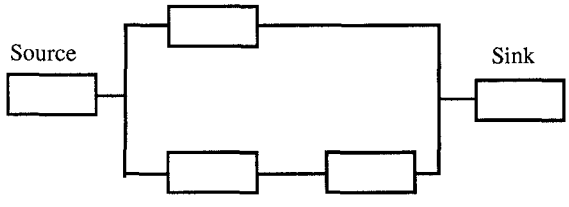

Figure 2 Data Flow Model

6.3.2 Scientific visualization

Haber and McNabb [7] have shown that a scientific visualization may be expressed as a pipeline of processes, as in figure 3, in which the raw data is first input, then passed through a filter process (to select the portion of data of interest), then a map process (to express the data in a geometrical form - that is, lines, areas,...) and finally a render process (to view the geometry from a camera angle, applying lighting and shading). The end result is an image, or a sequence of images forming an animation.

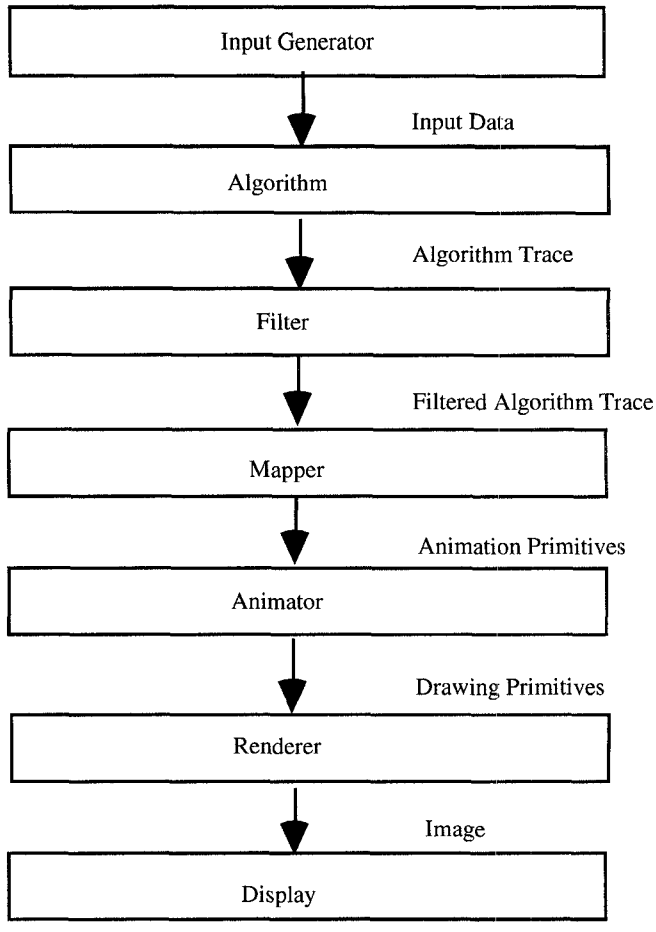

Figure 3 Process Pipeline

A visualization toolkit such as IRIS Explorer [5] will typically provide a set of different filter modules, and a set of different map modules, so that the user can 'plug and play' to create a range of different visualizations. Each module will be controlled by a set of parameters: for example, a map module to extract contour lines from a two-dimensional data set would have a list of contour levels as a parameter.

A key part of the toolkit is the data model: there needs to be standard data types for images, geometry and scientific data. The latter will be two- or three-dimensional arrays of values, positioned on a regular grid, or at scattered locations. By using standard datatypes, new modules can be progressively added to the system; indeed it is possible to allow a user to add new modules, and these will automatically interface if they use the standard data types.

The above model also acts as a basis for distributing the processing work. In most systems, each module in the pipeline can be distributed on a separate processor: the input, filter and map processes for example could be sited on a large central server, and the render process on a workstation. Indeed this is a suitable model for a web version: the geometry may be created on the web server, and converted to VRML or Java bytecodes so that it may be passed to the browser for viewing. VRML is a three-dimensional scene description language gaining rapid acceptance

as a Web standard - it provides machine and operating system independence (see section 6.4).

Scientific visualization is used both for the visualization of measurement data (e.g. medical scanners) and simulation data (e.g. computational fluid dynamics). In the latter case, the input module can be significant - that is, the simulation code itself - and the ability to run modules on different hosts becomes important. Alternatively, the simulation may be run off-line and the data stored; the visualization is then run as a post-processing exercise, as in figure 4.

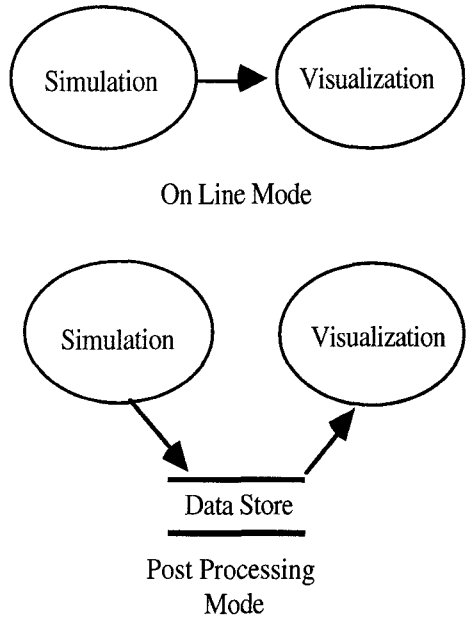

Figure 4 Online vs. Offline Visualization

A key difference is that on-line visualization allows steering of the simulation - that is, the user can control the simulation as it proceeds, making judgements based on the visualization. This is usually not possible in post-processing mode.

6.3.3 Algorithm visualization

The same ideas can be used to derive an architectural reference model for algorithm visualization, where the processes of an algorithm are illustrated. A proposed reference model both for algorithm visualization and for scientific visualization is shown in Figure 3. The source in this figure is data that is input to the algorithm - we term this the input generator process. This passes to the algorithm itself, which outputs an algorithm trace. There are at least two kind of traces:

Event traces report about what has happened in the algorithm. It can be a data change or the occurrence of an operation (e.g. an exchange in a sorting algorithm or a rotation in a AVL tree).

Data traces inform about the new state of the data, either tracing the whole state of the structure each time, or just the changes of the data, which can be called an incremental trace.

Event traces are algorithm specific, so it is difficult if not impossible to predefine them. On the other hand, data traces are easier to standardize as they are only dependent on the type of the data structure, many of which are well-known.

Some examples are:
Binary Search.
* Set array size (size).
* Set value of a component (index, value).
* Set value of the component to search (value).
* Set lower or upper bound (index).
* Set middle index (index).

Sorting.
* Set array size (size)
* Set value of a component (index, value)
* Exchange components (index, index).

Two kinds of data oriented traces can be distinguished: incremental traces and non-incremental traces. In a non-incremental trace it is necessary to specify the variable name, the data structure type, and the value of the data structure. A good method for expressing structured values is to use functional constructors. For instance a binary tree whose content is a node with x as root and y as right child could be expressed as:

node(empty, x, node(empty, y, empty))

In an incremental trace it would be also necessary to specify modify operations such as:

* Insert/remove element in a linked list

* Rotate children of node at a specified position of a tree

The algorithm trace feeds to a filter process which selects a subset of the trace, or modifies the trace, before passing it on to a mapper process. The mapper process assigns an abstract geometric representation to the elements of the algorithm trace. This representation includes both static and dynamic geometric primitives. For example, a dynamic primitive could be to move an object to a new position. We term these animation primitives. It is the map process which is critical to the design of an effective visualization - the abstract representation defines the visual metaphor which is used to communicate the ideas of the algorithm.

These animation primitives flow to an animator process which outputs static drawing primitives; these pass to a render process which generates an image. Similarly, Figure 3 extends to algorithm visualization. In this case, the output from the algorithm will consist of the algorithm trace. As before, this can be created off-line and played back later (post-processing), or can be fed directly to the visualization for on-line viewing.

6.3.4 Utility of the dataflow paradigm

As with scientific visualization, the model acts as a basis for understanding the logical processes involved in algorithm visualization. It can also be useful as an implementation model, and the arguments about distributing processes between client and server, and incorporation in the Web, all carry over immediately.

In particular, the model offers a logical separation between the algorithm and the visualization generator. The separation is

defined by the algorithm trace. This gives a common interface that could even be standardized as an Internet MIME type (the Chemistry community has already done this - there are MIME types to transfer molecular data files across the Internet for example). This protocol would support either on-line, or post-processing styles of working.

A separation between map and animator could be realized as VRML 2.0. This would give access to the emerging VRML web browsers. A separation between animator and renderer could be realized as VRML 1.0.

It is important to note that this model is only intended as a reference model. Any particular implementation could choose to ignore the separation between processes if it wished - but logically the processes take place however the implementation is performed.

Another advantage of the logical separation is that it identifies places where intermediate data may be stored. This allows playback of parts of the visualization if required, without the need to re-run the algorithm.

Finally the model helps us to understand how to convert existing applications. For example, the animator description language of Xtango [15] is at the same logical level as the animation primitives in the model. By creating a tool that translates Xtango animation primitives to VRML 2.0, then Xtango applications could be used over the WWW.

6.3.5 Examples

Consider the following parallel sorting algorithm, in which numbers are initially placed in a sequence of cells. Suppose there are an even number of cells. At each clockstep, there is a comparison between adjacent cells and data is swapped if necessary.

At alternate clocksteps, the comparisons are between cells:
A : (1, 2), (3, 4), ... (n-3, n-2), (n-1, n)
B : (2, 3), (4, 5), ... (n-2, n-1)
Sorting is guaranteed in n steps.

At the beginning the animator would receive requests to create the circles (associating identifiers with them) and to create text objects inside the circles with the initial values (also with associated identifiers). During the execution of the algorithm the mapper would issue movement commands to the animation process to interchange pairs of text objects.

The algorithm can pass the following to the visualization generator:
- unsorted list of numbers
- at each step, a list of operations of the form:
  exchange elements (i, j) of list

This could be visualized by a map process which did the following:

- represented the list as a sequence of circles with numbers inside each;

- represented each exchange operation as an animation of the exchange of numbers between circles.

Parameters to the map process could be the size and color of the circles and the speed of the animation.

The render process would take the geometric output of the map process and generate an image.

Next we provide an example of a visualization tool that fits into the reference model.

BALSA [2, 3] is an animation tool in which algorithms are annotated in order to produce a trace of the events of interest. In BALSA, the mapper process is termed the modeler and the filtering process is termed the adapter. Finally, the animator and renderer are collapsed into what BALSA calls the render process.

6.4 Enabling technologies

Java provides good tools for implementing visualizations and visualization generators. In particular, the Abstract Windowing Toolkit (AWT), the thread interface, and the network interface can be used to advantage. More important, however, is the machine and operating system independent graphics model that is provided, as well as the easy world wide distribution channel that is assumed.

Existing Java libraries are sufficient to implement MVC applications. Standalone, 'one off,' visualizations are easy to build with the AWT as either applications or applets. However, the ability to build visualization generators would be greatly enhanced by various, not yet extant, extensions of the Java libraries. In particular, a library of 'Computer Science data structures' such as trees and graphs would be useful here.

Likewise, VRML (especially the expected VRML 2.0) provides an enabling technology for machine/operating system ndependent graphics. Generating VRML code, perhaps through an intermediate stage of an internet MIME type is an excellent means of making visualizations widely available. VRML 2.0, with its ability to attach Java code to a visualization description, permits the additional dimension of animation.

7 Repositories

In spite of the wealth of development environments, the creation of visualization tools remains a difficult and time-consuming task. This is partly evidenced by the small number of available visualization tools. Given both the difficulty of visualization tool creation and their usefulness in teaching, an effective means of sharing these tools is called for.

With current technology, an effective means for sharing visualization tools would be a web based repository. This form of sharing has a number of benefits:

- Provides a single 'location' for both tool users and potential developers to investigate the set of already existing tools.

- Provides tried and tested visualization tools for anyone to use.

- Facilitates new development to build upon previous work; both in the elimination of duplication and in addressing the shortcomings of previous work. This is essentially important given the high 'cost' of tool creation.

- Encourages new development. Potential developers might have greater enthusiasm to undertake new development given assurances it will be both original and widely used.

Repositories, such as the one proposed, come in two types; peer review (for example, NetLib) and editor (or site maintainer) review (for example, Sunsite). There are pros and cons associated with each type. With a peer review repository there is a high degree of quality assurance since only those visualization tools favorably reviewed by a set of 'qualified' reviewers are present. Unfortunately, this quality assurance requires a long lead time between the time of tool 'submission' to the repository and when it is made widely available (the turnaround time). Also the fear of rejection may both encourage developers to create high quality software as well as inhibit potential developers from undertaking a new project.

An editor reviewed repository typically has a lower degree of quality assurance than one which is peer reviewed. Alternatively the turnaround time is much shorter under an editor review scenario. Finally, without any fear of rejection, a larger number of individuals may decide to undertake visualization tool development.

Regardless of which style of repository is implemented, the repository should contain the necessary data to simplify investigation. Repository entries (which are categorized by topic, platform, etc.) are each accompanied by a description of what the visualization tool does and how it is envisioned to be used. This descriptive data, which should be provided in a uniform and searchable format, will either be provided by the tool's author or the repository editor (or both).

There are a number of existing initiatives which could be expanded to provide the proposed repository.

- Scott Grissom at http://www.uis.edu/~grissom/VISUALS has begun an editor review repository for visualization tools. Unfortunately this repository, besides being very incomplete, does not provide the searchable tool descriptions described above.

- The ACM group SIGCSE has proposed a peer review repository to contain C.S. labs. This effort is being directed by Deborah Knox. [10]

- In [6] Michael Goldweber has proposed a peer review repository to contain large courseware systems. (that is, O.S. simulators, D.B. environments, etc.)

We observe that regardless of whether one of the above initiatives is expanded to incorporate the proposed visualization repository, or a new initiative is begun, the success is heavily dependent upon the acceptance of the repository by the C.S. education community. Acceptance is typically based upon the quality of the material in the repository and how widely the repository is known. We strongly recommend the continued development of such a repository.

References

1 Allemang, D., Aiken, R.M., Almassy, N., Schreter, Z., Wehrle, T., Rothenfluh, T., Teaching Machine Learning Principles with the Portable AI Lab, *International Conference on Computer Aided Learning and Instruction in Science and Engineering*, 1991(CALISCE'91).

2 Brown, Marc H., *Algorithm Animation,* MIT Press, Cambridge, MA,1987

3 Brown, Marc H, Exploring Algorithms Using Balsa-II, *IEEE Computer*, May 1988

4 Brown, Marc H. and Marc A. Najork, Algorithm Animation Using 3D Interactive Graphics, *DEC SRC Research Report 110a*, September 1993

5 Foulser, David, IRIS Explorer: A framework for investigation, *Computer Graphics,* vol. 29, no. 2, 1995, pp 13-16

6 Goldweber, Michael, Proposal for an On-line Computer Science Courseware Review (poster), *Proceedings of the SIGCSE/SIGCUE Conference on Integrating Technology into Computer Science Education,* (Barcelona, Spain, June, 1996)

7 Haber, R. B. and D. A. McNabb, Visualization Idioms : A Conceptual Model for Scientific Visualization Systems, appearing in *Visualization in Scientific Computing,* edited by G.M. Nielson, B. Shriver and L.J. Rosenblum, IEEE Press, 1990

8 Hannay, D. G., Hypercard automata simulation: finite-state,pushdown and Turing machines, *SIGCSE Bulletin,* vol.24, no.2, p. 55-8

9 Ingargiola, G., Hoskin, N., Aiken, R., Dubey, R., Wilson, J.,Papalaskari, M.A., Christensen, M., & Webster, R., A Repository that Supports Teaching and Cooperation in the Introductory AI Course, *ACM SIGCSE Proceedings of the Technical Symposium on Computer Science Education,* 1994.

10 Knox, Deborah and Ursula Wolz, Report of the Working Group on Computing Laboratories, *Proceedings of the SIGCSE/SIGCUE Conference on Integrating Technology into Computer Science Education,* (Barcelona, Spain, June, 1996)

11 Miller, Barton P., What to Draw? When to Draw? An Essay on Parallel Program Visualization, *Journal of Parallel and Distributed Computing,* vol. 18, 265-269 (1993)

12 Roberts, Eric, Tools for Creating Portable Demostration Programs, *Proceedings of the SIGCSE/SIGCUE Conference on Integrating Technology into Computer Science Education,* (Barcelona, Spain, June, 1996)

13 Roman, Gruia-Catalin and Kenneth C. Cox, A Taxonomy of Program Visualization Systems, *IEEE Computer*, December 1993

14 Shute, Valerie J. and Glaser, Robert, A Large-Scale Evaluation of an Intelligent Discovery World: Smithtown, *Interactive Learning Environments*, Vol 1(1) March 1990 (Ablex Publishing Corp, Norwood NJ).

15 Stasko, John, Tango: A Framework and System for Algorithm Animation, *IEEE Computer*, September, 1990

16 Stasko, John and Albert Badre and Clayton Lewis, Do Algorithm Animations Assist Learning? An Empirical Study and Analysis, *Proceedings of the INTERCHI '93 Conference on Human Factors in Computing Systems,* (Amsterdam, Netherlands, April, 1993)

17 West, T.G., *In the Mind's Eye*, Prometheus Books, 1991

18 West, T.G., Visual Thinkers, Mental Models and Computer Visualization, appearing in *Interactive Learning through Visualization*, edited by S. Cunningham and R.J. Hubbold, Springer-Verlag, 1992