HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Laboratory of Software Technology

Ville Karavirta

Facilitating Algorithm Animation Creation and Adoption in Education

Licentiate's Thesis submitted in partial fulfillment of the requirements for the degree of Licentiate of Science in Technology.

Espoo, December 7, 2007

Supervisor:	Professor Lauri Malmi
Instructor:	Docent Ari Korhonen

HELSINKI UNIVERSITY
OF TECHNOLOGY
Department of Computer Science and Engineering

ABSTRACT OF
LICENTIATE THESIS

Author: Ville Karavirta

Date: December 7, 2007

Pages: 54 + 53

Title of the thesis:
Facilitating Algorithm Animation Creation and Adoption in Education

Professorship: Software Systems

Code: T-106

Supervisor: Professor Lauri Malmi

Instructor: Docent Ari Korhonen

Algorithm visualization aims to aid the human understanding of a high-level representation of a piece of code. Algorithm animation (AA) is a dynamic algorithm visualization where the dynamic behavior can range from a series of static pictures to an animation requiring interaction from the user.

Algorithm animation has been found to be educationally effective, provided that it is interactive enough. However, algorithm animation has not been widely used in teachiing computer science. One of the main reasons for not taking full advantage of AA in teaching is the lack of time on behalf of the instructors. Furthermore, there is a shortage of ready-made, good quality algorithm visualizations. Based on this knowledge, this research focuses on facilitating the creation and adoption of AA in education by finding answers to the following two questions.

First, we research how to lower the effort needed to produce algorithm visualizations for teaching. To achieve this, we first define a taxonomy to measure the effortlessness of AA systems. Then, we present a new effortless system, MatrixPro, that is intended for teachers to use on lectures.

The second research question is, how can we enable data exchange between algorithm animation systems. To answer this, we specify a taxonomy of algorithm animation languages based on a survey of existing AA languages. We then use the taxonomy to combine the required features of a language that can be used as an intermediate language when exchanging data between algorithm animation systems. Based on this synthesis and the work of the international algorithm animation research community, we define an eXtensible Algorithm Animation Language, XAAL. In addition, we introduce an implementation of a set of tools that enables us to exchange data between various AA systems. Finally, we evaluate the new language based on the taxonomy of the algorithm animation languages and the tool implementation.

Keywords: algorithm animation, effortlessness, algorithm animation language, XAAL

TEKNILLINEN KORKEAKOULU Tietotekniikan osasto	LISENSIAATINTUTKIMUKSEN TIIVISTELMÄ
Tekijä: Ville Karavirta	Päiväys: 7. joulukuuta 2007
	Sivumäärä: 54 + 53
Työn nimi: Algoritmianimaatioiden luomisen ja käyttöönoton helpottaminen	
Professuuri: Ohjelmistojärjestelmät	Professuurin koodi: T-106
Työn valvoja: Professori Lauri Malmi	
Työn ohjaaja: Dosentti Ari Korhonen	

Algoritmien havainnollistamisella pyritään helpottamaan ihmistä ymmärtämään korkean tason esitystä ohjelmakoodista. Algoritmianimaatio on puolestaan dynaamista algoritmien havainnollistamista. Dynaaminen luonne voi vaihdella kuvasarjasta käyttäjän vuorovaikutusta vaativaan animaation.

Algoritmianimaatioista on todistettu olevan apua oppimisessa mikäli ne ovat tarpeeksi vuorovaikutteisia. Tästä huolimatta algoritmianimaatio ei ole saavuttanut suurta suosiota opettajien keskuudessa. Pääsyy tähän on, että opettajilla ei ole tarpeeksi aikaa animaatioiden luomiseen. Lisäksi valmiista, korkealaatuisista animaatioista on pulaa. Tämän tiedon pohjalta tässä työssä keskitytään tarkastelemaan algoritmianimaation luomisen ja käyttöönoton helpottamista tarkastelemalla seuraavia kahta kysymystä.

Ensin työssä tutkitaan miten animaatioiden tekemisestä saataisiin vähemmän vaivalloista. Tähän kysymykseen etsitään ratkaisua määrittämällä tapa mitata animaatiojärjestelmien vaivattomuutta. Lisäksi esitellään järjestelmä, MatrixPro, joka on vaivaton luentotyökalu opettajille.

Toisena kysymyksenä tarkastellaan kuinka voitaisiin toteuttaa algoritmianimaatioiden siirto järjestelmästä toiseen ei ole ollut mahdollista. Tähän työkaluksi määrittelemme taksonomian algoritmianimaatiokielten arvioimiseen. Tätä taksonomiaa käytetään hyödyksi määriteltäessä ominaisuuksia, joita vaaditaan algoritmianimaatiojärjestelmien väliseen tiedonvaihtoon soveltuvalta kieleltä. Tältä pohjalta määrittelemme laajennettavan algoritmianimaatiokielen (Xaal eXtensible Algorithm Animation Language). Kielen määrittelyssä käytetään hyväksi myös kansainvälisen työryhmän visiota yhteisestä algoritmianimaatiokielestä. Lisäksi esittelemme toteutuksen joukosta työkaluja, joka mahdollistaa tiedonvaihdon eri algoritmianimaatiojärjestelmien välillä. Lopuksi arvioimme määritellyn kielen taksonomian ja toteutuksen perusteella.

Avainsanat: algoritmianimaatio, vaivattomuus, algoritmianimaatiokieli, XAAL

Acknowledgements

This thesis has been done for the Software Visualization Group in Laboratory of Software Technology. First and foremost I would like to thank my supervisor Professor Lauri Malmi and instructor Docent Ari Korhonen for providing the facilities to do this work. Their input and feedback to the work during this process has been highly valuable.

An essential part of this work is the Matrix framework and I would like to thank the people who have been involved in developing it: Tapio Auvinen, Erik Fallenius, Juha Helminen, Jan Lönnberg, Jussi Nikander, Otto Seppälä, Petri Ihantola, and all the former members.

I am also grateful to the participants of the ITiCSE XML Working Group, Guido Rößling, Thomas Naps, Peter Brusilovsky, John English, Duane Jarc, Chuck Leska, Myles McNally, Andrés Moreno, Rocky Ross, and Jaime Urquiza-Fuentes, for the discussions and ideas during the intensive five-day spell in Portugal. I would especially want to thank Tom and Guido for giving me the chance to be part of the group.

My greatest gratitude goes to Linda for her love and support. And especially for constantly reminding me that there is life besides the thesis.

Finally, I wish to thank my family for the tremendous support over the years.

Otaniemi, December 7, 2007

Ville Karavirta

List of publications and the contributions of the author

This thesis consists of an introduction and the following publications [P1] - [P5]

[P1] Ville Karavirta, Ari Korhonen, Lauri Malmi, and Kimmo Stålnacke. MatrixPro - A tool for on-the-fly demonstration of data structures and algorithms. In *Proceedings of the Third Program Visualization Workshop*, pages 26–33, The University of Warwick, UK, July 2004.

In this paper, an algorithm animation system called MatrixPro is introduced. The system supports on-the-fly creation of animations using visual algorithm simulation. The author implemented the system based on the Matrix algorithm simulation framework [36].

[P2] Thomas Naps, Guido Rößling, Peter Brusilovsky, John English, Duane Jarc, Ville Karavirta, Charles Leska, Myles McNally, Andrés Moreno, Rockford J. Ross, and Jaime Urquiza-Fuentes. Development of xml-based tools to support user interaction with algorithm visualization. *SIGCSE Bulletin*, 37(4):123–138, December 2005.

This paper discusses requirements for a common algorithm animation language to be used by multiple AA systems. It gives examples and specifications for the different elements of AA. The author's main contributions were in the specification of the graphical primitives and transformations on them.

[P3] Petri Ihantola, Ville Karavirta, Ari Korhonen, and Jussi Nikander. Taxonomy of effortless creation of algorithm visualizations. In *ICER'05: Proceedings of the 2005 international workshop on Computing education research*, pages 123–133, New York, NY, USA, 2005. ACM Press.

This paper introduces a taxonomy of effortless creation of algorithm visualizations and evaluates some of the existing AV systems. All the authors of the paper contributed evenly on all parts of the paper.

[P4] Ville Karavirta, Ari Korhonen, and Lauri Malmi. Taxonomy of algorithm animation languages. In *SoftVis '06: Proceedings of the 2006 ACM symposium on Software visualization*, pages 77–85, New York, NY, USA, September 2006. ACM Press.

This paper introduces a taxonomy of algorithm animation languages. In addition, there is an evaluation of several existing AA languages. This work is based on the work by the author in [27].

[P5] Ville Karavirta. Integrating algorithm animation systems. In *Proceedings of the Fourth Program Visualization Workshop (PVW 2006)*, volume 178 of *Electronic Notes in Theoretical Computer Science*, pages 79–87, 4 July 2007.

This paper describes a new algorithm animation language called Extensible Algorithm Animation Language, XAAL. In addition, the paper shows how XAAL can be used to transfer algorithm animations between AA systems. The author of this thesis is the sole author of this work.

Contents

Chapter 1

Introduction

Due to the rapidly increased performance of computerized devices, software products have grown to be more and more complex. As a result, software developers need to understand very large parts of the software. To help them achieve this, a lot of research in the field of *Software Visualization* (SV) has been carried out. Software Visualization can be defined as *"the visualization of artifacts related to software and its development process"* [17].

Software Visualization can be applied to and benefited from on many different areas. Software developers can get insights on the class or package structures of an object-oriented software or detailed information of the state of the program through visual debuggers. In addition, they can test their software using *visual testing* [39]. On the other hand, algorithm developers and researchers can get a better view of the algorithms through visualizations. In education, students can use visualizations to help them understand and learn new concepts in software development and algorithmics.

In general, SV can be divided into visualizing the *structure*, *behaviour*, and *evolution* of software. Structure is the visualization of static parts and relations of the system. Behaviour is the visualization of the program execution with real or abstract data. Finally, evolution is the visualization of the development process of the software. [17]

Algorithm animation (AA) is one form of visualization of behaviour where the goal is to visualize the execution of an algorithm [17]. The main purpose of algorithm animation development is aimed toward use in educational context. This is also the focus in this thesis, although the ideas can be applied to different areas of SV as well.

Algorithm animation has been used in education for a few decades with the goal of helping students to learn the difficult concepts of data structures and algorithms. Recent studies have shown that to be *educationally effective* (*i.e.* aid students' learning) algorithm visualizations cannot be merely passive animations, the users must interact with the animation [23, 45]. Due to this discovery, different tools supporting various types of interaction have been developed. This user interaction can be, for example, *responding* to multiple-choice questions during the animation or *constructing* an algorithm animation using a visualization system.

1.1 Previous Work

In the Laboratory of Software Technology, the research in the field of SV started with the problem of manually grading exercises on the Data structures and algorithms course. With around 400 students participating on the course annually, an idea of an automatic assessment system was raised in early 1990's. The first system developed was TRAKLA [24] that automatically assessed students' solutions to algorithmic exercises. The answers had to be submitted to the system in a predefined textual format by email.

The first step in the field of SV was taken in mid 1990's when a graphical front-end for TRAKLA called TRAKLA-EDIT [24, 35] was developed. TRAKLA-EDIT allowed students to graphically solve the exercises and the system generated and sent the solutions by email. Although a major step forward, this system still had several limitations. First, there was no mapping between a data structure and the corresponding visualization. Thus, TRAKLA-EDIT was like a drawing tool for data structures and algorithms. Second, creating new exercises for the system was time consuming. To solve these problems, the development of a new application framework, eventually named Matrix [36], was initiated in 1999.

Matrix introduced a new concept, *visual algorithm simulation* [34], that allows direct manipulation of underlying data structures through a graphical representation. Matrix is a Java-based framework for developing applications that use visual algorithm simulation and visualization. Based on Matrix, a follower for TRAKLA-EDIT named TRAKLA2 was introduced in 2003 [35, 40]. TRAKLA2 is used to provide *visual algorithm simulation exercises*, where the student simulates the workings of actual algorithms. These exercises are automatically assessed and students get the feedback immediately.

1.2 Motivation and Research Problem

As mentioned, algorithm animation has been found to be educationally effective, provided that it is interactive enough [23]. Thus, a lot of research has been done on how to produce effective animations for the students. On the other hand, one of the main reasons for not taking full advantage of algorithm animation in teaching is the lack of time on behalf of the instructors [45]. Furthermore, there is a shortage of ready-made, good quality algorithm visualizations usable in teaching [56]. Based on this knowledge, this research focuses on the instructor's point of view and aims at finding answers to the following question.

1. *How can we develop algorithm animation systems to lower the effort needed to produce algorithm visualizations for teaching?*

To answer this question, we will explore what makes an algorithm animation system effortless. Based on this we will introduce a new AA system that is effortless to use in it's application area.

Currently, there are several systems (for example [2, 15, 22, 41, 46, 52, 59]) available for algorithm animation creation and usage in teaching. They provide different approaches to creating animations. In addition, for students using these animations, the systems provide different interaction methods. The problem is, however, that each of these systems has its own internal format for storing the animations and no data exchange between the systems is possible. Thus, when a teacher needs, for example, some specific type of activity, he or she has to create a new animation with a system supporting the required activity. For example, currently an animation used for passive viewing can not be used when wanting the students to respond to questions asked by the animation. Furthermore, a survey by Bassil and Keller [7] concluded that integrations of SV tools and importing/exporting visualizations from SV tools are the main challenges for the future of SV tool builders. This leads us to the second research problem:

2. *How can we enable data exchange between the existing algorithm animation systems?*

We will tackle this question by thoroughly analyzing the existing algorithm animation systems and thus identifying the key features of algorithm animation languages. This information is then used to implement the data exchange.

1.3 Contributions

The following points summarize the main contributions of this work.

- We define a Taxonomy of Effortless Creation of Algorithm Animations and introduce a system, MatrixPro, which allows effortless on-the-fly creation of algorithm animations by applying visual algorithm simulation and a simple user interface.
- We define a Taxonomy of Algorithm Animation Languages to help comparing the different AA languages.
- The taxonomy helps us in defining a new algorithm animation language, eXtensible Algorithm Animation Language, XAAL. In addition, we implement a parser for the language.
- We implement a set of tools for exchanging data between algorithm animation systems. The data exchange allows transferring data from an effortless, course-specific system (MatrixPro) to a general system (ANIMAL).

1.4 Structure of this Thesis

This thesis is structured as follows. Chapter 2 introduces the concepts discussed in this thesis in more detail. Chapter 3 begins with a short introduction to the history of algorithm animation and continues to discuss the characteristics of AA systems. Chapter 4 discusses the effortless creation of algorithm visualization. In Chapter 5, we introduce typical features of algorithm animation languages as well as present a taxonomy to evaluate the languages. Chapter 6 in turn describes our solution to implementing data exchange between AA systems. Finally, Chapter 7 discusses the usefulness of this work as well as looks into the future.

Chapter 2

Definition of Concepts

This chapter briefly defines the concepts used in the rest of this theses. We start by defining the field of Software Visualization and Algorithm Animation. Furthermore, we discuss the characteristics of SV systems as well as different roles in the SV production process.

2.1 Software Visualization and Algorithm Animation

Software Visualization can be defined as *"the visualization of artifacts related to software and its development process" [17].* As mentioned earlier, SV can be divided in visualizing the *structure*, *behaviour*, and *evolution* of software [17]:

- Structure is the visualization of static parts and relations of the system. The information visualized is available by statically analyzing the source code without executing it. Examples of structure visualization are pretty printing, control flow graphs, and UML class diagrams, just to mention a few.
- Behaviour is the visualization of the program execution with real or abstract data. Topics of behaviour visualization are dynamic architecture visualization, algorithm animation, visual debugging, and visual testing. Of these, algorithm animation is of special interest in this theses. In algorithm animation, the goal is to visualize the behaviour of an algorithm on a higher level of abstraction than source code.
- Evolution is the visualization of the development process of the software. Evolution visualization can be, for example, visualizing software metrics changes, visualizing structural changes, or visualizing software archives such as CVS.

2.1.1 Characteristics of Software Visualization Systems

It is difficult to choose a proper tool for software visualization from the vast amount of different SV tools supporting different features, target scope, and interaction techniques. The best suitable tool depends heavily on the type of the task. To help this process, taxonomies categorizing SV tools have been defined [42, 47, 48, 62]. One of the most well-known ways to categorize and evaluate Software Visualization systems is the *Taxonomy of Software Visualization* by Price et al. [47]. The taxonomy defines a structure of characteristics of SV systems that consists of six categories. These categories and the questions they should answer are the following.

- Scope — *"What is the range of programs that the SV system may take as input for visualization?"*
- Content — *"What subset of information about the software is visualized by the SV system?"*
- Form — *"What are the characteristics of the output of the system (the visualization)?"*
- Method — *"How is the visualization specified?"*
- Interaction — *"How does the user of the SV system interact with and control it?"*
- Effectiveness — *"How well does the system communicate information to the user?"*

These categories and their meaning in the SV production process are illustrated in Figure 2.1. They will be more thoroughly explained in the next chapter.

2.1.2 Roles in Software Visualization

The four different *roles* of persons who take advantage of software visualization are also shown in Figure 2.1. *Programmer* is a person who develops the algorithm or program without considering whether or not it is (going to be) visualized. *SV software developer* is a person who designs and implements software for SV. A person creating the visualization is called *visualizer*. Finally, the person using the visualization is addressed as *user*. In practice, these roles are often overlapping and it is common that, for example, the SV software developer is also a visualizer and a programmer. [47]

In this thesis, the main focus is on the educational use of SV. Thus, the persons involved are student and instructor. When considering the roles in SV, the usual

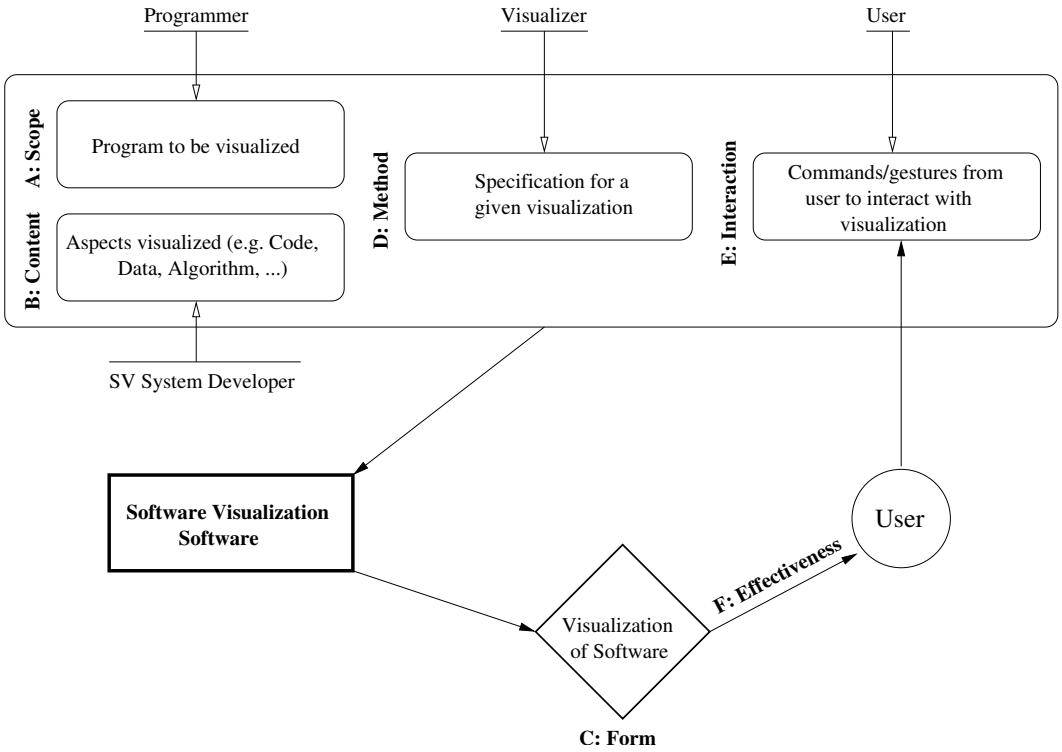


Figure 2.1: SV production process [47].

case is that student is the user and instructor has the rest of the roles. However, when considering, for example, a situation where the students are required to create their own visualizations, the student is in the role of visualizer. In this thesis, we will use the terms student and instructor and indicate which of the SV roles we are discussing unless it is clear from the context.

Until recently, the instructor has often been in the role of the developer. Usually, this is a task that requires a lot of effort and to gain wider audience, SV systems need to allow the instructor to be able to work only in the roles of programmer and visualizer.

When considering the student using the visualization, research has shown that passively viewing algorithm animations does not have a significant effect on learning outcomes [23]. Therefore, *engagement* (activity) by the student is needed for a tool to be pedagogically useful. The different *levels* of engagement according to the engagement taxonomy [45] are *viewing*, *responding*, *changing*, *constructing*, and *presenting*. Viewing is passive watching of an animation where student only controls the visualization's execution. In responding, the student is engaged by asking questions about the visualization. Changing requires the student to modify the visualization,

for example, by changing the input data. In constructing, the student is required to construct his/her own algorithm animation. At the highest level, presenting, the student presents a visualization for an audience.

2.1.3 Algorithm Animation Language

Throughout this thesis we will talk about *algorithm animation language* (or simply language). With this term we mean a textual representation describing an algorithm animation or visualization. The language should have a well-defined set of concepts, syntax, and semantics defined in the *language specification*. An *algorithm animation system* is a tool capable of interpreting a script written in an algorithm animation language and animating/visualizing it.

2.1.4 Other Definitions of Software Visualization

Price et al. [47] have defined software visualization as *"the use of the crafts of typography, graphic design, animation, and cinematography with modern human-computer interaction technology to facilitate both the human understanding and effective use of computer software."* They divide SV into two separate fields: *algorithm visualization* and *program visualization*. Program visualization is the use of visualization to enhance the human understanding of computer programs. Algorithm visualization (AV) is the visualization of a high-level representation of a piece of code. AV can be further divided into *static algorithm visualization* and *algorithm animation* (AA). Algorithm animation is a dynamic algorithm visualization. The dynamic behavior can range from a series of static pictures to an animation requiring interaction from the user. The problem with this definition of SV is that the line between algorithm visualization and program visualization has become fuzzy.

Chapter 3

Algorithm Animation

In this chapter, we discuss algorithm animation in more detail. We start with a short introduction to the history of algorithm animation. Next, we introduce the characteristics of software visualization and algorithm animation systems. Finally, we briefly describe several algorithm animation systems.

3.1 History of Algorithm Animation

The research on algorithm animation is often considered to have begun from the Sorting out Sorting video [5] by Ronald M. Baecker in 1981. It was a 30 minutes long video animating the behavior of nine different sorting algorithms. However, the first algorithm animations we are aware of were created in 1966 by Ken Knowlton, who made a movie about list processing using the L^6 programming language [33]. More of the early work was done by Baecker who in 1975 presented two systems that made it *"possible for an instructor to produce short quick-and-dirty single-concept film clips with only hours of effort"* [4].

The field has evolved a lot since the first videos and systems were introduced. The first well-known computerized system was BALSA (Brown ALgorithm Simulator and Animator) [13]. BALSA is an interactive algorithm animation framework that has a support for multiple dynamic views of an algorithm and the data structures associated to it. Another recognized system of the early years of algorithm animation is TANGO (Transition-based ANimation GeneratiOn) [57]. It is an AA system that introduced the *path-transition paradigm* and supported *smooth animation*. Since then, numerous algorithm animation systems have been developed (see, *e.g.*, [2, 9, 15, 22, 28, 38, 41, 46, 52, 59]). Figure 3.1 shows a timeline of the vari-

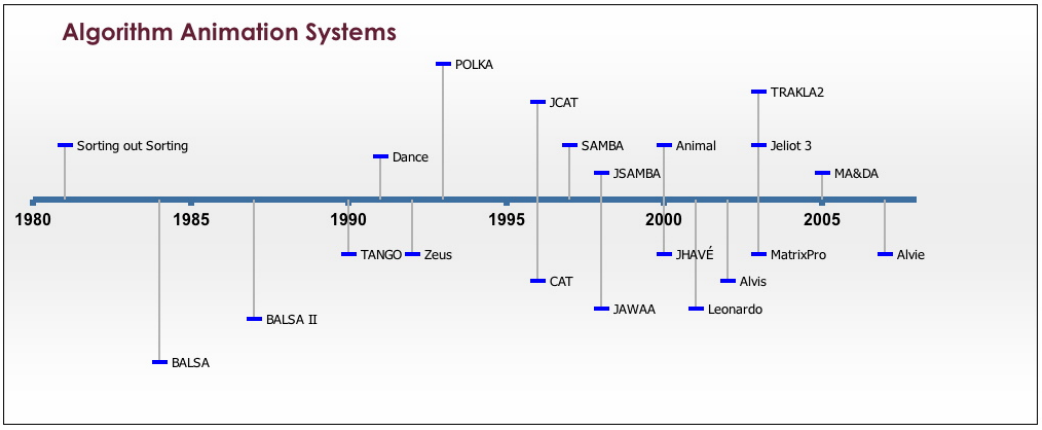


Figure 3.1: History of some Algorithm Animation Systems. The vertical positioning is merely a matter of improving readability.

ous AA systems. Plenty more systems exist, but the contributions of the selected systems will be briefly mentioned in the next section.

Currently, the main focus of the research is on engaging the student's with the tools [23, 45] and aiding the adoption of AA in teaching [25, 45, 55]. The current tools are thought to require too much time and effort to be useful [45], although compared to the Sorting out Sorting video that took three years to make [6], they are quite effortless to use.

3.2 Characteristics of Algorithm Animation Tools

To introduce algorithm animation and the research done on the field, this section will consider the characteristics and functionality of current algorithm animation tools. This will be done by examining the categories of a widely-accepted Software Visualization taxonomy [47]. We will not faithfully follow the taxonomy since it is already somewhat outdated and was intended to be extended and updated. However, the main categories are the same but with some parts left out or updated based on their relevance and research done after the taxonomy was published.

3.2.1 Category Scope

The category Scope is defined in the taxonomy to describe the range of programs that the system can visualize. It has two subcategories: *Generality* and *Scalability*. Scalability is used to measure how well the system scales up to visualize large examples. Generality describes how general a range of programs the system can

display. Generality also includes subcategories such as operating system and hardware. These subcategories are out of date, since today a tool must be *easy to obtain and install* and *platform independent* [49]. If these criteria are not met, it is highly unlikely that the tool will gain wide audience.

3.2.2 Category Content

The category Content describes how much information about the software is visualized by the system. Most of the subcategories are related to program visualization that is not considered here. The interesting subcategory is *Algorithm*. This subcategory describes the degree that the system visualizes the high-level algorithms. This is divided in two: *instructions* and *data*.

Instructions describes the degree of support for visualizing code instructions in the algorithm. This is often called *code visualization*. It is a mapping between the changes in the code and the visualization of the data. Code visualization can be done in various ways, for example, by highlighting single code lines or showing several codes of the same algorithm on different abstraction levels [14]. In addition to highlighting the current line, the code visualization can show things such as executed lines of codes (distinguished from the ones not executed) and the lines executed just before the current line [37].

The subcategory Data describes the capability of the system to visualize high-level data structures used by the algorithm. In algorithm animation, the lower extreme here is a system that uses only graphical primitives to describe the data structures. An example of such a system is SAMBA [61]. The other extreme is a system that visualizes only high level data structures, like, for example, MatrixPro [28]. These different approaches have both benefits and drawbacks. By using graphical primitives, the system can visualize almost any kind of structures, but the creation of such animations can require quite a lot of effort. On the other hand, systems using data structures can provide an effortless way to create the animation, but are typically limited to the set of structures supported by the system.

3.2.3 Category Form

The category Form describes the characteristics of the visualization created with the system. The subcategories are *Medium*, *Presentation Style*, *Granularity*, *Multiple Views*, and *Program Synchronization*.

Medium describes the target medium of the visualization system. This can be, for example, screen or paper. Currently, the focus is on *web deployment* [31]. This

adds requirements for AA tools that must either support suitable export-formats or be implemented as Java applets. There is a wide range of web-based systems available today, for example, JAWAA [2], Jeliot [20], JHAVÉ [46], JCAT [43], and JSAMBA [59].

The Presentation Style category describes the general appearance of the visualization. Algorithm visualization and animation can apply several different styles. *2D graphics* is the most used style to visualize algorithms. An option is *3D graphics* that can be used to express additional information about data, add another view to the same data, or visualize the history of a two-dimensional view [12]. For example, 3D graphics can be used in JCAT [43], and POLKA [58] has also been extended to support 3D animations. JCAT is a Java-based system for developing and viewing web-based collaborative active textbooks on algorithms. POLKA is an algorithm animation system that was designed to animate parallel programs.

A subcategory of Presentation Style is *Sound*. *Auralization* is a technique that uses sounds and melodies to describe the execution of an algorithm or a program. This can be done, for example, by mapping program data and events to certain sounds. There are several reasons to use auralization: humans can process aural information without actively listening, we have the ability to detect patterns in sound and to detect omissions in an aural pattern, sound and music can naturally portray parallel events, and the different sense used makes it possible to focus on two "views" at the same time [18]. Voice is also a good way to keep the user interested in the animation [19]. Although sound was used already in the Sorting out Sorting video [6], one of the pioneers using sound was Zeus [11]. Zeus used sound for reinforcing visual display, conveying patterns, replacing visuals, and signaling exceptional conditions. Another example of a system using auralization is CAITLIN [64]. CAITLIN allows the specification of an auralization for Pascal code constructs. It has been used to aid novice programmers debug their Turbo Pascal programs using auralization.

Another part of the Presentation style is the use of animation. In the context of algorithm animation, this will naturally be supported by the systems. However, there are two different techniques to support animation: *smooth animation* and *discrete changes*. In smooth animation, the changes in an animation step are shown as smooth transitions. With discrete changes, the changes are made without transitions and the changed parts are often highlighted. It has been argued that smooth animation is suitable for small data sets, whereas discrete changes are suited for large sets of data [10]. In addition to using smooth animation to *highlight changes*, several other techniques can be used to notify the user. For example, the shape or color of the graphical object that changed can be modified [10].

The category Granularity measures how well the system supports different levels of granularity. That is, does the system show the fine-grained details as well as allow the filtering of small details to view the bigger picture. This is important, since various users will have different needs, thus the tool must support different *levels of abstraction* [63].

The category Multiple Views measures the degree of multiple view support. Multiple Views eases the visualization of complicated algorithms or allows showing multiple, synchronized aspects of a simple algorithm, and is thus an important feature of AV systems [14].

The last category, Program Synchronization, measures the support for synchronized visualization of several different algorithms. This can help the comparison of different algorithms. For example, the Sorting out Sorting video shows several algorithms sorting the same data, thus making it easy to see how efficient they are.

3.2.4 Category Method

The category Method describes how the visualization is specified. It has two subcategories: *Visualization Specification Style* and *Connection Technique*.

Connection Technique describes how the program source code and the visualization are connected. This is, however, not relevant to our task here since we are not dealing with program visualization.

Visualization Specification Style describes the way visualizations are specified. In the original taxonomy this was measured using terms like *hand-coded*, *library*, and *automatic*. However, since the taxonomy was introduced, many different visualization specification styles have emerged. Thus the list above is out-dated and we will introduce an alternative categorization in the following. The list is loosely based on [50]. It should be noted, that many of the current systems include several of the techniques.

- *Topic-Specific Animation* - Topic specific animations are, as the name suggests, built specifically for some topic. Usually these are stand-alone animations instead of algorithm animation systems. For example, the software packages by Khuri and Hsu concentrate on image compression algorithms [32].
- *Direct Manipulation* - In direct manipulation [60], the animation is specified by manipulating graphical objects. *Visual algorithm simulation* [34] takes direct manipulation one step further by allowing the animation to be specified manipulating concrete data structures through visualizations. Examples of AA systems using direct manipulation are Dance [60] and MatrixPro [28].

- *API-based Generation* - In API-based generation, the animations are generated through method invocations of a visualization system's application programmer's interface (API). An example of this approach is JHAVÈ and the API to generate GaigsXML [44].
- *Scripting-based Generation* - In scripting-based generation, the animations are described using some intermediate format, usually a textual format. Examples are ANIMAL [51], and JAWAA [2].
- *Declarative Visualization* - Declarative visualization specifies the visualization by declaring mappings between a program state and a graphical representation. This is done by using mathematical expressions. An example of this approach is the ALPHA language [16].
- *Code Interpretation* - Code interpretation is also a popular style due to its effortlessness. An example of such system is Jeliot 3 [41] that automatically visualizes small Java programs.

This topic of visualization specification styles is relevant for the implementation strategies of the data exchange. In discussion in Chapter 7, we will consider how these different approaches fit to the idea of data exchange between systems.

3.2.5 Category Interaction

This category contains metrics to evaluate the way the user interacts with an SV system. The subcategories are *Style*, *Navigation*, and *Scripting Facilities*.

The category Style describes the method used to give instructions to the system. Here we will extend this category and include a new taxonomy to describe the style (or level) of interaction (or engagement) supported by the system. The different *levels* of engagement according to the engagement taxonomy [45] are the following.

- *Viewing* is the core level of engagement. It is passive viewing of an animation. However, the student can have controls to move backward/forward in the visualization, change the speed, etc. It should be noted, that viewing is included on all of the higher levels of engagement.
- In *responding*, the student is engaged by asking questions about the visualization. The question can be, for example, "What will happen in the next step of the algorithm?". The main idea is that students use the visualization to find the answer for the questions.

- *Changing* requires the student to modify the visualization. This can be, for example, changing the input data of the algorithm allowing the student to explore the algorithm's behavior in different situations.
- In *constructing*, the student is required to constructs his/her own algorithm animation. This can be done, for example, in terms of direct manipulation in some algorithm animation system. It should be noted, that coding of the algorithm is not a requirement on this level.
- At the highest level, *presenting*, the student presents a visualization for an audience. This can be, for example, a situation where a student presents a visualization for the instructor and peers. The visualization can be made by the student or a third-party.

The Navigation subcategory is interesting and requires good *control over the animation* for the tool to be evaluated good in this category. This is important in AA in education because when using the tool for viewing an animation, the user will need to compare steps in the animation to understand the algorithm. Good controls include the capability to move backwards and forwards, play and pause, and change the speed of the animation [3, 52].

The Scripting category answers the question whether the tool supports some *s*cripting language to define the animations. This is naturally a feature highly related to our topic. Thus, we will survey some systems supporting this feature in Section 3.3 and their (scripting) languages in Chapter 5.

3.2.6 Category Effectiveness

This category is related to measuring the effectiveness of the created animations. From the educational point of view, this is an extremely important category. However, this is not related to the production of an algorithm animation or to the specification of an algorithm animation language and is therefore not examined here.

3.3 Algorithm Animation Systems

In this section, we will briefly introduce some of the numerous algorithm animation systems developed over the years. We will describe systems that include an algorithm animation language that can be used to specify the animations because our target is to define a new algorithm animation language. These descriptions are

merely an introduction to the different systems, more detailed descriptions can be found from Appendix A or from the cited papers.

ALVIS ALgorithm VIsualization Storyboarder (ALVIS) [22] is an interactive algorithm visualization system designed to create and view *low-fidelity* algorithm visualizations. In low-fidelity visualizations, the algorithm is illustrated with few inputs using a sketched, unpolished appearance.

Animal ANIMAL [52] is a general-purpose animation tool with a current focus on algorithm animation. It has a scripting language called ANIMALSCRIPT [51]

DsCats Data Structure Computer Animation Tools (DsCats) [15] is an application focused on learning the data structures and algorithms. It supports tree structures such as B-Tree and binary search tree.

JHAVÉ JHAVÉ [46] is an algorithm visualization environment that is intended as a platform for algorithm visualization systems. The current version of JHAVÉ has an XML language called GaigsXML [44].

JAWAA JAWAA 2.0 [2] is a data structure and algorithm animation system that consists of three parts: a scripting language, a graphical editor, and an applet capable of showing animations defined in the scripting language.

JSAMBA JSAMBA [59] is a front-end for the POLKA [58] algorithm animation system. Basically, it is a viewer to visualize animations written in the Samba scripting language [61].

MatrixPro MatrixPro (see Publication [P1] for details) is a system to create algorithm animations using visual algorithm simulation.

Chapter 4

Effortless Creation of Algorithm Animations

The effort and time needed to create algorithm visualizations is one of the main reasons for educators not adopting AV in their teaching. Thus, in this chapter, we will first consider the effortless creation of algorithm animations. Furthermore, we briefly describe our implementation of an effortless system, MatrixPro.

4.1 Taxonomy of Effortless Creation of Algorithm Visualizations

In the past, we have identified that there are either specific, low effort systems or general, high effort systems [29]. That research was, however, our subjective view of the subject. The next step was a survey targeting computer science educators [30]. The survey resulted in an initial set of measures for effortlessness. Based on that data, we introduced a Taxonomy of Effortless Creation of Algorithm Visualizations (see Publication [P3]). The main categories of the taxonomy are briefly introduced in the following, for more detailed discussion, see Publication [P3].

Category Scope This category measures how wide the application area of the visualization system is. The taxonomy defines four levels: *lesson-specific*, *course-specific*, *domain-specific*, and *non-specific* with non-specific systems having the widest scope. For example, a lesson-specific system can only be used on one lecture whereas course-specific can be used on most lectures on a course.

Category Integrability This category measures the features that make the system easy to integrate into an education setup. This includes features such as

ease of installation, documentation, course management support, and integration into hypertext.

Category Interaction This category measures the interaction provided by the system. It distinguishes two types of interaction: *producer-system interaction* and *visualization-consumer interaction*. Producer-system interaction measures the level of preparation needed for different tasks such as lecture examples or creating an exercise for examination. Visualization-consumer interaction measures the level of interaction (or engagement) provided for the user of the visualization.

Publication [P3] includes evaluations of four systems (ANIMAL [52], JAWAA [2], Jeliot 3 [41], and MatrixPro) using this taxonomy. The main finding in the evaluation is that there are no generic systems that can be used without prior preparation (see Figure 4.1). Thus, the final question in the article is, can such a system be developed? In the next section, we will introduce a system, MatrixPro, that is course-specific and that can be used on-the-fly.

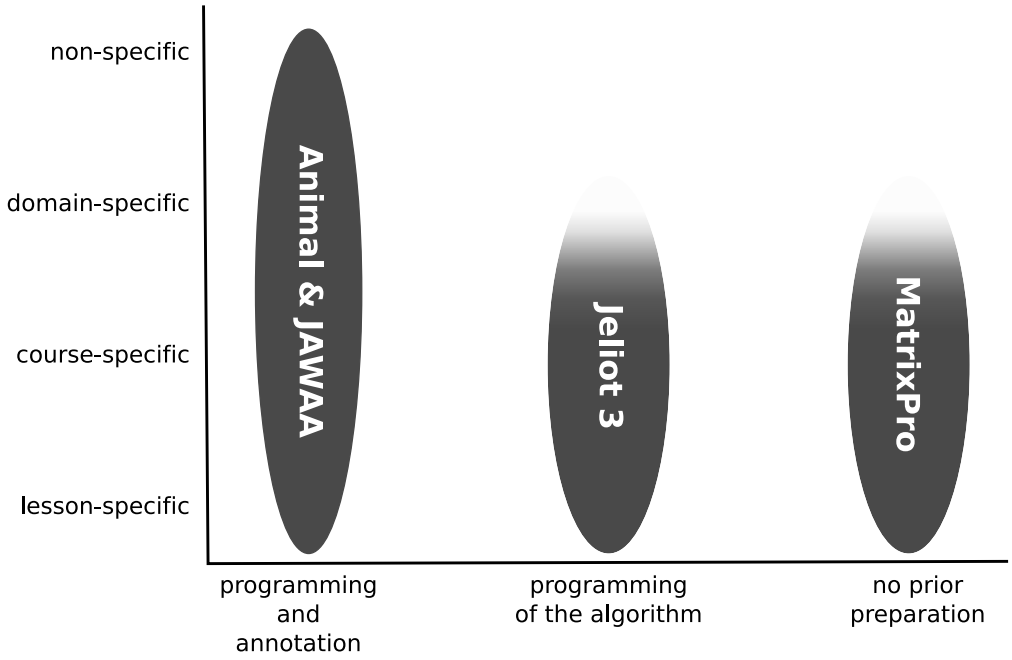


Figure 4.1: Evaluation of effortlessness of four AA systems. A single system might support several levels of on-the-fly use, but only the most typical level is marked.

4.2 MatrixPro

Publication [P1] introduces a new algorithm animation system called MatrixPro (see Figure 4.2) that allows on-the-fly creation of algorithm animations. It is based on the Matrix algorithm simulation framework [36]. The following will briefly summarize the main features of the system. A more detailed description can be found in Publication [P1].

In MatrixPro, the animations are created using *visual algorithm simulation*. In this approach, the user manipulates visualizations of the underlying structure and creates a sequence of simulation steps. These steps include basic variable assignments, reference manipulation, and operation invocations such as insertions and deletions. All the operations are done using direct manipulation, *i.e.* by drag and dropping.

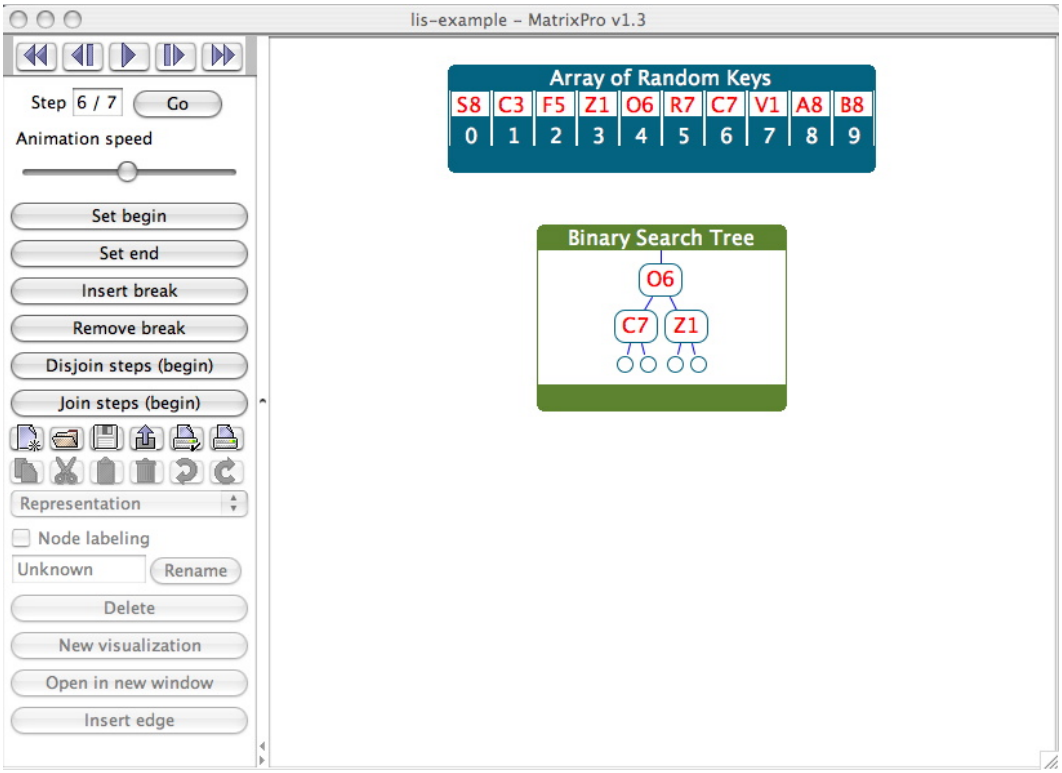


Figure 4.2: MatrixPro main window.

The main window of MatrixPro is shown in Figure 4.2. The main functionality of the system is in the toolbar on the left and the menubar (not shown in the figure). The toolbar is an essential component which enables users to modify the created animations. Through the toolbar the user can modify the animation easily, for

example, by changing the granularity of the animation sequence. In addition, the toolbar (as well as the menubar) contains controls for moving backward and forward in the animation.

The area of visualizations contains the visualizations of the data structures that the user can interact with in terms of visual algorithm simulation. The simulation consists of *drag and drop operations* which can be carried out by picking up the *source* and moving it onto the *target*. Each single operation performs the *proper* action for the corresponding underlying data structure. An action is proper if the underlying data structure object accepts the change (*e.g.*, change of a key value in a node or change of a reference target).

The main features of the system are the following.

On-the-fly usage The most important feature of MatrixPro is the ability to use the system on-the-fly. This is achieved by combining the visual algorithm simulation and a library of ready-made data structures that can be animated. For example, insertion to a B-tree can be demonstrated by simply drag and dropping keys on the B-tree visualization.

Customized animations The system supports customization of animations in two ways. The instructor can use whatever input data he/she wants. In addition, the granularity level of the animation can be changed, *i.e.*, how large steps are shown when playing the animation.

Storing and Retrieving Animations Although the system supports on-the-fly usage, some instructors will want to prepare their animations in advance. For this purpose, MatrixPro supports storing and retrieving of the created animations. The animation can be stored as serialized Java objects or exported as Scalable Vector Graphics (SVG) [65]. In addition, single steps in the animation can be exported as PNG or TEXdraw [26].

Customizable user-interface The user interface of MatrixPro can be easily customized by changing the set of toolbar objects. This allows it to fit the needs of various users. For example, when demonstrating ready-made animations on lecture, the instructor probably needs only the animation controls.

Library MatrixPro includes a library of data structures that can be used to produce animations making the production process less error-prone.

Chapter 5

Algorithm Animation Languages

In this chapter, we will introduce the main features of the algorithm animation languages used in the systems listed in Section 3.3. Furthermore, we will summarize a Taxonomy of Algorithm Animation Languages introduced in Publication [P4].

While reading this chapter, the reader should keep in mind that we deal with the languages, not the systems. Some of the features considered might be available in a system, but not through the language the system uses.

5.1 Features of Algorithm Animation Languages

The following subsections show the main characteristics of the algorithm animation languages. As stated earlier, we see algorithm animation language as a textual representation describing an algorithm animation or visualization and it should have a well-defined set of concepts, syntax, and semantics. The distinction between algorithm animation languages and other languages is slightly fuzzy. The main principle is that a language has to have something specifically designed for animating algorithms to be considered an algorithm animation language.

The reader should note that this introduction will not state every feature of the languages, only the ones that are most common or distinctive. Also, the examples shown of the languages are often not complete with all the details required in the language and they most likely cannot be used as-is in any system. For descriptions of the languages themselves, see Publication [P4]or the cited articles.

5.1.1 Representation Format

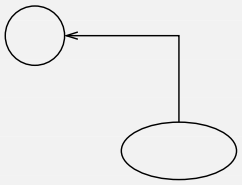
The first noticeable feature is the format of the language. All the languages we are discussing have a textual format. Listing 5.1 gives a simple example of AnimalScript [51], the scripting language of Animal.

```
1 circle "C" (150, 100) radius 30 color black filled
      fillColor red depth 3
2 move "C" along line (130, 80) (130, 170) within 200 ms
```

Listing 5.1: Example of graphical primitives and basic animation in AnimalScript.

In the recent languages, XML as a format has become more and more popular because it makes it easy for software to process data using the multitude of different tools available. Listing 5.2 gives an example of an XML format, GraphXML [21].

```
1  <node name="example">
2    <position x="20" y="20"/>
3    <size width="20" height="10"/>
4  </node>
5  <node name="example2">...</node>
6  <edge source="example" target="example2">
7    <path type="polyline">
8      <position x="10" y="5"/>
9      <position x="30" y="5"/>
10     <position x="30" y="20"/>
11   </path>
12 </edge>
```



Listing 5.2: Example of GraphXML showing node geometry example.

5.1.2 Level of Abstraction

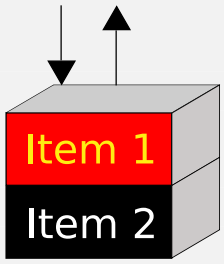
A distinguishing characteristic of the languages is the level of abstraction they use to describe the animations. One extreme is the languages that use graphical primitives to describe the animations. This approach allows the visualizer to visualize almost anything he/she wants to. The downside is that the creation of the visualizations is often time consuming. Listing 5.3 gives an example of graphical primitive visualization in JAWAA [2].

```
1 rectangle r1 10 10 100 50 black blue
2 oval o1 10 10 100 50 black orange
```

Listing 5.3: An example of JAWAA graphical primitives.

The other extreme is the animation languages that describe the animation using data structures. Listing 5.4 gives an example of using a stack in GaigsXML [44].

```
1  <snap>
2    <title>Stack example</title>
3    <stack>
4      <list_item color="red">
5        <label>Item 1</label>
6      </list_item>
7      <list_item color="black">
8        <label>Item 2</label>
9      </list_item>
10   </stack>
11 </snap>
```



Listing 5.4: Example of GaigsXML showing a stack example.

It should be noted, that it is typical for the languages with graphical primitives to have also some data structures. For example, JAWAA, mentioned in the example above, includes several data structures as well as the graphical primitives.

5.1.3 Animation

Since we are dealing with algorithm animation, the languages support also animating the visualizations. Again, animation by modifying the graphical primitives is the lowest level of abstraction. Listing 5.5 shows an example of graphical primitive animation of SAMBA [61].

```
1 circle c1 0.8 0.8 0.1 red half
2 rectangle r1 0.1 0.9 0.1 0.1 blue solid
3 comment Exchanging circle and rectangle!
4 exchangepos c1 r1
```

Listing 5.5: Example of Samba command language.

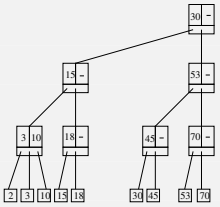
The other approach is again to modify the data structures using some of the operations specified for them. Listing 5.6 gives an example of animating an array in SALSA [22].

```
1 create array a1 with 3 cells
2 set a1[0] to 1
3 set a1[1] to 6
4 set a1[2] to 11
5 make a1[2] say "swapping me with 6"
6 swap a1[1] with a2[2]
```

Listing 5.6: Example of SALSA commands.

Listing 5.7 gives an example of using high-level data structure operations in DsCats language. In the example, keys are inserted into a B-Tree in two steps. Finally, a key is deleted from the tree. Note also the pause operation that requires the user to interact with the animation by restarting the play.

```
1 OPTION DS B-TREE
2 INSERT 20 15 30 2 18 24 70 3 45
3 INSERT 10
4 PAUSE -- End of inserts
5 DELETE 24
```



Listing 5.7: DsCats command language example. The figure represents the data structure after the operations are executed.

It should also be noted that not all the languages describe animations as modifications done to the visual objects. For example, GaigsXML approaches animation by allowing the visualizer to specify discrete snapshots of the state of the data structures. These snapshots are then visualized by the system.

5.1.4 Programming Concepts

Some of the languages support the creation of animations using programming constructs such as variables, conditionals, and loops. Listing 5.8 shows an example of the AnimalScript2 [53] programming concepts.

```
1  array "values" (10, 10) length 5 int {3, 2, 4, 1, 7}
2  int pos = 1
3  int minIndex = 0
4  arrayMarker "pos" on "values" at Index pos label "pos"
5  arrayMarker "minIndex" on "values" at Index minIndex
       label "minIndex"
6  while ( pos < 5 ) {
7    if ( values[pos] < values[minIndex] ) {
8      minIndex = pos ;
9      moveMarker "minIndex" to position pos within 5 ticks
10   }
11   pos = pos + 1
12   moveMarker "pos" to position pos within 5 ticks
13 }
14 arraySwap on "values" position 0 with minIndex within
       10 ticks
```

3	2	4	1	7

1	2	4	3	7

Listing 5.8: An example of programming concepts of AnimalScript2 [53]. The figure shows the array before (above) and after (below) the elements are swapped.

5.1.5 Interaction

Interaction is another feature of some of the animation languages. SALSA, for example, includes a command to request input data from the user. Listing 5.9 gives an example of this asking the user to give an integer value for variable var1 and integer values for elements in array arr1.

```
1 input var1 as integer between 1 and 20
2 input elements of arr1 as integers
```

Listing 5.9: Example of SALSA input command.

GaigsXML supports another kind of interaction requiring users of the visualization to respond to pop-up questions specified in the language. Listing 5.10 gives an example of the specification of a question in GaigsXML. Animal has also been extended to support this kind of interaction [54].

```
<show>
  <snap>
    ...
    <question_ref ref="0"/>
  </snap>
  ...
  <questions>
    <question type="MCQUESTION" id="0">
      <question_text>What will the value of node A be in
          the next step?</question_text>
      <answer_option>3</answer_option>
      <answer_option is_correct="yes">8</answer_option>
      <answer_option>5</answer_option>
    </question>
  </questions>
</show>
```

Listing 5.10: Example of GaigsXML's interactive questions.

5.2 Taxonomy of Algorithm Animation Languages

Based on a survey of the existing algorithm animation languages, we have defined *a Taxonomy of Algorithm Animation Languages* to evaluate the languages. The taxonomy is introduced in Publication [P4] and we only summarize it here. Figure 5.1 illustrates the two top levels of the taxonomy.

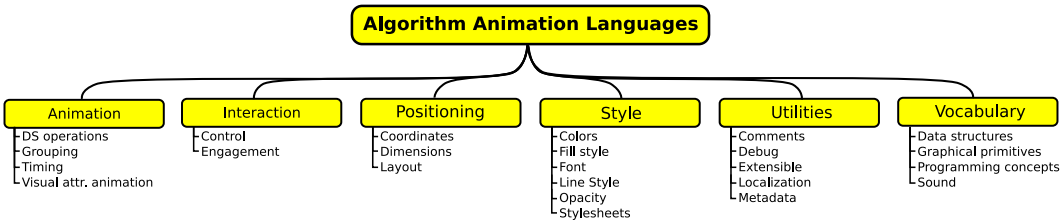


Figure 5.1: Taxonomy of Algorithm Animation Languages.

The main categories of the taxonomy are *Animation*, *Interaction*, *Positioning*, *Style*, *Utilities*, and *Vocabulary*. In the following, we will briefly describe the main categories of the taxonomy. For a more detailed discussion, see Publication [P4]. Publication [P4] also evaluates some of the algorithm animation languages introduced in the previous section using the taxonomy, here we only summarize the findings of the evaluation.

Category Animation The category Animation describes the level and versatility of animation effects available in the language and how the final animation can be customized through the language. Animation has four subcategories: *data structure (DS) operations*, *grouping*, *timing*, and *visual attribute animation*.

Category Interaction The category Interaction describes the type and level of interaction that can be specified using the language. Interaction has two subcategories: *Control* and *Engagement*. When using the taxonomy, emphasis should be placed on distinguishing between the interaction provided by the tool and interaction supported by the language. This distinction is not always obvious.

Category Positioning Category Positioning describes the ways available in the language to position the objects in the animation. Positioning has three subcategories: *coordinates*, *dimensions*, and *layout*.

Category Style The category Style measures the variety of styling options available in the language. By styling, we mean setting the style of the objects in the language's vocabulary. Style is divided in six subcategories: *colors*, *fill style*, *font*, *line style*, *opacity*, and *stylesheets*.

Category Utilities The category Utilities describes the support of features that are not directly related to algorithm animation but instead are useful in the animation creation process. The subcategories in Utilities are *comments*, *debug*, *extensible*, *localization*, and *metadata*.

Category Vocabulary Vocabulary describes the amount of supported object types. These are the building blocks used to compose the animation. Vocabulary has four subcategories: *data structures*, *graphical primitives*, *programming concepts*, and *sound*.

Summary and Discussion In this section, we have introduced a Taxonomy of Algorithm Animation Languages. As a result, we have a more detailed overview of the features and properties of the languages. In Publication [P4] we evaluated several algorithm animation languages. For comparison purposes we also evaluated Scalable Vector Graphics (SVG) [65]. The evaluation done could be summarized by stating again that there are languages supporting graphical primitives and languages supporting data structures. In addition, SVG has the most advanced features in

many of the categories, especially when SVG is used together with ECMAScript. However, SVG is missing the data structures that are essential in AA.

The evaluation of the languages is straightforward, although it requires quite deep knowledge and understanding of the evaluated languages. In the future, as AA languages are developed further and new features emerge, this taxonomy is likely to be outdated. In such case, updates to the taxonomy should be made. To support this, we have made an up-to-date version available online[1] hoping that the research community would contribute to it.

[1] `http://svg.cs.hut.fi/aaltaxonomy/`

Chapter 6

Data Exchange in Algorithm Animation

One of the main questions in this work is how to allow data exchange between algorithm animation systems. In this chapter, we will present our solution to the problem. First, we will introduce the international work aiming at a common algorithm animation language. Based on that work and the taxonomy defined earlier, we will introduce a new algorithm animation language. Finally, we will describe our implementation of data exchange using the new AA language.

6.1 ITiCSE Working Group

In the Conference on Innovation and Technology in Computer Science Education (ITiCSE) 2005 a working group titled *"Development of XML-based Tools to Support User Interaction with Algorithm Visualization"* convened to come up with XML specifications to support algorithm animation. The group visioned a set of features of a common algorithm animation language and wrote a report that introduces them (see Publication [P2]). The different aspects are:

- high level *objects* (often, data structures)
- *graphical primitives* and *transformations* on them
- *narrations* (text, graphics, and audio) related to the visualization
- *questions* related to the visualization
- *metadata* describing the different resources (questions, animation, etc)

In the next section when we discuss XAAL, we have adopted some of the ready defined and suitable parts of the WG specifications, in order to support the interna-

tional goal of a uniform algorithm animation language specification. However, most of the aspects of AA were not formally defined by the working group. Thus, not all of these specifications are used as a part of the new language.

6.2 XAAL

Based on the Taxonomy of Algorithm Animation Languages and the work by the ITiCSE Working Group we have defined a new AA language, XAAL (eXtensible Algorithm Animation Language). XAAL is defined as an XML language by specifying the allowed document structure. XML makes it easy for any software to process data using the multitude of different tools and architectures available today. In addition, transforming XML documents to different XML formats or text is relatively simple and flexible using XSLT (Extensible Stylesheet Language Transformations).

The following will briefly introduce the most important features of XAAL. The reader should note that this text is merely an overview of the language. For a more detailed discussion, see Publication [P5] and [27], and for the actual XML schemas, see the XAAL website[1].

Graphical Primitives The basic graphical components that can be composed to represent arbitrarily complex objects (*e.g.*, a tree data structure) are graphical primitives. The graphical primitives in XAAL are as specified by the Working Group, where the following primitives have been defined: point, polyline, line, polygon, arc, ellipse, circle and circle-segment, square, triangle, rectangle, and text.

Data Structures XAAL supports the usage of data structures to specify the visualizations, lowering the effort needed to produce them. The set of structures is basically the same as, for example, in JAWAA [2]: array, graph, list, and tree. To support the different approaches of existing algorithm animation languages, all structures support an optional graphical presentation indicating how the structure should be visualized.

Animation A crucial part of the algorithm animation language is the animation functionality. The *animation operations* in XAAL have been divided in three groups: graphical primitive transformations (for example, rotate), elementary data structure operations (for example, replace), and abstract data structure operations (for example, insert). Every abstract operation can contain the same transformation on a

[1] `http://www.cs.hut.fi/Research/SVG/XAAL/`

lower level of abstraction as graphical primitive transformations and as elementary data structure operations. However, these are both optional.

6.3 Implementing Data Exchange

Publication [P2] describes a general architecture of a system using the specifications of the group. The architecture describes how to combine the XML documents describing interesting events (operations, etc), objects, questions, narrations, metadata, and graphical primitives into one, *complete visualization specification* of the animation. The idea is that this complete visualization specification would then be used by existing visualization systems through system-specific *adapters*.

The XAAL language can be seen as a simplified version of the complete visualization specification, as it includes most of the data in the different XML documents mentioned above. Thus, our objective is to implement XAAL in a way that could be useful for other developers aiming at implementing the WG specifications.

In this section, we will briefly introduce two different processing pipelines to add XAAL support into existing algorithm animation systems. For more discussion, see Publication [P5] and [27].

Object hierarchy The first processing solution is an architecture discussed at the ITiCSE Working Group to implement the language. This architecture is represented in Figure 6.1. The basic idea is to have one XAAL *parser* that can be used by multiple algorithm animation systems. This parser generates a Java *object hierarchy* (in the figure, XAAL objects). In addition, there is a part of software that can *serialize* the object hierarchy as a XAAL XML document.

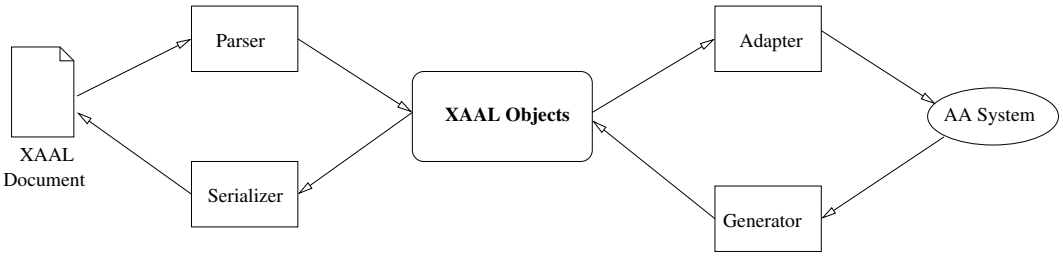


Figure 6.1: Integrating XAAL with existing AA systems using an object hierarchy.

The existing AA systems can implement *adapters* that convert the XAAL object hierarchy into an animation in that particular system. By implementing a *generator*, existing systems can generate the object hierarchy and serialize it as XAAL.

The natural benefit of this approach is that the object hierarchy allows API based generation of XAAL animations.

XSL Processing Another way to integrate XAAL with existing systems is represented in Figure 6.2. In this solution, an existing XAAL document is processed with an XSL processor using an appropriate XSL stylesheet.

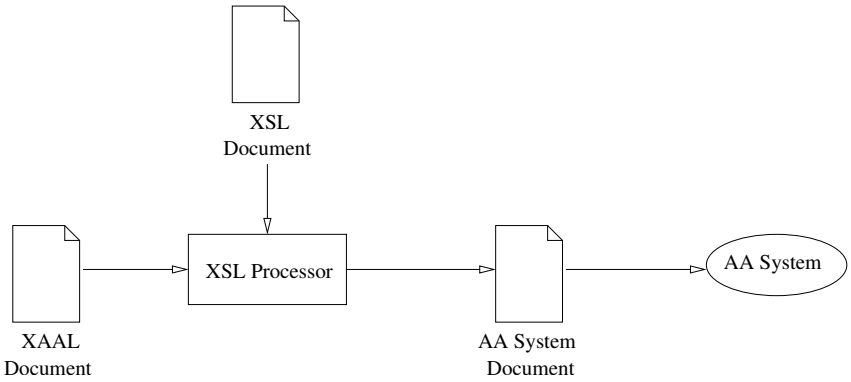


Figure 6.2: Integrating XAAL with existing AA systems using XSL stylesheet rocessing.

The stylesheet is a mapping from XAAL to the language of the target system. This provides a simple solution to import XAAL documents into existing AA systems that have an algorithm animation language. It can also be used to export different formats from a system that supports XAAL. The benefit of this approach is that the target system needs not be changed at all. This makes it possible to integrate XAAL with systems that are not open-source.

6.3.1 Prototype Implementations

We have a prototype implementation of the language and transformations between various existing algorithm animation languages. In the following, we will briefly describe these prototypes and discuss the advantages and disadvantages of the different solutions, as well as state the level of XAAL features supported.

Xaal Objects and Xaal Parser The center of the XAAL implementation is XAAL Objects (XO). This is a collection of Java classes that correspond to the different elements and attributes in XAAL documents.

The XAAL objects hierarchy can be generated in multiple ways, the most natural of which is the XAAL parser. The XAAL objects and XAAL parser prototype implementations support most of the elements and functionality specified in the language.

However, one major lack is that the current parser does not behave well when the source document is not well-formed XML.

Adapters and Generators An important part of the implementation is the generator that allows the generation of XAAL Objects from MatrixPro animations.

The adapter implementation allows us to adapt XO hierarchies to be used in MatrixPro. It essentially generates a MatrixPro animation sequence from the XAAL objects hierarchy. The adapter supports the data structures and data structure operations of XAAL. Another adapter allows to view the graphical primitives of XAAL in JHAVÉ [46].

XSL Stylesheets In this thesis, we have implemented the XSL solution to translate XAAL documents to ANIMALSCRIPT, JAWAA, and SVG. These implementations are only prototypes, and do not cover the whole range of features in the language.

6.4 Evaluation

6.4.1 Taxonomic Evaluation

In this section we will use the taxonomy defined in Publication [P4] to evaluate XAAL. In addition, we will include the evaluation of SVG and compare XAAL with SVG. Reason for using SVG is that in Chapter 5 we concluded SVG having the richest set of features in many of the categories.

Category Animation Evaluation of XAAL in category Animation is in Tables 6.1 and 6.2. Compared to the existing AA languages [2, 15, 22, 51, 53], XAAL has quite a rich set of data structure operations. However, these require advanced features from the system implementing the language. SVG, on the other hand, lacks the data structure and operations on them. Thus, it has no features that make it especially suitable for algorithm animation.

In both languages, all the style properties (see Category Style) can be changed. In XAAL, the graphical primitive transformations available are the ones defined by the ITiCSE XML Working Group. Thus, XAAL fulfills the requirements for an algorithm animation language as seen by the international AA community. However, these features are not as versatile as in SVG.

Table 6.1: Evaluation of the languages in category Animation (1/2).

Language	**DS operations**	**Grouping**		**Timing**
		Granul. control	Concurrency	
XAAL	create, remove, replace, swap, insert, delete, and search	yes	yes	delay, duration
SVG	none	yes	yes	delay, duration, min, max, repeat, key times

Table 6.2: Evaluation of the languages in category Animation (2/2).

Language	**Visual attribute animation**			
	Rotate	Scale	Style	Translate
XAAL	yes	yes	all style properties	move, move relative, move along
SVG	yes	yes	all style properties	move, move relative, move along

Category Interaction The evaluation of XAAL in category Interaction is represented in Table 6.3. Since SVG documents can include ECMAScript [1] code and is often used this way, we have evaluated pure SVG and SVG+ECMAScript separately. This is because the evaluation results differ significantly.

The only interaction that XAAL supports is pausing the animation. This is a known lack of the language since interaction is important. However, we decided not to define any interaction, since the Working Group has not finished its specification. Thus, we can include more interaction techniques in the language in the future. SVG+ECMAScript can be considered to support any kind of interaction. For example, interaction on responding level can be implemented by showing questions for the student, thus the type would be pop-up questions.

Table 6.3: Evaluation of the languages in category Interaction.

Language	**Control**	**Engagement**	
		Type	Level
XAAL	pause	none	none
SVG	none	none	none
SVG + ECMAScript	any	any	any

Category Positioning Evaluation of XAAL in category Positioning is in Table 6.4. Like many of the existing AA languages, XAAL supports 2 dimensions with the additional depth setting for overlapping objects. Layout for data structures can be specified in XAAL but this is not required. This allows it to be used in tools that support automatic layout as well as in tools where the layout must be user specified. Except for layout, features in SVG are quite similar in this category.

Table 6.4: Evaluation of the languages in category Positioning.

Language	**Coordinates**	**Dimensions**	**Layout**
XAAL	absolute, relative to a location	2.5	can be specified but is not required
SVG	absolute, relative	2	n/a

Category Style Evaluation of XAAL in category Style is in Tables 6.5 and 6.6. XAAL supports colors as RGB values and some predefined color names (the same 17 colors as in CSS2 [8]). Compared to existing AA languages, the styling options in XAAL are more than adequate. However, SVG has a more diverse set of styling options, and including these in XAAL remains a future challenge.

The advanced feature compared to the existing AA languages is the support for reusable and extensible stylesheets. These are not, however, as versatile as in SVG due to the more limited styling functionality of XAAL.

Category Utilities Evaluation of XAAL in category Utilities is in Table 6.7. In XAAL, we decided not to use any standard for the metadata due to the sheer complexity of such standards. XAAL has, however, support for more metadata than the existing AA languages. We also believe that including a small but specified

Table 6.5: Evaluation of the languages in category Style (1/2).

Language	**Colors**	**Fill style**	**Font**		
			Family	Size	Variant
XAAL	RGB-values, 17 predefined	solid	serif, sans serif, monospaced	yes	bold, italic
SVG	RGB-values, 16 predefined	solid, gradient, pattern	serif, sans-serif, cursive, fantasy, monospaced, user specified	yes	bold, italic, small caps

Table 6.6: Evaluation of the languages in category Style (2/2).

Language	**Line style**	**Opacity**	**Stylesheets**
XAAL	arrows, solid, dashed, dotted, width	yes	yes
SVG	width, dash pattern, color, line end, line join	yes	yes

amount of metadata is more beneficial than allowing arbitrary metadata, as done in SVG. Again, in future versions, we might decide to also endorse some metadata standard.

Table 6.7: Evaluation of the languages in category Utilities.

Language	**Comments**	**Debug**	**Extens.**	**Localiz.**	**Metadata**
XAAL	yes	no	yes	yes	author, application used, and animation description, subject, and keywords
SVG	yes	yes	yes	yes	any

Category Vocabulary Evaluation of XAAL in category vocabulary is in Table 6.8. The graphical primitives in XAAL are the same supported by most of the AA languages and SVG. However, there are more data structures supported in XAAL than in SVG. Programming concepts are currently not supported in XAAL and

thus these are not included in the table. SVG with ECMAScript has advanced programming functionality.

Table 6.8: Evaluation of the languages in category Vocabulary.

Language	**Data structures**	**Graphical primitives**	**Sound**
XAAL	array, graph, list, and tree	point, polyline, polygon, line, arc, ellipse, circle and circle-segment, square, triangle, rectangle, and text	yes
SVG	none	rectangles, circles, ellipses, polylines, polygons, text	no

6.4.2 Implementation-based Evaluation

Although the language does not include some of the most complex features that came up (for example, programming concepts), XAAL is still quite a complex language. The current prototype implementation is a good indicator of this, since the original aim of this thesis was to provide a full implementation. However, that was not achieved due to the limited time to finish the thesis.

Nevertheless, we can consider the feasibility of XAAL as an intermediate language. The main problem in the implementation are the different levels of abstraction. Transformations of animations from one abstraction level to another are bound to lose some information. In addition, unless the source format includes all the necessary information, it is difficult to transform animations between different levels of abstraction. For example, so far the only system capable of adding graphical primitive information into XAAL documents is MatrixPro. Thus, we must include another processing step when transforming a XAAL document containing only data structures into, say, SVG. Another problem is caused by languages such as GaigsXML, where the structure of the language is based on snapshots of the animation. Since XAAL presents the animation as modifications to the elements, conversion of animation between these languages would probably require some complex XSL templates. However, we believe that this could definitely be implemented.

All in all, the transformations from XAAL to the other languages is pretty straightforward. An interesting problem would be to try to transform, for example, ANIMALSCRIPT into XAAL, and try to recognize possible data structures from the graphical primitives. In this thesis, we did not, however, have the time to implement an ANIMALSCRIPT parser that would be required to do such transformations.

Chapter 7

Discussion

In this chapter, we will summarize our work and address the research questions. First, we will discuss the first question: *"How can we develop algorithm animation systems to lower the effort needed to produce algorithm visualizations for teaching?"* Next, we will tackle the second problem: *"How can we enable data exchange between the existing algorithm animation systems?"* Finally, we will consider possible future directions of this research.

7.1 Lowering the Effort

To make the algorithm animation production more effortless, we started researching what is effortless production. In the first step we identified that there are either specific, low effort systems or general, high effort systems [29]. This research was, however, our subjective view of the subject. The next step was a survey targeting computer science educators [30]. The survey resulted in an initial set of measures for effortlessness. Finally, in Publication [P3] we introduced a taxonomy of effortless creation of algorithm animations.

In the first step of the research on effortlessness, we found that Matrix [36] allowed effortless creation of animations. However, it was a research prototype demonstrating the features of the framework and not suitable for end-users. Thus, we developed a Matrix-based application, MatrixPro (see Publication [P1]). MatrixPro was designed to support *on-the-fly* demonstrations without the need to prepare all the examples before lectures. The most important feature supporting this was the automatic animation of several ready made data structures. MatrixPro is effortless to use for the narrow scope it was designed.

7.2 Data Exchange

To allow data exchange, we have first surveyed a number of algorithm animation systems and the algorithm animation languages they use. Based on this survey, we have defined a Taxonomy of Algorithm Animation Languages to help in comparing the different AA languages. This taxonomy helped us in defining a new algorithm animation language, eXtensible Algorithm Animation Language, XAAL.

XAAL supports both of the two main approaches in the existing AA languages: graphical primitives and data structures. We have implemented various adapters and generators between XAAL and other algorithm animation languages. The current selection of formats is presented in Figure 7.1. As can be seen, MatrixPro is currently the only AA system capable of creating XAAL animations. In addition, we have a transformation from SVG to XAAL. XAAL documents can then be transformed to ANIMALSCRIPT, JAWAA, and SVG, or viewed with a JHAVÉ visualization plugin. Thus, we already have several different formats available for the same animation.

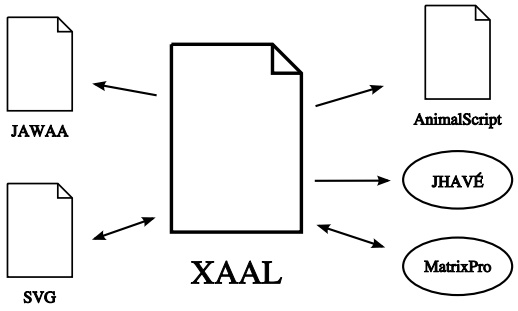


Figure 7.1: Prototype format transformations implemented in this thesis. The arrows represent the direction of the transformation.

When considering the current implementation from the effortlessness point of view, we can say that we have made it possible to transform animations from an effortless system (MatrixPro) to more general purpose tools (ANIMAL, JAWAA, SVG). This allows us, for example, to easily create an example of a complex topic, say B-Tree in MatrixPro, transform it to ANIMALSCRIPT and customize the animation with ANIMAL. In addition, as the JHAVÉ system is intended as a visualization platform, XAAL implementation for that platform is a good step towards more general tool integration.

In Chapter 6, we introduced two different processing pipelines to implement the data exchange: 1) parsing the XAAL document into a set of (Java) objects and transforming that, and 2) transforming the XAAL document using XSLT. A future

challenge is to implement data exchange between more systems. Thus, the following considers the suitability of the two processing pipelines for the different visualization specification styles introduced in Section 3.2.

Topic-Specific Animation As topic-specific animations are not animation systems, it probably is not worth the effort to implement any data exchange with such animations.

Direct Manipulation Direct manipulation as a visual specification style can be implemented in a multitude of ways. Thus, it is not feasible to speculate how the data exchange with such a system could be implemented as it depends completely on the system architecture. For example, the XAAL import/export in MatrixPro is implemented by transforming an animation between the internal object hierarchy of Matrix and the XAAL object hierarchy.

API-based Generation In API-based generation there is some programming API that can be used to generate the animations. Thus, the natural method for implementing data exchange in such cases is to transform the Java object hierarchy to suitable method calls of the API.

Scripting-based Generation In scripting-based generation, the animation system has some scripting language (or, algorithm animation language) that it understands. Thus, using XSLT to transform XAAL documents into this scripting language is the most sensible option. However, transforming the object hierarchy might be a useful solution as well, especially if there are significant differences between XAAL and the target language. When transforming an animation from the scripting language to XAAL, XSLT is a suitable solution if the scripting language is XML. Otherwise, it requires a parser of the scripting language.

Declarative Visualization In declarative visualization, the visualization is specified by declaring a mapping between a program state and a graphical representation. Generally, transforming between this mapping and XAAL is not a suitable approach, since XAAL does not have a program state attached. Thus, the best approach is again completely dependent on the system architecture.

Code Interpretation Implementing data exchange with a tool that uses code interpretation is not meaningful from XAAL to the system. The other way around, it could be beneficial. However, the implementation depends completely on the architecture of the system.

From the discussion above, we can summarize that it is not obvious in most of the cases how the data exchange is best implemented. The best approach is

typically dependent on the architecture of the animation system. However, in the case of API-based generation and Scripting-based generation, natural choices are transforming the Java object hierarchy and XSL transformations, respectively. It should be noted, that often systems have more than one visualization specification style so there will be different possibilities to implement data exchange as well.

7.3 Future Work

As can be seen in the evaluation of Xaal in the previous chapter, it does not have all the necessary features at this point. We have numerous improvements and ideas for the future of the language, and here we will write down some of the most interesting ones. The most urgent requirement is naturally to finish the prototype implementation of the parser and the adapters and generators. A future challenge is to be able to generate Xaal documents with other systems, or alternatively, parse/transform other formats into Xaal.

The language could be extended to include programming concepts and thus allow the definition of algorithms and program visualization. This could be achieved, for example, by allowing o:XML[1] notation to be included in Xaal documents.

Support for interaction should also be added to the language. This could possibly use the specification of the ITiCSE XML Working Group once it finishes the definitions. Another option is to adopt the questions from the specification of GaigsXML.

Yet another future goal is to allow easy integration of algorithm visualizations into hypertextbooks [55]. One way to integrate Xaal animations into hypertextbooks is the implementation for the JHAVÉ environment. However, there could be even easier ways to do this by allowing the end-users (in our case, students) to view algorithm animations without a dedicated algorithm animation system. Our vision is to implement a viewer for Xaal animations for the web browsers. With the rapid development of JavaScript libraries[2] to aid in such process, it would be interesting to implement such a viewer using only HTML and JavaScript.

7.4 Final Remark

We feel that the new language can be successfully used in exchanging data between different algorithm animation systems. However, it would require a more complete implementation of the language and the adapters and generators to get the full benefit of the language.

[1] o:XML is an XML language for object-oriented programming, see `http://www.o-xml.org/`.
[2] For example, Dojo, MooTools, Prototype, and Scipt.aculo.us just to mention a few.

Appendix A

Algorithm animation systems

In this section, we will introduce in more detail some of the algorithm animation systems introduced in Section 3.3. The reader should note that these descriptions are not supposed to be exhaustive feature lists of the tools. They are merely intended to give a general overview of the tools and give relevant information about the I/O functionality and implementation of the languages. Thus, we encourage the interested reader to read more about the tools from the cited articles.

A.1 ALVIS

General ALgorithm VIsualization Storyboarder (ALVIS) [22] is an interactive algorithm visualization system designed to create and view *low-fidelity* algorithm visualizations. In low-fidelity visualizations, the algorithm is illustrated with few inputs using a sketched, unpolished appearance.

Goal and targeted users The goal of the system is to make presenting algorithms easy and the system includes features supporting this. The system has been used in an undergraduate algorithmics course.

Animation specification The generation of the animations is done using direct manipulation. In addition, the tool supports dynamic modification of the animation by directly inserting commands of the language of ALVIS, SALSA, into the code.

Interaction The system is designed to support interaction levels constructing and presenting, thus it has several features supporting this. The direction of execution can be reversed to go back to a previous view. For highlighting interesting parts while showing the visualization, the tool includes a presentation pointer tool. Moreover, there is a tool that can be used as a mark-up pen to dynamically annotate the animation.

I/O As stated above ALVIS uses a command language called SALSA. The SALSA scripts can be saved and loaded with the system.

Implementation ALVIS is implemented using Visual Studio .NET, thus the system is platform dependent. ALVIS is intended to be used as-is, so no implementation on the behalf of the user is possible or required.

A.2 Animal

General ANIMAL [52] is a general-purpose animation tool with a current focus on algorithm animation.

Goal and targeted users ANIMAL is designed to support multiple roles. The tool has features for developers, visualizers, and users. The authors of the tool feel that the most important of these is the user that views the animations.

Animation specification Visualizers are provided with three different methods to specify the animation. He/she can use the graphical editor to create the animations by drag and dropping graphical elements. Another method is to use the scripting language, ANIMALSCRIPT [51]. Finally, the tool supports generation of the scripts through a Java API. The developers of the system expect advanced users to use either the scripting language or the API.

Interaction The tool offers a viewer with versatile controls over the animation for viewing and presenting. The user is also able to set the speed and scale of the animation.

ANIMAL has also been extended to support responding by adding multiple-choice questions to the animations that require answers from users before continuing the viewing [54].

I/O Scripts in the ANIMALSCRIPT language can be saved and loaded with the system. ANIMAL also supports a wide variety of export formats including BMP, JPG, PNG, Photoshop PSD, and QuickTime videos.

Implementation The tool is implemented in Java and offers developers the possibility to add extensions to the tool. These extensions can be, for example, new graphical primitives, animation effects, scripting commands, or import/export filters. In addition, making an internationalized version for a new language is straightforward.

A.3 DsCats

General Data Structure Computer Animation Tools (DsCats) [15] is an application focused on learning data structures and algorithms.

Goal and targeted users The goal of the tool is to help learning of basic data structures and algorithms. It is intended for students as a self-study material.

Animation specification The animations can be created either by using the tool or by writing a text file using a simple command language. In the tool, inserting and deleting keys to and from the structures is done by simply entering the key values and pressing appropriate buttons.

Interaction The tool has many features supporting viewing of animations. The most important features include the ability to move backward and forward in the animation sequence, support for visualizing large data sets, and the possibility to vary the level of detail of the animation. The tool also automatically adds comments on the steps when inserting or deleting keys into the structures making the animations useful as self-study material.

I/O In the tool, the animations can be saved and restored. For this purpose, the system includes a simple command language.

Implementation The current version of the tool includes implementations for tree structures (binary search tree, AVL-tree, and B-tree). However, the tool is implemented in Java and is designed to be extensible, thus allowing users to extend the selection of data structures. This can be done by extending one base class of the tool.

A.4 JAWAA

General JAWAA 2.0 [2] (in the following, simply JAWAA) is a data structure and algorithm animation system that consists of three parts: a scripting language, graphical editor, and an applet capable of showing animations defined in the scripting language.

Goal and targeted users The tool has been used on CS courses of different levels giving students tasks to construct visualizations of data structures and algorithms with JAWAA.

Animation specification JAWAA supports two ways to create the animations. The JAWAA Editor is a graphical user interface allowing the visualizer to create animations graphically. This is done by laying out primitive objects and modifying them to produce animations. The animations are defined by changing the states

of the objects across time. Another way is to write or generate the animations using the JAWAA scripting language. These methods can of course be combined by creating the layout of the objects with the editor and then modifying these objects using output of a program.

Interaction The JAWAA applet that views the animations provides basic control over the animation. The user can play, stop, and pause it as well as change the speed. The authors of the system use the whole JAWAA system in teaching by requiring students to construct animations.

I/O The animations in the JAWAA Editor can be saved and loaded in an internal format. Animations can also be exported in the JAWAA scripting language and added to web pages to be viewed with the JAWAA applet.

Implementation JAWAA is implemented in Java and is freely available with the source code. However, extending is not documented by the authors and the system is intended to be used as-is.

A.5 JHAVÉ

General JHAVÉ [46] is not an AV system but rather an environment for different AV systems called AV engines. The aim of the environment is to provide a standard user interface for students.

Animation specification As JHAVÉ itself is not an AV system, animation specification is not supporter. However, the different AV engines support different types of animation specification. The main AV engine, Gaigs, supports animation specification through API-calls and has an XML-language [44]. Another AV engine is the Xaal engine implemented as part of this thesis.

Interaction JHAVÉ environment offers standard VCR-like controls for the user. Furthermore, it supports engagement level responding by offering functionality to ask users multiple-choice questions.

I/O The animations are loaded by the AV engines, typically from text files. The Gaigs AV engine loads the animations from GaigsXML-files and Xaal AV engine from Xaal-documents. In addition, there are AV engines that render Samba and AnimalScript animations.

Implementation JHAVÉ is implemented in Java but the source code is not freely available at the moment of writing. However, documentation on how to extend the system is available.

A.6 JSAMBA

General JSAMBA [59] is a front-end for the POLKA [58] algorithm animation system. Basically, it is a viewer to visualize animations written in the Samba scripting language [61].

Animation specification The animations are specified by entering (or copy pasting) Samba commands into a text area. The system does not include an animation editor.

Interaction The tool offers basic control over the animation with play, step, and pause controls as well as the possibility to change the speed of the animation.

I/O Since the tool is an applet, it has no access to the file system and the animation scripts are either selected from a menu of a variety of ready made examples or typed/cut and pasted as Samba commands into a text area.

Implementation The tool is implemented in Java and is freely usable in the Internet, but the source code or API documentation is not available. Thus, the tool cannot be extended.

A.7 MatrixPro

General MatrixPro (see Publication [P1] for details) is a system to create algorithm animations using visual algorithm simulation.

Goal and targeted users MatrixPro is targeted for computer science educators to use on lectures. The main idea is to support *on-the-fly* usage, thus enabling the instructor to create the animations during the lecture. For the students, the tool offers numerous automatically assessed exercises [40].

Animation specification The animations are specified using visual algorithm simulation. The simulation consists of manipulating visual representations of actual data structures through a graphical user interface. The tool includes a library of data structures that the user can operate on.

Interaction To support the presenting of animations, the tool has a number of useful features. For example, the tool offers ways to customize the animations and the user interface.

The exercises for students offer interaction where the user has to simulate the working of an algorithm. Student's simulation sequence can then be automatically assessed.

I/O The animations created with MatrixPro can be saved as serialized Java objects or exported as Scalable Vector Graphics animations. A view in the animation can be saved into an ASCII file or exported in TEXdraw format to be included in LaTeX documents. In the current version, the view or animation can also be exported as a series of PNG pictures. As part of this thesis, we have also implemented XAAL export.

Implementation MatrixPro is based on the Matrix algorithm simulation framework [34] that is implemented in Java and designed to be extended. For example, a developer can add new data structures or layouts for the visualizations.

Bibliography

[1] ECMAScript language specification, 3rd ed. Technical report, Ecma International, December 1999.

[2] Ayonike Akingbade, Thomas Finley, Diana Jackson, Pretesh Patel, and Susan H. Rodger. JAWAA: easy web-based animation from CS0 to advanced CS courses. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education, SIGCSE'03*, pages 162–166, Reno, Navada, USA, 2003. ACM Press.

[3] Jay Martin Anderson and Thomas L. Naps. A context for the assessment of algorithm visualization systems as pedagogical tool. In *Proceedings of the First Program Visualization Workshop*, pages 121–130, Porvoo, Finland, 2001. University of Joensuu.

[4] Ronald M. Baecker. Two systems which produce animated representations of the execution of computer programs. In *Proceedings of the fifth SIGCSE technical symposium on Computer science education, SIGCSE'75*, pages 158–167. ACM Press, 1975.

[5] Ronald M. Baecker. Sorting out sorting. Narrated colour videotape, 30 minutes, 1981.

[6] Ronald M. Baecker. Sorting out sorting: A case study of software visualization for teaching computer science. In M. Brown, J. Domingue, B. Price, and J. Stasko, editors, *Software Visualization: Programming as a Multimedia Experience*, chapter 24, pages 369–381. The MIT Press, Cambridge, MA, 1998.

[7] S. Bassil and RK Keller. Software visualization tools: survey and analysis. *Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on*, pages 7–17, 2001.

[8] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. Cascading Style Sheets, CSS 2.1 specification. W3C Candidate Recommendation, World Wide Web Consortium, July 2007.

[9] M. H. Brown and R. Raisamo. JCAT: Collaborative active textbooks using Java. *Computer Networks and ISDN Systems*, 29(14):1577–1586, 1997.

[10] Marc H. Brown. Fundamental techniques for algorithm animation displays. In M. Brown, J. Domingue, B. Price, and J. Stasko, editors, *Software Visualization: Programming as a Multimedia Experience*, chapter 7, pages 82–101. The MIT Press, Cambridge, MA, 1998.

[11] Marc H. Brown and John Hershberger. Color and sound in algorithm animation. *Computer*, 25(12):52–63, 1992.

[12] Marc H. Brown and Marc A. Najork. Algorithm animation using 3D interactive graphics. In *Proceedings of the 6th annual ACM symposium on User interface software and technology, UIST'93*, pages 93–100, Atlanta, Georgia, United States, 1993. ACM Press.

[13] Marc H. Brown and Robert Sedgewick. A system for algorithm animation. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques, SIGGRAPH'84*, pages 177–186. ACM Press, 1984.

[14] Marc H. Brown and Robert Sedgewick. Techniques for algorithm animation. *IEEE Software*, 2(1):28–39, January 1985.

[15] Justin Cappos and Patrick Homer. DsCats: Animating data structures for CS2 and CS3 courses. Technical paper published online, 2002. `http://www.cs.arizona.edu/dscats/dscatstechnical.pdf`.

[16] C. Demetrescu and I. Finocchi. Smooth animation of algorithms in a declarative framework. *Journal of Visual Languages and Computing*, 12(3):253–281, 2001.

[17] S. Diehl. *Software visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer New York, 2007.

[18] Joan M. Francioni, Larry Albright, and Jay Alan Jackson. Debugging parallel programs using sound. *SIGPLAN Notices*, 26(12):68–75, December 1991.

[19] P. A. Gloor. User interface issues for algorithm animation. In M. Brown, J. Domingue, B. Price, and J. Stasko, editors, *Software Visualization: Programming as a Multimedia Experience*, chapter 11, pages 145–152. The MIT Press, Cambridge, MA, 1998.

[20] Jyrki Haajanen, Mikael Pesonius, Erkki Sutinen, Jorma Tarhio, Tommi Teräsvirta, and Pekka Vanninen. Animation of user algorithms on the Web. In *Proceedings of Symposium on Visual Languages*, pages 360–367, Isle of Capri, Italy, 1997. IEEE.

[21] Ivan Herman and M. Scott Marshall. GraphXML - an XML-based graph description format. In *Graph Drawing*, pages 52–62, 2000.

[22] Christopher D. Hundhausen and Sarah A. Douglas. Low-fidelity algorithm visualization. *Journal of Visual Languages and Computing*, 13(5):449–470, October 2002.

[23] Christopher D. Hundhausen, Sarah A. Douglas, and John T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13(3):259–290, June 2002.

[24] Juha Hyvönen and Lauri Malmi. TRAKLA – a system for teaching algorithms using email and a graphical editor. In *Proceedings of HYPERMEDIA in Vaasa*, pages 141–147, 1993.

[25] Petri Ihantola, Ville Karavirta, Ari Korhonen, and Jussi Nikander. Taxonomy of effortless creation of algorithm visualizations. In *Proceedings of the 2005 international workshop on Computing education research*, pages 123–133, New York, NY, USA, 2005. ACM Press.

[26] Peter Kabal. TEXdraw – PostScript drawings from TEX. Web page, 1993. `http://www.tau.ac.il/cc/pages/docs/tex-3.1415/texdraw_toc.html`.

[27] Ville Karavirta. XAAL - extensible algorithm animation language. Master's thesis, Department of Computer Science and Engineering, Helsinki University of Technology, December 2005. Available online at `http://www.cs.hut.fi/Research/SVG/publications/karavirta-masters.pdf`.

[28] Ville Karavirta, Ari Korhonen, Lauri Malmi, and Kimmo Stålnacke. MatrixPro - A tool for on-the-fly demonstration of data structures and algorithms. In *Proceedings of the Third Program Visualization Workshop*, pages 26–33, The University of Warwick, UK, July 2004.

[29] Ville Karavirta, Ari Korhonen, Jussi Nikander, and Petri Tenhunen. Effortless creation of algorithm visualization. In *Proceedings of the Second Annual Finnish / Baltic Sea Conference on Computer Science Education*, pages 52–56, October 2002.

[30] Ville Karavirta, Ari Korhonen, and Petri Tenhunen. Survey of effortlessness in algorithm visualization systems. In *Proceedings of the Third Program Visualization Workshop*, pages 141–148, The University of Warwick, UK, July 2004.

[31] Andreas Kerren and John T. Stasko. Algorithm animation. In Stephan Diehl, editor, *Software Visualization: International Seminar*, pages 1–15, Dagstuhl, Germany, 2001. Springer.

[32] Sami Khuri and Hsiu-Chin Hsu. Interactive packages for learning image compression algorithms. *SIGCSE Bull.*, 32(3):73–76, 2000.

[33] K. C. Knowlton. L^6: Bell telephone laboratories low-level linked list language. 16 mm black and white sound film, 16 minutes, 1966.

[34] Ari Korhonen. *Visual Algorithm Simulation*. Doctoral dissertation (tech rep. no. tko-a40/03), Helsinki University of Technology, 2003.

[35] Ari Korhonen and Lauri Malmi. Algorithm simulation with automatic assessment. In *Proceedings of The 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'00*, pages 160–163, Helsinki, Finland, 2000. ACM Press, New York.

[36] Ari Korhonen and Lauri Malmi. Matrix — Concept animation and algorithm simulation system. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 109–114, Trento, Italy, May 2002. ACM Press, New York.

[37] Ari Korhonen, Erkki Sutinen, and Jorma Tarhio. Understanding algorithms by means of visualized path testing. In Stephan Diehl, editor, *Software Visualization: International Seminar*, pages 256–268, Dagstuhl, Germany, 2002. Springer.

[38] Markus Krebs, Tobias Lauer, Thomas Ottmann, and Stephan Trahasch. Student-built algorithm visualizations for assessment: flexible genera tion, feedback and grading. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE*

conference on In novation and technology in computer science education, pages 281–285, New York, NY, USA, 2005. ACM Press.

[39] Jan Lönnberg, Ari Korhonen, and Lauri Malmi. MVT — a system for visual testing of software. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI'04)*, pages 385–388, May 2004.

[40] Lauri Malmi, Ville Karavirta, Ari Korhonen, Jussi Nikander, Otto Seppälä, and Panu Silvasti. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, 3(2):267 – 288, 2004.

[41] Andres Moreno, Niko Myller, Erkki Sutinen, and Mordechai Ben-Ari. Visualizing programs with Jeliot 3. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 373 – 376, Gallipoli (Lecce), Italy, May 2004.

[42] Brad A. Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing*, 1:97–123, 1990.

[43] Marc Najork. Web-based algorithm animation. In *Proceedings of the 38th conference on Design automation, DAC'01*, pages 506–511, Las Vegas, Nevada, United States, 2001. ACM Press.

[44] Thomas Naps, Myles McNally, and Scott Grissom. Realizing XML driven algorithm visualization. In *Proceedings of the Fourth Program Visualization Workshop*, 2006.

[45] Thomas L. Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodgers, and J. Ángel Velázquez-Iturbide. Exploring the role of visualization and engagement in computer science education. *SIGCSE Bulletin*, 35(2):131–152, June 2003.

[46] TL Naps. JHAVÉ: Supporting Algorithm Visualization. *Computer Graphics and Applications, IEEE*, 25(5):49–55, 2005.

[47] Blaine A. Price, Ronald M. Baecker, and Ian S. Small. A principled taxonomy of software visualization. *Journal of Visual Languages and Computing*, 4(3):211–266, 1993.

[48] G. Roman and K. C. Cox. A taxonomy of program visualization systems. *IEEE Computers*, pages 97–123, December 1993.

[49] Rockford J. Ross and Michael T. Grinder. Hypertextbooks: Animated, active learning, comprehensive teaching and learning resource for the web. In Stephan Diehl, editor, *Software Visualization: International Seminar*, pages 269–283, Dagstuhl, Germany, 2002. Springer.

[50] Guido Rößling. ANIMAL-FARM*: An Extensible Framework for Algorithm Visualization*. Phd thesis, University of Siegen, Germany, 2002. Available online at `http://www.ub.uni-siegen.de/epub/diss/roessling.htm`.

[51] Guido Rößling and Bernd Freisleben. Program visualization using ANIMALSCRIPT. In *Proceedings of the First Program Visualization Workshop*, pages 41–52, University of Joensuu, Finland, 2000.

[52] Guido Rößling and Bernd Freisleben. ANIMAL: A system for supporting multiple roles in algorithm animation. *Journal of Visual Languages and Computing*, 13(3):341–354, 2002.

[53] Guido Rößling, Felix Gliesche, Thomas Jajeh, and Thomas Widjaja. Enhanced expressiveness in scripting using AnimalScript 2. In *Proceedings of the Third Program Visualization Workshop*, pages 10–17, The University of Warwick, UK, July 2004.

[54] Guido Rößling and Gina Häussage. Towards tool-independent interaction support. In *Proceedings of the Third Program Visualization Workshop*, pages 110–117, The University of Warwick, UK, July 2004.

[55] Guido Rößling, Thomas Naps, Mark S. Hall, Ville Karavirta, Andreas Kerren, Charles Leska, Andrés Moreno, Rainer Oechsle, Susan H. Rodger, Jaime Urquiza-Fuentes, and J. Ángel Velázquez-Iturbide. Merging interactive visualizations with hypertextbooks and course management. *SIGCSE Bulletin*, 38(4):166–181, 2006.

[56] Clifford A. Shaffer, Matthew Cooper, and Stephen H. Edwards. Algorithm visualization: a report on the state of the field. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 150–154, New York, NY, USA, 2007. ACM Press.

[57] J. T. Stasko. TANGO: A framework and system for algorithm animation. *IEEE Computer*, 23(9):27–39, 1990.

[58] J. T. Stasko and E. Kraemer. A methodology for building application-specific visualizations of parallel programs. *Journal of Parallel and Distributed Computing*, 18(2):258–264, 1993.

[59] John T. Stasko. Jsamba - java version of the SAMBA animation program. Available online at `http://www.cc.gatech.edu/gvu/softviz/algoanim/jsamba/`.

[60] John T. Stasko. Using direct manipulation to build algorithm animations by demonstration. In *Proceedings of Conference on Human Factors and Computing Systems*, pages 307–314, New Orleans, Louisiana, USA, 1991. ACM, New York.

[61] John T. Stasko. Using student-built algorithm animations as learning aids. In *The Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education*, pages 25–29, San Jose, CA, USA, 1997. ACM Press, New York.

[62] John T. Stasko and Charles Patterson. Understanding and characterizing software visualization systems. In *The Proceedings of the IEEE Workshop on Visual Languages*, pages 3–10, Seattle, WA, USA, 1992.

[63] Linda Stern, Harald Søndergaard, and Lee Naish. A strategy for managing content complexity in algorithm animation. In *Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education, ITiCSE'99*, pages 127–130, Kracow, Poland, 1999. ACM Press.

[64] Paul Vickers and James Alty. CAITLIN: A musical program auralisation tool to assist novice programmers with debugging. In *Proceedings of the Third International Conference on Auditory Display, ICAD'96*, pages 17–24, Palo Alto, California, United States, 1996.

[65] W3C. Scalable Vector Graphics (SVG) 1.0 specification. `http://www.w3.org/TR/SVG`, September 2001.

P1

Ville Karavirta, Ari Korhonen, Lauri Malmi, and Kimmo Stålnacke. MatrixPro - A tool for on-the-fly demonstration of data structures and algorithms. In *Proceedings of the Third Program Visualization Workshop*, pages 26–33, The University of Warwick, UK, July 2004.

P2

Thomas Naps, Guido Rößling, Peter Brusilovsky, John English, Duane Jarc, Ville Karavirta, Charles Leska, Myles McNally, Andrés Moreno, Rockford J. Ross, and Jaime Urquiza-Fuentes. Development of xml-based tools to support user interaction with algorithm visualization. *SIGCSE Bulletin*, 37(4):123–138, December 2005.

P3

Petri Ihantola, Ville Karavirta, Ari Korhonen, and Jussi Nikander. Taxonomy of effortless creation of algorithm visualizations. In *ICER'05: Proceedings of the 2005 international workshop on Computing education research*, pages 123–133, New York, NY, USA, 2005. ACM Press.

P4

Ville Karavirta, Ari Korhonen, and Lauri Malmi. Taxonomy of algorithm animation languages. In *SoftVis '06: Proceedings of the 2006 ACM symposium on Software visualization*, pages 77–85, New York, NY, USA, September 2006. ACM Press.

P5

Ville Karavirta. Integrating algorithm animation systems. In *Proceedings of the Fourth Program Visualization Workshop (PVW 2006)*, volume 178 of *Electronic Notes in Theoretical Computer Science*, pages 79–87, 4 July 2007.