Making Geometry Visible: An introduction to the Animation of Geometric Algorithms

Alejo Hausner and David P. Dobkin *

Computer Science Department
Princeton University
Princeton, NJ 08544 U.S.A.
email: {ah,dpd}@cs.princeton.edu

May 15, 1996

This chapter surveys the field of geometric algorithm animation, which has flourished in the past five years. Algorithm animation uses moving pictures to explain a computer algorithm to students learning programming, to algorithm researchers, or to programmers involved in debugging. The chapter surveys systems used to create animations of algorithms, focusing on those that specialize in geometry. It considers a selection of animation, pointing out some useful techniques. It continues with some consideration of general design issues and specific animation techniques, and closes with a look at the future.

1 Introduction

Which would you prefer?

*This research was supported in part by the National Science Foundation under Grant CCR93-01254 and Dimacs, an NSF Science and Technology Center.

Imagine yourself learning a complicated new algorithm that solves a geometric problem. You might read a book or a research paper that explains the algorithm. You probably will have to pore over the algorithm's code, which is a difficult way to learn complicated algorithms! Even if the author had provided you with simplified pseudocode, you might still need a long time to understand the details of the algorithm.

Now suppose an animation of the algorithm is available. Instead of having to imagine what is happening to the algorithm's variables, you see colored shapes which represent the program's data. These shapes move and change as the program run. A triangle does not appear as the coordinates of three points, but is drawn on the screen, changing as the program worked on it. As you choose different inputs, the animation will be different. You can slow the animation down, to better understand complicated parts, or speed it up to skip what you already understand, or even run it backwards to recall where things used to be.

Which scenario would you prefer, reading about the algorithm or watching it in action? If you prefer the animation, then this chapter is for you.

Who should read this?

Anyone who explores geometry and computation may find algorithm animation useful. Explorers of geometry can be mathematicians, users or designers of geometric algorithms, or students. All of them need explanatory tools, whether they are learning about the subject or searching for new solutions to a problem. Computer-based geometric visualization and algorithm animation can provide such an aid. Visualization is the creation of images to describe an abstract idea. Computer-generated images have advantages over hand-draw ones. They are accurate, can be modified with ease, and in the case of three-dimensional geometric objects, they can be examined from any angle.

It's not conventional animation

Algorithm animation visualizes the activity of an algorithm through moving images. As in conventional animation, the movement of images is not real, but is simulated by presenting the viewer with a sequence of still images. Unlike conventional animation, however, in algorithm animation the sequence of images is not fixed. Each image represents the state of the algorithm at some point in time. Since the inputs to an algorithm can vary, its behavior will vary and hence so will the animation of that behavior.

You've already seen it.

In a way, algorithm animation is as old as Euclid. It is not too far-fetched to say that a constructive geometric proof is a form of animated algorithm. For example, the proof that one can construct an equilateral triangle with a given side (Proposition 1) instructs the reader to draw two circles centered at the sides' endpoints and to draw lines to the circles' intersection. The computer in this case is a person armed with compass and straightedge, who is following instructions and reaching a result, so the procedure qualifies as an algorithm.

The need for animation.

Some explorers of geometry devote their time to designing algorithms that solve geometric problems. Even though their solutions will run on a computer, they often find themselves working "in the dark", needing to visualize not just the objects they are manipulating but also the process by which the algorithm modifies the objects. Geometric algorithms are often more difficult to design, program and debug than other algorithms. This is because the designers often lack the means to see the objects their algorithms are manipulating, and are forced to pore over printouts of point coordinates. Algorithm animation can help.

How can algorithm animation help you?

There are several reasons for animating an algorithm. First, a theoretician who is devising geometric algorithms needs a medium for disseminating his/her ideas. Although research papers satisfy this role, a well-crafted animation on videotape can communicate an idea more quickly than a printed description. If the author makes an interactive animation available on the world-wide web, people can learn the algorithm more easily. Second, a teacher who is presenting an algorithm to students can benefit in the same way. However, the audience in this case will not consist of experts as in the first case, so the animation must be prepared with more care. Third, animating an algorithm can stimulate the designer into finding new variations. In effect, such an animation explains the algorithm to its author! A geometric algorithm may need to handle special or degenerate cases, and animation often helps identify these cases. Finally, algorithm animation can help with debugging of general programs, although this application turns out to have limited scope.

It will improve further

Algorithm animation is a young discipline, and will probably change in the next few years. The changes will come both from improvements in software and hardware.

Although animation as a medium can make an algorithm more understandable, it also has some limitations. For example, there are some algorithms that are rather complicated, and are difficult to understand even with a well-crafted video. In an animation where many things are happening simultaneously, the user will want to stop and ask questions, such as "why didn't the scan line advance when that point was processed?". Hypertext documents allow the viewer to ask such questions, but in the past they have portrayed either fixed images or fixed animations, and this limitation has reduced their explanatory powers. There are situations where the user wants to interact with a running the algorithm, and also wants to see or hear other information about it. There is a great potential for improvement in algorithm animation through web-based programming languages like Java, which present the user with running programs and not just static images.

Animation places great demands on computer resources. If a prepared animation is made available over the internet, it will need large amounts of storage space, and more importantly, a very high-speed connection to the net, lest the users spend hours downloading a five-minute movie. On the other hand, if the user does not download the data but rather runs a program that creates an animation in real time, he/she may need a very fast computer, lest a complicated five-minute animation take 30 minutes to run. A similar problem exists if the user requests a three-dimensional scene stored as VRML data. If the scene is complicated, the rendering speed will suffer, and hence, so will the user. Increases in storage, communication bandwith and processing power will make possible new algorithm animations and geometric visualizations.

Chapter Overview.

Algorithm animation has its roots in educational film and in mathematical visualization. It has also been influenced by recent developments in scientific data visualization, which is at present an extremely diverse and active field. Early efforts in algorithm animation in the late 1970's resulted in an educational film [Ba81] which is still used today. The middle 1980's saw the development of systems [Br88b][LD85] which allow a user to run algorithms and see animations interactively. Recent efforts have added color and sound [Br91n][Dg92], as well as the use of object-oriented programming [Br91a][AR93][Sc92][EK94]. Section 2 provides a survey of systems used to create animations of general algorithms. There are excellent overviews of the field [PB93][Ta95], so in this one we will concentrate on questions that a designer of similar systems must face. Geometric algorithms present special opportunities and difficulties, and systems to make animations for them have also evolved [Sc92][Ge95][TD95]. Section 3 gives a survey of these systems. Again, the survey will not be exhaustive but will focus on technique.

The past five years have seen the field of algorithm animation flourish. Rather than require the audience to install the animation software on their own computers, researchers have found it convenient to use video tape to record representative sessions. Section 4 surveys these video recordings. The videos selected for discussion use interesting animation techniques.

Whether a researcher is creating an animation or is planning a system to assist in their creation, he/she must consider some general issues. These are covered in section 5. Finally, section 6 presents some techniques which can be used in an algorithm animation to convey ideas more clearly.

2 General systems

2.1 Sorting out Sorting

The most well-known example of an early algorithm animation is the film "Sorting out Sorting", created by Ronald Baecker [Ba81] and presented at Siggraph '81. It explains concepts involved in sorting an array of numbers, illustrating comparisons and swaps.

The film ends with a race among nine algorithms, all sorting the same large random array of numbers. It shows the state of the array as a collection of dots in a rectangle; each dot represents a number: x is the array index and y the value. In a single screen, nine rectangles with moving dots vividly explain the behavior of nine algorithms. The audience can compare the $O(n^2)$ and $O(n \log n)$ running times of insertion sort and quick-sort. Not surprisingly, quick-sort finishes first. The movements of the dots also convey the method used by each algorithm.

The film was very successful, and is still used to teach the concepts behind sorting. Its main contribution was to show that algorithm animation, by using images, can have great explanatory power. The film also demonstrated that tools were needed. The effort involved in its production (three years using custom-written software) stimulated research into making more general-purpose algorithm animation systems.

2.2 BALSA and Zeus

Many current systems have their roots in BALSA [BS84][Br88b], developed by Marc Brown and others in the middle 1980's. Its influence has been enormous, and by looking at its design one can learn about the problems that any algorithm animation system must solve.

BALSA and BALSA-II are general purpose animation systems, which means that in principle they can be used to animate any algorithm. They are primarily used to supplement the teaching of computer programming.

Events: BALSA introduced the paradigm of the *interesting event*. Not all steps in an algorithm need to be visualized. With some thought, one can identify points in an algorithm where significant changes take place, and mark them. For example, in a sorting algorithm, comparisons and swaps are events. The idea of events reappears in all animation systems after BALSA.

In order to animate an algorithm, a programmer using BALSA must implement the algorithm in some programming language, and then modify the resulting program by inserting special procedure calls at the places where interesting events happen.

Visualizing events: The events must now be visualized. To continue with the above example, an array of numbers being sorted could be visualized as a row of thin rectangles, each with height proportional to the number it represents. A comparison event might be visualized by highlighting two rectangles, and an swap event by making two rectangles change places. BALSA leaves the task of visualization to the programmer, who must write procedures to visualize events. A collection of these procedures is called a *renderer*, and produces a *view* of the algorithm. BALSA takes care of calling these procedures at the appropriate time.

Although in principle any algorithm can be animated, BALSA was intended primarily as an educational tool, and that was indeed its main use. Modifying large programs is a laborious task, and hence it is difficult to use BALSA as a debugging tool in large-scale software development. Moreover, a novel algorithm requires the programmer to write new procedures to visualize events, and this task often proves difficult. On the other hand, because so much is left up to the programmer, the system is very flexible. Clearly, BALSA illustrates a tradeoff between flexibility and ease of programming.

Aids to writing renderers: Algorithm animation can be a laborious task, and for that reason BALSA supplies some facilities to make the task easier. When an algorithm uses several renderers, a *modeler* stores a single representation for the state of algorithm, preventing the duplication of data. Sometimes a new algorithm resembles a previously-animated one, and an *adapter* can be written to call procedures in the old algorithm's renderer, thus saving a great deal of programming.

Aids to the user: Once an algorithm has been modified, BALSA lets users experiment with it, and watch the resulting animations. Several features make experimenting easier.

BALSA allows the user to simultaneously invoke several algorithms which solve the same problem, in order to learn which one runs fastest. He/she can also run races among several instances of the same algorithm, with each instance taking a different amount of time to perform a given subtask. For example, the user can vary the times taken by comparisons and exchanges, to learn how they affect the running time of a given sorting algorithm.

There are cases where a user wants to test an algorithm with a certain type of data, and it may unreasonable to ask him/her to enter a large amount of input data satisfying some criterion. BALSA provides a solution though *input generators*. These are menu-driven procedures with a few parameters, which will generate data for a given algorithm. The user can configure an input generator interactively. An example is a generator which produces a nearly-sorted array of numbers, which can be used with several sorting algorithms to see which one best handles this nearly-degenerate case.

Zeus After completing his work on BALSA, Marc Brown and others developed Zeus [Br91a], an object-oriented algorithm animation system similar to BALSA.

Aware of the difficult task of building renderers, Brown added several features to Zeus that make implementation easier. *Zume* is a custom preprocessor which generates procedure definitions for renderers based on a file of prototype events, and which thus makes it easier to annotate an algorithm with event calls.

Another aid to writing renderers is a multi-view editor, which simultaneously displays a graphical image of a view and the textual code which specifies it. The programmer can manipulate either the graphics or the text, and observe the result. This frees the programmer from the need to specify precise positions for graphical elements, for example.

The multi-view editor also introduces an interesting feature: The user can change the behavior of the algorithm while it is executing, simply by manipulating one the views. For example, consider a euclidean graph algorithm: the user can move a node in a planar graph to a new location while the algorithm is running, and observe the effect this has on the algorithm's behavior.

2.3 "Descendants" of BALSA

AnimA and GeoLab AnimA shares many features with BALSA. AnimA is a system for algorithm animation, and it is written using the facilities of GeoLab, a system for geometric computation.

AnimA [AR93] runs within GeoLab [RJ93], and both were developed by P.J. Rezende and others in the early 1990's. AnimA is a general purpose algorithm animation system, and implements many of the features of BALSA, such as renderers, adapters, and input generators.

The facilities of GeoLab give users flexibility. To run an animation, a user can choose a problem, an algorithm for that problem, and an input generator for the algorithm. He/she can then run the algorithm, and step through the program events, pause or abort. Dynamic linking allows all the components to be chosen at run-time.

AnimA uses GeoLab's facilities for program development, which includes a specialized editor that controls access to source code to make it possible for a group of developers to work on large projects consistently.

GeoLab GeoLab is an environment for programming geometric algorithms, It is object-oriented and dynamically linked. Within GeoLab, each geometric object has two representations: pure form and graphical form. The purely geometric form of a line segment, for example, is the coordinates of its two endpoints. The graphical form of an object describes how the object should be drawn on the computer screen. Since GeoLab is object-oriented, this description is encapsulated and can consist of programs as well as data.

A user can execute a program by choosing data interactively and applying a program to the data. A dispatcher takes care of the task of extracting the *pure* data from the user's choice, sending it to the program, then adding the *graphic* data to the program's pure output. This feature is used in AnimA.

Tango Tango [St90a] was developed by John Stasko in the late 1980's, and is a general purpose system for algorithm animation.

One of the novel features of Tango is its emphasis on smooth animation. Human audiences find it difficult to keep track of events if they occur suddenly and discontinuously, and can understand gradual changes more easily. Tango provides *path transitions* [St90b], which make it easier for the programmer to specify gradual changes that correspond to program events. The programmer can give the path an object will follow as it moves from one place to another. He/she can also decide how the object's color will change during the transition, or whether it will fade in or out.

Stasko developed an algebra for specifying how to build a transition from several others, by concatenation (which means they happen one after the other), and compositions (which means several transitions happen during the same time period).

Recently, Stasko has made his system accessible interactively through the internet [St94], for anyone with a web browser and X-windows.

Movie and Stills An interesting approach to algorithm animation appears in Movie and Stills [BK91a], programs developed by Bentley and Kernighan. These authors use the Unix text-filter paradigm.

In this system, a program is animated by inserting output statements, and not function calls. Each output statement produces a textual description of the changes in the state of the program. This text must be written in a very simple language for describing shapes to be drawn and also identifying program events. In effect, by thus modifying a program, it will itself output a program which, if properly interpreted, results in an animation. One advantage of this approach is language-independence: the original program can be written in any language and run on any machine, since the animation information is communicated through simple ASCII output.

Two utilities are provided to interpret a program's output. The first one, Movie, animates the program on a workstation's screen. The user can pause the animation, speed it up or slow it down. The other tool, Stills, is a `troff` filter which allows frames from the animation to be included in printed documents.

In a way, this system resembles Tango and others which break the algorithm and animation into separate processes. The difference here is that the process communication occurs through pipes or intermediate files, and is in only one direction. This restriction makes it more difficult to animate programs that require graphical input, since the user interface must be written by the programmer. Nevertheless, the system is very easy to learn and use, and can animate many algorithms satisfactorily.

Summary: Animating General Algorithms There are several concepts that are relevant to all the animation systems presented in this section. First of all, the idea of an *interesting event* appears in all the systems. Each time an event occurs, it may lead to a change in an image which the user is viewing. This image visualizes the current state of the program. As the program runs, the state changes, and so does the image. In other words, strictly speaking, what is visualized is not an *algorithm* but rather its *data*. From the changes in the image, the user is expected to infer the logic of the algorithm.

This leads to another issue, which is that not all events can be visualized with equal ease. It may be easy to write a procedure that swaps two rectangles when two numbers in an array are exchanged, but more effort is needed when representing the act of inserting a key into a red-black tree. Because writing renderers can be difficult, systems like BALSA, Tango and AnimA provide facilities for re-using renderers from related algorithms.

3 Geometry-oriented Systems

Designers of geometric algorithms face special challenges. While much of their discipline uses visual objects, computers deal with numbers and symbols. As a result, geometric objects in a computer program must be represented in a form which is difficult to translate back into images. Hence computational geometry can benefit from visualization.

Another reason for visualizing geomtric algorithms is its relative ease. As we saw in the previous section, the most laborious task in writing an animation is writing renderers that portray the program data as images. In the case of geometric algorithms, the program data contains positional information, and hence can be drawn on the computer screen without the need for translation.

For these reasons, since about 1990 several systems have evolved that try to satisfy the needs of computational geometers. This section examines them.

3.1 Two-dimensional Systems

XYZ Geobench Peter Schorn's *XYZ Geobench* [Sc92], created around 1990, provides tools for writing and testing geometric algorithms. It provides a library of geometric abstract data types.

The Geobench emphasizes robust computation. For example, a poorly-implemented algorithm for segment intersections may not detect points where more than two segments coincide. To this end, the Geobench provides tools for creating degenerate data which is useful for torture-testing algorithms.

Aside from being limited to two-dimensional problems, the GeoBench's animation facilities are somewhat limited. Basically, a programmer wishing to animate an algorithm must visualize the program events directly. This means that for each event, the programmer must write code which draws the geometric changes, pauses to let the user see them, then continues.

Workbench In 1994, Epstein, Kavanagh and others published a paper describing their *Workbench for computational geometry.* They were concerned that many published geometric algorithms have not been implemented, and that the costs of many published algorithms are given only in asymptotic form. Hence they wanted to get actual measurements of the cost behavior of implemented algorithms.

The Workbench resembles the Geobench. The authors wrote a library of abstract data types and of elementary (constant-cost) geometric operations. Using this library, they wrote several currently-proposed algorithms in computational geometry. The algorithms implemented addressed the following two-dimensional problems: convex hull, polygon triangulation, visibility, point location and line segment intersection.

The workbench includes a facility for animation. For each algorithm implemented, there is a corresponding animation which can be as simple as rendering all the changed objects in the algorithm's current state. The system separates the algorithm from its animation, because animation affects the time and space requirements, and a major goal of the workbench is measuring these requirements. The workbench builds blocks of code which are passed to the animation if one is active, or else discarded. In this fashion the same code can be measured or animated.

3.2 Beyond Two Dimensions

The emphasis of both the Workbench and the Geobench is robust geometric algorithms, not glossy renderings. Moreover, their algorithms are restricted to two-dimensional geometry. Two systems choose a different approach, concentrating on the presentation of geometry in three and higher dimensions.

Geomview Geomview [Ge95] is a group effort from the Geometry Center of the University of Minnesota. The program is essentially a sophisticated viewer for geometric data. Users can interactively control the appearance of objects, by changing the lighting, shading and material properties of the surface. Phong shading leads to realistic-looking objects. Moreover, three- and four-dimensional objects can be viewed with perspective, orthogonal, hyperbolic and spherical projections.

The appearance and display of geometric data can also be controlled through software. Geomview includes an interpreter for a lisp-like command language, through which a program can control all of the system's features.

Objects displayed by Geomview can consist of polyhedra, quadrilaterals, meshes, vectors and Bezier patches. The description of an object can either be stored in files, or can be generated by a separate program which sends its output to Geomview through a pipe.

This latter feature makes it possible to animate geometric algorithms with Geomview. A programmer needs to modify a program to output descriptions of geometric objects in a Geomview format, and in this way use Geomview as means to visualize the program's behavior. This approach is also the way to animate an algorithm, since Geomview has no built-in methods for animation.

An enthusiastic programmer can also use Geomview for interaction with a user, since Geomview can be configured to inform a program if a user selects a geometric object with the mouse.

Geomview's versatility is shown in the wonderful movies [EG91] [LM94] [LM95] that have been made with its help. Of course, with all that versatility comes complexity: not only is Geomview a very large piece of software, but also users may take some time to learn its special features.

GASP GASP [TD95] aims to provide high-quality animations of geometric algorithms with minimal programmer effort. It tries to achieve the high-quality renderings of three-dimensional objects that Geomview provides, while minimizing the effort needed to create them.

The spirit of GASP is to separate geometry from animation. To animate a geometric algorithm, a programmer inserts some function calls which tell GASP which geometric objects have been added or removed. This is usually easy to do, since the coordinates of the geometric objects are already known to the program. Based on the information given to it in the function calls, which tell it of the changes in the scene, GASP then decides how to represent these changes.

Changes in a GASP animation are grouped into *atomic units*. These correspond to transitions in TANGO. Within each atomic unit, there can be several changes to the data. For example, several facets of a polyhedron may be simultaneously removed during one atomic unit. In the spirit of TANGO, the changes are usually smooth.

The changes to a scene can be represented in several ways. A new object can fade into place, blink as it appears, or move in from outside the scene. Conversely, removed objects fade out gradually, or move away. There are defaults for all the aspects of how the change is represented, including the type of change, the time it takes, the colors used, etc. These defaults are stored in *style files* which are simple text files that the programmer can modify as necessary. This use of defaults means that the novice programmer can produce simple animations without learning all the details of the system.

The system includes a user interface that follows the model of a video tape recorder. The user can view the animation forwards or backwards, at normal or high speed.

Advantages of GASP include its ability to represent 3D geometry, high production values and good color facilities. In particular, the range of colors that are attractive and easy to see on a computer monitor is not the same as the colors that are suitable for recording on video tapes. Bearing this difference in mind, the style files in GASP can be set to use colors most suitable for the output medium being used. Another good aspect of GASP is that it frees the programmer from learning about computer graphics or rendering of three-dimensional geometry. Programmers are free to concentrate on geometry.

Of course, GASP does have some limitations. The most obvious one is that it is mainly limited to geometry. This means that, if used to animate general algorithms, the programmer must be familiar with geometry. Another drawback lies in the rewind facility which is slow, although this difficulty arises from the implementation and not from a weakness in the design. It should be simple to remedy.

4 Visualizations

In any field, skill in the use of tools is acquired not only by studying the tools themselves, but also by studying the products of other skilled workers. Whereas in the previous sections we considered the tools, *ie* systems which can be used to create animations of algorithms, in this section we focus on the products of these tools, the actual animations.

This section will focus mainly on videos published in the *Annual Video Review of Computational Geometry*, which is the main vehicle for dissemination of techniques in the field The field of algorithm animation is rich, and still growing. As they seek to explain algorithms, researchers borrow techniques from related disciplines such as scientific and mathematical visualization, and interactive environments. As a result, we find in these videos examples not only of algorithm animation [VR1a,c,d,f,g,i,j, 2b,c,d,e,h, 3a,b,c,g, 4b,d], but also mathematical visualizations [VR3e,4e,4f] and interactive visualizations [VR1e,2a,3f,h,4a,c]. In addition to the *Video Review*, we will also briefly discuss three films from the Geometry Center [EG91] [LM94] [LM95].

This section does not aim at a thorough review of *all* the videos extant, but rather seeks to present specific animation techniques which are used in some of them. These techniques (*i.e.* tricks used twice!) turn a video into more than simply “dancing data”. Animations may depict how an algorithm processes its data, but they should also explain the reasons for particular actions.

4.1 Mathematical Visualization

Before considering animations of geometric algorithms, it may help to consider three movies produced at the Geometry Center of the University of Minnesota. They were all produced with Geomview, as well as other software.

Not Knot [EG91] introduces the viewer to knot theory, by considering the complement of a knot. Visualizing the complement of the Borromean rings leads to a brief excursion into hyperbolic geometry. *Outside In* [LM94] visualizes the remarkable theorem which states that a sphere in three-space can be smoothly everted, in other words, turned inside out. *The Shape of Space* [LM95] visualizes toroidal space. It explains, in more detail, some concepts that were brought up in *Not Knot*.

All three films show that a great deal of effort went into their production. They achieve the difficult task of explaining problems in topology to novices. The production costs of these films are relatively large. For example, *Outside In* took 2-3 person-years of programming effort, in addition to the time needed to develop general-purpose tools for geometric visualization. Once the tools were developed, production was easier. The last film, *The Shape of Space* was produced in only four months. Even with proper tools, the need to make the film accessible to novices required additional effort. If a concept taxes the audience's geometric imagination, then its explanation will tax the film-maker's imagination.

A short video, *Animating Proofs* [VR4f] uses a couple of techniques to effectively illustrate Pythagoras' Theorem. The narrator reads the proof of the theorem from Euclid's *Elements*, and whenever a geometric object such as a line or point is mentioned, the corresponding graphical element is highlighted. Through this simple trick of timing, the burden of referring back and forth between proof and diagram is avoided. Another simple trick is used when two triangles are shown to have equal areas. One triangle is smoothly deformed into the other.

4.2 Demonstrations

Several of the video segments [VR1b] [VR1h] [VR2f] [VR3d] contain animations of algorithms, but are primarily intended to demonstrate the capabilities of an algorithm animation system. To enable the viewer to concentrate on the capabilities of the system, these demonstrations tend to animate well-known algorithms, *e.g.* Graham's scan. For this reason, they will not be considered here.

4.3 Algorithm Animations

Optimal Two-Dimensional Triangulations [VR1c] shows a typical use of color. The video animates versions of an algorithm that computes optimal 2-d triangulations. Each triangle is colored according to some measure, such as maximum vertex angle. Optimal triangulations minimize that measure. As the algorithm runs and the triangulation is improved, its overall color approaches the optimal one.

Boolean Formulae for Simple Polygons [VR1d] uses color and also multiple views to convey information. The algorithm computes the boolean formula for a polygon: Given the half-planes determined by each edge of the polygon, the boolean formula is the combination of OR's and AND's of these halfplanes that defines its interior. For a convex polygon, AND's suffice, but concave polygons require OR's as well.

The animation uses multiple views, which simultaneously show the state of the algorithm in different ways. One view shows a tree of partially-satisfied polygons, and another shows the same tree in schematic form. Through the schematic tree, the user sees at glance how certain types of polygons lead to very unbalanced trees. Another view shows, in text form, the boolean formula as it is built, and finally there is a view that shows the edges of polygon being processed by algorithm. Each view corresponds to a way of abstracting the information being processed by the algorithm.

Like the previous animation, *The New Jersey Line-Segment-Saw Massacre* [VR1j] uses multiple views to explain an algorithm, this time Chazelle and Edelsbrunner's optimal sweepline algorithm for general line-segment intersections. As the sweepline advances, a separate view shows the red-black tree holding partially-processed segments. The video is difficult to understand, mostly because the algorithm itself is very complicated. Perhaps many more views would have shown in detail what was going on, but the authors may have feared overloading the viewer's ability to keep track of too many simultaneous changes. The time limitations of the video also preclude the necessary drawn-out explanation.

Building and Using Polyhedral Hierarchies [VR2b] shows that often multiple views are not needed. It shows a polyhedron being simplified as certain vertices and their incident edges are removed. Implicitly, a hierarchy of simpler polygons is built, although the viewer does not need a visualization of it to understand it. The video also shows the use of color to encode time information: each changed polygon facet is drawn in a new color, and this trick allows the viewer to see the algorithm's history at a glance.

In theory, the systems described above can be used to visualize a new idea, with the visualization appearing simultaneously with the research paper that presents the new concept. In practice, however, most algorithm animations describe ideas which are already well-known. *Objects that Cannot be Taken Apart With Two Hands* [VR2h], is remarkable for appearing at the same time as the discovery it explains.

An Animation of a Fixed-Radius All-Nearest Neighbors Algorithm [VR3c] is an excellent video. The algorithm described here solves the following problem: Given a set of points and a distance δ, find groups of points where each member of the group is within δ of the others. The algorithm uses a scan line to partition points into an adaptive grid, made up of horizontal and vertical slabs of width $\geq \delta$. Occupied grid entries are identified by labelling each point with its grid coordinates, and then sorting the list of labels. Points with the same label lie in the same non-empty grid square. Finally, these grid entries are scanned for pairs of points within δ of each other, and these pairs are reported.

The animation is skillfully made. The narration is well-synchronized with the video. For example, labels appear just after they are mentioned. Skill is also evident in the use of multiple views. One view shows the search tree which stores point labels. The sorted list of labels corresponds to bottom level of the tree. The animation first shows this list, then fades in the search tree on top of it. This approach is preferable to showing the insertion of each label, which leads to much confusing extraneous activity.

The video closes with a three-dimensional application of the algorithm: finding an isosurface of a molecule. This section is harder to follow, perhaps because elements are not labelled, forcing the viewer to remember the identity of each element.

4.4 Interactive Visualizations

This section examines a class of systems which are better classified as interactive visualizations than as algorithm animations. The videos selected here demonstrate interactive systems which illustrate an algorithm through its results. By contrast, most algorithm animations show the viewer an algorithm's behavior. With an interactive visualization, the user can change some graphical data interactively, and immediately see the resulting structure produced by the (usually geometric) algorithm under consideration.

Visualizing Fortune's Sweepline Algorithm for Planar Voronoi Diagrams [VR2a] does animate that algorithm for *building* a Voronoi diagram, but primarily it showcases an interactive tool for exploring the *properties* of Voronoi diagrams. Given a set of points in the plane, the user can turn several structures on and off. These structures are the points themselves, their Voronoi diagram, Delaunay triangulation, or convex hull. The user can also add an abitrary paraboloid of revolution, project points onto it, add tangent planes, and verify that the paraboloid's lower hull indeed corresponds to the Voronoi diagram. Since this is a video tape, the viewer can't control the system, but must watch a recording of the author's interaction with the system. Nevertheless, the video shows the advantage of giving the user the ability to turn aspects of a visualization on and off.

Exact Collision Detection for Interactive Environments [VR3f] presents a real-time system that detects collisions between polyhedral objects. If two features like a vertex and an edge are close together, the system displays a line between them, indicating that those two features are being tested for collision. To the user, this produces the illusion of elastic filaments joining adjacent features on neighbouring objects. The video then demonstrates the use of bounding volumes. Two objects are tested for collision only if their bounding volumes intersect. The effectiveness of this acceleration technique is immediately apparent, since the filaments appear only when bounding volumes meet, and their number is greatly reduced. We can see the effect of an algorithmic technique even though we are only shown the *product* of the algorithm.

5 Design Issues relevant to Algorithm Animation systems

In this section, we consider some issues that must be considered by someone designing a system for algorithm animation. Each of the considerations below affects three classes of people: first, there is the designer of the system itself, second, the programmer who uses the system to create an animation of an algorithm, and finally the user who interacts with the animation (in some cases, one person may fall into more than one category). The design decisions considered in this section affect the facilities available to each type of person, as well as the effort that each one must expend.

5.1 Scope

One aspect that must be considered is the range of animations that can be created. These can range from special-purpose to general, and can be:

- Single-purpose: Hand-crafted code that illustrates one algorithm or a group of algorithms in detail (examples: [Ba81], [VR2a]).
- Specialized: A system that specializes in algorithms from a field of computer science, such as computational geometry (examples: [RJ93], [TD95]).
- General-purpose: Systems that can, in principle, animate any algorithm (example: [Br88b]).

Obviously, the greater the number of algorithms that can be animated, the more desirable the result. However, with increased flexibility comes increased complexity. A general-purpose system will often require much more work from the programmer than a specialized system. This occurs because of the gap between abstraction and representation. When an abstract concept such as an algorithm is visualized, methods are needed to convert internal objects into pictures. If the algorithm is restricted to a field where a given data type is used often, then a visualization for them needs to be written only once. On the other hand, a general-purpose animation system must be able to handle *all* data types, and this gives the implementer of the system two alternatives:

1. Provide the programmer with all possible visualizations

2. Provide tools for building visualizations.

The former is of course impossible, while the latter burdens the programmer who would rather concentrate on other things. A good general-purpose system tries to compromise by providing a large set of ready-made pieces of visualizations, together with general tools.

Systems that restrict themselves to animating only algorithms in a field like computational geometry, will not be able to represent many other algorithms in other fields easily, but on the other hand will be able to provide an advantage, because the system developer can devote a great deal of effort can be devoted to make pleasing, informative visualizations of frequently-used objects. This makes the programmer's task much easier.

5.2 Author and Audience

The amount of effort needed to craft an animation, as well as the type of system used, is also affected by the relationship between author and audience. The author will necessarily be a programmer, while the audience may or may not. Authors can fall into several categories:

1. A researcher in a field, who wishes to explain his/her new algorithm to colleagues in the field, perhaps at a research conference, and finds a printed description inadequate. [TD95]

2. A teacher, who must create an animation to explain some fundamental algorithm to students, who in turn will be seeing the algorithm for the first time. [Br88b][St90a]

3. A programmer, whose debugging needs exceed the capabilities of the usual debugging tools, and wants a visual trace of his/her program's flow of control.

Corresponding to these categories, the audience will consist of:

1. A researcher at a conference, who wishes to learn about a colleague's new approach, or needs further understanding of a printed journal article. [TD95]

2. A student learning an algorithm for the first time. [Br88a][St90a]

3. A programmer, examining the animation he/she just created.

Each situation demands a different kind of animation system, and it is difficult to build a system that satisfies them all. Here are some reasons:

If making animations is difficult, then a researcher will probably lack the time to create one, given that he/she must already do the research and also prepare a written description. Thus he/she will likely prefer a simple animation system which can be learned and used with as little effort as possible.

A teacher will need the flexibility of a general-purpose system, because he/she is communicating with an audience that is unfamiliar with the algorithm being animated. The student learning the algorithm will not be able to ignore irrelevant details, and keeping the animation clear and simple can often require a great deal of work on the programmer's part! For example, one insertion into a balanced tree can lead to a great deal of re-arrangement of the tree, which can confuse an inexperienced viewer. Hence the animation of the tree insertion must be carefully prepared to minimize distracting and irrelevant movement [ST92]. Thus, a teacher will probably be willing to tolerate a greater deal of learning and programming effort, in exchange for a clearer animation.

A programmer developing a large piece of software will, most likely, use animation tools for debugging. Hence he/she will want to change his original program as little as possible. It is not clear if non-intrusive animation of a general program is easy (or even possible). Nevertheless, there are systems that can help to visualize the flow of control within a program in an over-all, module-by-module fashion [FK95] [NK94].

5.3 Expressiveness

A useful system should have as many facilities as possible for expressing the idea behind an animation. Whether these facilities are used effectively or abused depends, of course, on the programmer's discretion. A good system should give the expert programmer control of:

- Color.

- Sound. [Br91n]
- Time. In some systems [TD95][St90a], the programmer can describe the way transitions in the algorithm are presented: they may be sudden or gradual.
- Three dimensionality. Most animation systems represent algorithms in a two-dimensional way. The third dimension can be used in many ways, such as [St92]
  - to represent an extra dimension in the data, such as the size of data items or some other attribute of the data,
  - to illustrate the passage of time, or the algorithm's progress [Ko91]
  - for aesthetic effect (as in the "USA-Today" graphs)
  - to represent objects which are inherently three-dimensional. [TD95]

It would seem obvious that more cues available to the reader would translate into greater understanding. However, too much color and sound can also be distracting. Here we must follow Tufte's [Tu83] principle, which is that a graphic's excellence is a measure of its information density. By this he means that a good graph will contain only elements that communicate information, and no extraneous ones. By this principle, the overuse of three-dimensionality for two-dimensional objects should be avoided. If the designer of the animation system makes an effort to choose defaults that are clear, attractive and minimize visual clutter, then the novice will be more likely to obtain good results with ease.

5.4 User Interaction

Algorithm animation is different from ordinary animation, in that the final product is not simply a film which the user watches passively. Because an algorithm has inputs, its animation can be different each time the algorithm runs. Ideally, the final user should be able to interact with an animation. Degrees of interactivity can range along the following scale:

1. None: The system produces stills, [BK91a] or a movie which the viewer must watch passively [Ba81].

2. Some: The user can stop and start the animation, or change its speed. Most systems support this level of interaction.

3. More: The user can choose the input data for the algorithm being animated [Br88b].

4. More: The user can pan and zoom to see different parts of the display [Br88b],[St90a].

5. Fair: The user can rewind the animation, and replay parts which were not understood on previous runs [TD95].

6. Good: The user choose data while the algorithm is running, such as selecting a starting node for an all-points shortest path algorithm [Br88a].

7. Excellent: The user can change the data while the algorithm runs, for example by dragging a data point to a different location and watching the effect of this change.. [VR2a]. The possibility of this last feature raises many interesting questions.

Clearly, the amount of interactivity is related to the type of algorithm being animated, as much as it is to the system being used. Some algorithms are batch-oriented, and it may not be possible or meaningful to change the data they are working on. For example, many sorting algorithms will not work properly if an array element is moved to a different position while the sort is in progress.

6 Techniques

In this section, we present some animation techniques. They could either be built into an animation system, or used by a programmer who is creating an animation. Some of the techniques are applicable to animation in general, while others apply to the visualization and animation of three-dimensional geometry.

6.1 Abstraction

There are times when the amount of data produced by a program overwhelms the user. When this happens, an animation confuses more than it educates.

If the algorithm is complex, and uses several different data structures and sub-algorithms, the user may get lost in the details and not see the over-all picture.

In such cases, the programmer should condense complicated parts of the scene into simpler items, like boxes. This is the approach taken in several videos we have reviewed. Time can also be abstracted, if several phases of an algorithm are omitted and only the final result of several program steps is presented.

An ideal system should include facilities that help the programmer implement this sort of abstraction. Ideally, all the detail should be accessible to the user if he/she needs to see it. This is called *semantic zooming*.

Sometimes, of course, it may be desirable to present the viewer with large amounts of information. This occurs when several sorts are simultaneously compared in [Ba81]. Although the screen is filled with information, the means of presentation lets the user's visual system grasp the difference both in running time and behavior of the sorting algorithms. This scene confirms Tufte's [Tu83] principle of high information density.

6.2 A Few Animation Techniques

In this section, we draw attention to some methods used to convey meaning in animation.

Relatedness: Sometimes it is necessary to associate two or more objects in the user's mind. For example, in a set of points a subset of points may be special in some way. Points may be grouped by:

- displaying them all in the same color, or
- using the same icon (such as a small triangle) to represent all of them, or
- using icons of the same size to represent all of them, or
- connecting them with arcs. This last approach should be avoided if there are many points, to avoid clutter.

Multiple views help the user understand an algorithm's progress, but they can split the user's attention. Many animation systems feature multiple views. This means that several renderings of the algorithm's state are presented on the same screen. For example, one view may present a picture of a collection of line segments and the sweep-line that marks the search for their intersections, while another view may be the search tree that stores the endpoints currently under consideration. As the algorithm runs, both views change. In order to help the user keep track of corresponding elements in the two views, the programmer should associate the elements somehow. This can be done with color, or icons as in the previous paragraph. In some cases, it may be possible to keep one coordinate of corresponding elements the same. In the case of the sweep-line algorithm, this can be done by drawing the search tree on its side: since it is ordered by y coordinate, its nodes can have the same y as the points they store.

If the animation is recorded on a fixed medium and has a narrated soundtrack, the actions being described should occur immediately after they are narrated.

State: When animating geometric algorithms, it may be necessary to convey more than just the geometric changes. Often program objects have state, which changes as the algorithm runs, and there is a need to convey these changes of state. For example, in Dijkstra's shortest-path algorithm, a node can be one of two states, corresponding to whether or not its distance from the source has been determined.

The state of an object can be denoted by color, although this method may require an explanation by the narrator. If the objects are points, they may be represented by icons which change as the state changes.

Color: Color is versatile tool. It can be used to match related elements, as described above, but its continuous nature can also be used to represent gradual changes.

Gradual change can be time, and time usually represents the algorithm's progress. We can convey history with color. If the color used for drawing is changed at each major step in an algorithm, the color of an object will encode the time it was created.

Color can also be used to represent direction in two dimensions. This follows naturally from the Hue component in the HLS color model. If the Saturation is held constant, the fact that two degrees of freedom remain in an RGB triplet means that color can also encode directions in tree dimensions.

Color can indicate a numerical values associated with objects. Here, a range of values can be represented by holding the hue constant and varying the saturation, by varying the hue. Hue variations are more effective when large collections of small objects are portrayed.

6.3 Problems specific to 3D

Why is 3D visualization different from 2D? The fundamental difference is that we live *inside* a three-dimensional world, whereas we can stand completely *outside* any two-dimensional world, such as the world on a sheet of paper. This ability to "step outside" the region of interest is often lost in 3D geometric visualization.

Here are some solutions that may be used in crafting 3D geometric animations:

Rotation: If you can't see the whole object at once, then turn it continuously so that the viewer can eventually see most of it. Rotation has the side effect of preserving the illusion that the objects on the flat screen are actually three-dimensional objects viewed through a transparent window. Without continuous rotation, the viewer can lose this sense, and start perceiving polyhedra as strangely-shaped collections of flat polygons.

Of course, rotation must not be used carelessly:

- Axis: When objects are rotated to make all sides visible, they should not be tumbled at random, changing the axis of rotation frequently. In fact, it is better to change the angle of rotation as little as possible, or not at all, simply rotating the object about, say, the vertical axis. Of course, this restriction goes against the desire to show as much of the object as possible, but is needed to keep the viewer from losing track of where things are.

- Discontinuity: Sometimes videos are prepared in several sessions, each of which may correspond to a different phase of the algorithm or to an idea that must be communicated. The animators may pause for a coffee break, and continue their video recording session later. Whatever the reason, if the same scene is being displayed in separate sessions, and the scene is rotating continuously, there may be a sudden jump in the scene's orientation at the point where the animators paused their recording session. To avoid this disorienting break in continuity, the system should save the orientation and rotation speed of the scene each time there is a pause.

- Speed: The rotation speed should be chosen carefully: It should be related to to the rate at which the object is changing. If a rapidly-rotating object is experiencing a lot of change with wach program event, the user may feel that there is not enough time to see all the changes.

- Goal-oriented: For algorithms such as 3D quick-hull, which act on an opaque object first on one side of it, then somewhere else, it may be impossible to predict in what order the actions will occur. Hence rotation at a fixed rate about a fixed axis may cause many events to occur on the side of the object which the user can't see. In these cases, it can be useful to turn the object as each action occurs, in order to keep the locus of activity facing the user's viewpoint.

Transparency: Another way to see more of a 3D object is to make it transparent or translucent (partially transparent). This allows the user to see inside the object, and it can useful if the algorithm deals with nested polyhedra. This approach must be used with care, otherwise there may simply be too much for the user to see. It is better to make selected portions of an object transparent, rather than forcing the user to understand the whole thing at once. Used in this way, transparency is a means to draw the viewer's attention to an important piece of the animation.

Navigation: Sometimes the user may need to immerse him/herself in a geometric scene and navigate through it. In such cases, one must make it easy to explore the environment, while keeping the user from getting lost in it. This is a problem faced not just in geometric algorithm animation, but also visualization in general and in virtual reality. Some facilities that can aid the user include:

- Local transparency: the user should be able to see through nearby objects, to see the next place he/she wants to go.

- Local level of detail: since distant objects are small, they need not be rendered in detail. In fact, performance may be improved by not rendering objects beyond a certain distance at all.

- Gravity: to prevent disorientation, it is best to keep the user's orientation relatively fixed. If the user looks up or down, there should be an option to return to the standard orientation.

Exploration of Objects: Very complicated three-dimensional objects may be best understood if the user can interact with them and look at their insides. An ideal interface will allow the user to look inside an object by cutting it open by slicing it along a selected plane or breaking it apart at "natural" boundaries and then separating the pieces to examine them separately.

7 Conclusion

We have tried to convey the state of geometric algorithm animation, to inform the prospective algorithm designer the options available, whether he/she is designing a new system or using an existing one to create animations.

As mentioned in the introduction, there are some algorithms that are more difficult to animate than others. It may be wise to warn the reader that, in fact, even well-crafted algorithm animations by themselves may not help some people understand an algorithm.

Stasko and others have tried to experimentally asses the effectiveness of algorithm animation for instruction. One report [SB93] describes an experiment where a group of computer-science graduate students were taught a new algorithm in two ways: one group read a research paper describing the algorithm, and the other divided its time between reading the paper and interacting with an animation of the algorithm. All the students were tested on their understanding of the algorithm. Although the group of students exposed to the animation performed better, the difference was very slight, and several of the subjects complained that they needed more explanations of what was happening. Moreover, some in the second group complained that they "knew" how the algorithm worked, but when pressed couldn't explain it. Other studies [Pa91] reported that students exposed to animations retained less understanding after a week's time had elapsed.

Some of the most effective animations discussed in section 4 are those that combine text and images. For example, *An Animation of a Fixed-Radius All-Nearest Neighbors Algorithm* [VR3c] uses both these modes of expression very effectively. In fact, the second part of this video has no text, and (not coincidentally) is harder to understand.

Perhaps the lesson to be learned here is that thinking in images involves a different part of human consciousness from understanding mathematical symbols. A necessary component when presenting abstract ideas remains the use of text. The ideal algorithm explanation may be some sort of interactive combination of text, sound, speech and animation. In other words, a hypertext multimedia system.

The Future

The development of HTML has made the world computer web accessible to people with little technical expertise. However, the amount of interactivity on the web is limited because users can only request prepared data, be it as images, animation or text. The interactivity needed for geometric algorithm animation, together with the need for hypertext explanation of the algorithms, may be supplied by carefully crafted animations in Java or some other web-based programming language.

The possibility of linking text and animations together presents challenges to animators, programmers and hardware designers. Animators need to conceive new ways of allowing the user to interact not just with the stream of moving pictures, but more importantly with the objects that the pictures portray. Software designers must also be involved in this task. On the hardware side, the demand to download or generate real-time video places large demands on network bandwidth and CPU processing power.

An ideal algorithm animation will allow the user to run a program, choose "interesting" data, change the data interactively, and ask about the program's objects and behavior. Existing animation systems present users with a choice of algorithms and data, but do not allow a great deal of interaction or questioning.

Real-time interaction requires a great deal of computing power. The user should be able to change the data with some interactive device, and sees the effects of the change "immediately". Of course, nothing is instantaneous, but the illusion of instant response can be provided if the algorithm is run on the changed data at video rates, or 60 times per second. If there is a lot of data, the algorithm will not be able to keep up with the user's changes. As computers improve and get faster, larger data sets and more interesting animations will be accessible to real-time interaction.

A user's question may lead to running a related algorithm for comparison, or a component of a larger algorithm. How will the newly-requested algorithm be made available? If it is explained by a prepared animation, large amounts of disk space and communication bandwidth will be required. On the other hand, generating it in real time will place heavy burdens on the CPU. This is especially true if the animation is coded in an interpreted language such as Java.

As both hardware and software improve, new animations will become possible. The future has always been difficult to predict, but we can say with confidence that geometric algorithm animation has good prospects.

References

[AR93] R.V. Amorin and P.J. Rezende, "Compreensão de Algoritmos através de Ambientes Dedicados a Anima cão" (in Portuguese, English title: *Algorithm Understanding through Dedicated Animation Environments*), Technical Report DCC-25/93, State University of Campinas, Brazil 1993.

[Ba81] R.M. Baecker, "Sorting out Sorting" (video), in *Siggraph Video Review 7*, 1981.

[BK91a] J.L. Bentley and B.W. Kernighan, "A System for Algorithm Animation", Computing Systems (Winter 1991) 4(1) pp. 5-31.

[BK91b] J.L. Bentley and B.W. Kernighan, "A System for Algorithm Animation: Tutorial and User Manual", Technical Report 132, AT&T Bell Labs, 1991.

[BS84] M.H. Brown and R. Sedgewick, "A System for Algorithm Animation", ACM Computer Graphics (July 1984) 18(3) pp. 177-186.

[BS85] M.H. Brown and R. Sedgewick, "Techniques for Algorithm Animation", IEEE Computer (January 1985) pp. 28-39.

[Br88a] M.H. Brown, "Exploring Algorithms Using Balsa-II", IEEE Computer (May 1998) pp. 14-36.

[Br88b] M.H. Brown, "Algorithm Animation", PhD. Thesis, Brown University, 1988.

[Br91a] M.H. Brown, "Zeus: A System for Algorithm Animation and Multi-View Editing", Proceedings, 1991 IEEE Workshop on Visual Languages pp 4-9.

[Br91n] M.H. Brown, "Color and Sound in Algorithm Animation", Proceedings, 1991 IEEE Workshop on Visual Languages pp 10-17.

[BN93] M.H. Brown and M.A. Najork, "Algorithm Animation Using 3D Interactive Graphics", Technical Report 110a, DEC Systems Research Center, 1993.

[BN94] M.H. Brown and M.A. Najork, "A Library for Visualizing Combinatorial Structures", Technical Report 128a, DEC Systems Research Center, 1994.

[Dg92] C.J. DiGiano, "Visualizing Program Behavior Using Non-speech Audio", M.S. Thesis, University of Toronto, 1992.

[Du86] R.A. Duisberg, "Animated Graphical Interfaces", ACM SIG CHI 86 Conference on Human Factors in Computing Systems, pp 131-136.

[EK94] P. Epstein, J. Kavanagh, A. Knight, J. May, T. Nguyen and J.-R. Sack, "A Workbench for Computational Geometry", Algorithmica (1994) 11 pp. 404-428.

[FK95] G. Fowler, D. Korn, E. Koutsofios, S. North and K-P. Vo, "Libraries and File System Architecture; Intertool Connections", in *Practical Reusable Unix Software* John Wiley & Sons, 1995.

[Ge95] `http://www.geom.umn.edu/software/download/geomview.html`

[HH89] E. Helttula, A. Hyrskykari and K.-J. Räihä, "Graphical Specification of Algorithm Animations with ALADINN", Hawaii International Conference on System Sciences (22nd, Volume 2) pp. 892-901.

[Ko91] K. Konstantinides, "Algorithm Visualization Using Tree Graphs", The Visual Computer (1991) 7:220-228

[LD85] R.L. London and R.A. Duisberg, "Animating Programs Using Smalltalk", IEEE Computer (August 1985) pp 61-71.

[My86] B.A. Myers, "Visual Programming, Programming by Example and Program Visualization: A Taxonomy", ACM SIG CHI 86, Proceedings, Conference on Human Factors in Computing Systems, pp. 59-66.

[NK94] S.C. North and E. Koutsofios, "Applications of Graph Visualization" Graphics Interface '94, pp 235-245.

[Pa91] S. Palmiteer and J. Elkerton, "An Evaluation of Animated Demonstrations for Learning Computer-based Tasks", Proceedings of the ACM SIGCHI '91 Conference on Human Factors in Computing Systems, pp. 257-263.

[PP92] M. Petre and B.A. Price, "Why Computer Interfaces are not Like Paintings: The user as a Deliberate Reader", Proceedings of East-West HCI 92: The St. Petersburg International Conference on Human-Computer Interaction, Vol I. pp. 217-224.

[Pr90] B.A. Price, "A Framework for the Automatic Animation of Concurrent Programs", M.S Thesis, University of Toronto, 1990.

[PS93] B.A. Price, I.S. Small and R.M. Baecker, "A Taxonomy of Software Visualization", Proceedings of the 25th Hawaii International Conference on System Sciences, Vol II, pp. 597-606.

[PB93] B.A. Price, R.M. Baecker and I.S. Small, "A Principled Taxonomy of Software Visualization", Journal of Visual Languages and Computing (1993), 4(3).

[RJ93] P.J. Rezende and W.R. Jacometti, “Geolab: An Environment for Development of Algorithms in Computational Geometry”, Technical Report DCC-26/93, State University of Campinas, Brazil 1993.

[Sc92] P. Schorn, “The XYZ GeoBench for the Experimental Evaluation of Geometric Algorithms”, in Computational Support for Discrete Mathematics: DIMACS Workshop (March 1992), pp. 137-152.

[St90a] J.T. Stasko, “Tango: A Framework and System for Algorithm Animation”, IEEE Computer (September 1990), pp. 27-39.

[St90b] J.T. Stasko, “The Path-transition Paradigm: a Practical Methodology for Adding Animation to Program Interfaces”, Journal of Visual Languages and Computing (1990) 1, pp. 213-236.

[St92] J.T. Stasko, “Three-Dimensional Computation Visualization”, Technical Report GIT-GVU-92-20, Georgia Institute of Technology, Atlanta, GA, 1992.

[St94] `http://www.cc.gatech.edu/stasko/cgi-bin/animation`

[SB93] J.T. Stasko, A. Badre and C. Lewis, “Do Algorithm Animations Assist Learning? An Empirical Study and Analysis”, INTERCHI ’93 Proceedings, Conference on Human Factors in Computing Systems (April 1993) pp 61-66.

[ST92] J.T. Stasko and C.R. Turner, “Tidy Animations of Tree Algorithms”, Technical Report GIT-GVU-92-11, Georgia Institute of Technology, Atlanta, GA, 1992.

[Ta95] A.Y. Tal, “Animation and Visualization of Geometric Algorithms”, Ph.D. thesis, Princeton University, 1995.

[TD95] A.Y. Tal and D.P. Dobkin, “Visualization of Geometric Algorithms”, IEEE Transactions on Visualization and Computer Graphics (TVCG) Volume 1, Number 2.

[Tu83] Tufte, Edward R., “The Visual display of quantitative information”, Cheshire, Conn, Graphics Press, c1983.

[Wh94] G. Whale, "DRUIDS: Tools for Understanding Data Structures and Algorithms", 1994 IEEE First International Conference on Multi-Media Engineering Education: Proceedings (1994) pp. 403-407.

Videos

[VR1] "Animation of Geometric Algorithms: A Video Review", Edited by Marc H. Brown and John Hershberger, Digital Equipment Corporation's Systems Research Center (DEC SRC), 1992.

This video tape contains the following animations:

[VR1a] Simon Kahan, "Real-Time Closest Pairs of Moving Points".

[VR1b] Peter Schorn, Adrian Brüngger and Michele De Lorenzi, "The XYZ GeoBench: Animation of Geometric Algorithms".

[VR1c] Herbert Edelsbrunner and Roman Waupotitsch, "Optimal Two-Dimensional Triangulations".

[VR1d] John Hershberger and Marc H. Brown, "Boolean Formulae for Simple Polygons".

[VR1e] Chandrajit L. Bajaj, "SHASTRA: A Distributed and Collaborative Design Environment".

[VR1f] Leonidas Palios and Mark Phillips, "Tetrahedral Break-Up".

[VR1g] Joseph Friedman, "Compliant Motion in a Simple Polygon".

[VR1h] P. Epstein, J. Kavanagh, A. Knight, J. May, T. Nguyen and J.-R. Sack, "Workbench for Computational Geometry".

[VR1i] Marc H. Brown and Harald Rosenberger, "Topologically Sweeping an Arrangement: A Parallel Implementation".

[VR1j] Ayellet Tal, Bernard Chazelle and David Dobkin, "The New Jersey Line-Segment-Saw Massacre".

[VR2] "The Second Annual Video Review of Computational Geometry", Edited by Marc H. Brown and John Hershberger, DEC SRC, 1993.

This video tape contains the following animations:

[VR2a] Seth Teller, "Visualizing Fortune's Sweepline Algorithm for Planar Voronoi Diagrams"

[VR2b] David Dobkin and Ayellet Tal, "Building and Using Polyhedral Hierarchies"

[VR2c] Stefan Schirra, "Moving a Disc between Polygons"

[VR2d] John Hershberger, "Compliant Motion in a Simple Polygon"

[VR2e] Estarose Wolfson and Micha Sharir "Implementation of a Motion Planning System in Three Dimensions"

[VR2f] P.J. de Rezende and W.R. Jacometti "Animation of Geometric Algorithms using GeoLab"

[VR2g] Michael Murphy and Steven S. Skiena "Ranger: A Tool for Nearest Neighbor Search in High Dimensions"

[VR2h] Jack Snoeyink "Objects That Cannot Be Taken Apart With Two Hands"

[VR3] "The Third Annual Video Review of Computational Geometry", Edited by Marc H. Brown and John Hershberger, DEC SRC, 1994.

This video tape contains the following animations:

[VR3a] John Hershberger and Jack Snoeyink, "An $O(n \log n)$ Implementation of the Douglas-Peucker Algorithm for Line Simplification"

[VR3b] David Dobkin and Dimitrios Gunopulos, "Computing the Rectangle Discrepancy'

[VR3c] Hans-Peter Lenhof and Michiel Smid, "An Animation of a Fixed-Radius All-Nearest Neighbors Algorithm"

[VR3d] Ayellet Tal and David P. Dobkin, "GASP–A System to Facilitate Animating Geometric Algorithms"

[VR3e] Adrian Mariano and Linus Upson, "Penumbral Shadows"

[VR3f] J.D. Cohen, M.C. Lin, D. Manocha, and M.K. Ponamgi, "Exact Collision Detection for Interactive Environments"

[VR3g] Hervé Brönnimann, "Almost Optimal Polyhedral Separators"

[VR3h] A. Varshney, F.P. Brooks, Jr., and W.V. Wright, "Interactive Visualization of Weighted Three-dimensional α-Hulls"

[VR4] "The Fourth Annual Video Review of Computational Geometry", Edited David Dobkin, Association for Computing Machinery (ACM) 1995.

This video tape contains the following animations:

[VR4a] Leo Loskowitz and Elisha Sacks, "HIPAIR: Interactive Mechanism Analysis and Design Using Configuration Space".

[VR4b] Bernard Geiger, "3-D Modeling Using the Delaunay Triangulation".

[VR4c] Mahav K. Ponamgi, Ming C. Lin and Dinesh Manocha, "Incremental Collision Detection for Polygonal Models".

[VR4d] B. Chazelle, D. Dobkin H. Shouraboura and A. Tal, "Convex Surface Decomposition".

[VR4e] Fredo Durand and Claude Puech, "The Visibility Complex Made Visibly Simple".

[VR4f] Steve Glassman and Greg Nelson, "An Animation of Euclid's Proposition 47: The Pythagoras Theorem".

Videos from the Geometry Center

[EG91] David Epstein, Charlie Gunn, *et al*, "Not-Knot", distributed by A K Peters, Wellesley MA, 1991.

[LM94] Silvio Levy, Delle Maxwell and Tamara Munzner. "Outside In", distributed by A K Peters, Wellesley MA, 1994.

[LM95] Stuart Levy, Tamara Munzner, Lori Thomson *et al*, "The Shape of Space". (To be released)