Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 224 (2009) 97–103
www.elsevier.com/locate/entcs

Work in Progress: Automatic Generation of Algorithm Animations for Lecture Slides

Otto Seppälä[1]

Department of Computer Science and Engineering
Helsinki University of Technology
Espoo, Finland

Ville Karavirta[2]

Department of Computer Science and Engineering
Helsinki University of Technology
Espoo, Finland

Abstract

Algorithm visualizations have not been widely adopted in teaching. One possible reason for this is that visualizations are often developed as standalone systems which can be difficult to integrate into lectures. Recently XML based formats for the two major presentation tools have been introduced. We present a method and a prototype implementation which allows creation of algorithm animations in the ODF format. This allows integrating the animation seamlessly within the lecture material.

Keywords: Algorithm animation, animation, lecture, lecture slides, presentation

1 Introduction

Algorithm visualizations have not been widely adopted in teaching. The problem of integration of visualization in self-study material has been studied in the context of HTML-based hypertextbooks [12]. HTML allows integrating animations as Java applets, Flash animations and videos to name a few.

Integrating visualizations in lecture material has been studied a lot less. In a survey study done by the ITiCSE 2002 Working Group [10] 79% of the educators listed the "time it takes to adapt visualizations to teaching approach and/or course content" as a substantial impediment for adopting new visualizations to be used on a course. These same difficulties were also found in a recent international survey

[1] Email: oseppala@cs.hut.fi
[2] Email: vkaravir@cs.hut.fi

doi:10.1016/j.entcs.2008.12.053

by Lahtinen et al.[8]. Ben-Bassat Levy and Ben-Ari argue that tool developers often don't invest enough in how pedagogical software can be embedded into a curriculum [1]. While the most commonly used presentation tools (Powerpoint, OpenOffice Impress) allow embedding e.g. video into the lecture slides, the lecturer is deprived of the possibility of adapting the animation to the specific lecture case.

For lecture use, the development in AA systems has focused on systems that can be used to give presentations. For example, Alvis [5], ANIMAL [11] and MatrixPro [7] all have features to support use on lectures. However, in most cases, animations created with the AA tools cannot be embedded into the lecture material since they are implemented as independent applications. Instead, they require the educator to switch between a number of programs during classroom presentation. In some algorithm visualization tools it might be possible to overcome this limitation by designing the lecture slides inside the algorithm visualization system. In most cases this is neither desired nor in any way feasible. The fact is that the presentation tools are much more sophisticated than the AA authoring tools.

Interaction provided by the algorithm animation has a major impact on the learning results [6]. However, in lecture situations, it has been shown that using animations or lecture slides are equally effective [9]. Thus, automatic generation of lecture slides is a valid approach to promote algorithm animation in teaching.

In this on-going research, we explore the idea of lowering the barrier of integration. Our approach is to allow the teacher to create algorithm animations for the presentation tool he/she is already using on lectures. This allows the lecturer to insert the slides seamlessly within the other lecture material used in the same lecture. The recent introduction of XML formats for Microsoft PowerPoint and OpenOffice Impress has made it feasible to generate such formats with external tools. In this paper, we present a proof-of-concept of such a tool that can be used to produce animations of the Kruskal's algorithm in the form of Open Office Impress presentations.

2 Motivation

Think of a typical scenario on a data structures and algorithms course where a teacher is to give a lecture on e.g. Kruskal's algorithm. In order to use an existing visualizations for this algorithm, a suitable presentation must first be found. Here, the teacher can already run into problems. Recent research shows that the existing algorithm visualization are typically of poor quality and concentrate on the simpler algorithms [13].

Visualizations often concentrate on certain features of the algorithm and ignore others. This is often required to limit visual complexity. As research suggests, the pedagogical style of the visualization might not match the style of the teacher [1]. The final choice of the visualization is likely to be a compromise which addresses most points the teacher wants to address during the lecture. In some cases the slides have to address limitations of the visualization as the visualizations themselves cannot be altered.

Some features of the visualization can also affect the structure of the slides. At least the visualization often has to be included as a "chunk" to avoid constantly switching between the presentation software and the visualization tool.

A whole another problem is that visualizations are often topic-specific tools. For a complete course, the teacher might have to use visualizations provided by a large number of developers. Naturally, this results in visualizations that do not share a uniform look and can cause unnecessary confusion for teacher as well as the students. Additional confusion can also be caused by (even subtle) variations in terminology.

According to results of an international survey, the classroom set-ups vary a lot, with the most typical set-up being a class with a computer and a ceiling-mounted projector [10]. Thus, the teacher has to make preparations before the lecture. Applets should be downloaded and local webpages created to account for problems in network connections. In some cases the visualization software has to be installed, which might even be impossible in some lecture hall setups. The safest alternative is often to use a personal laptop for giving the presentations. However, PowerPoint or PDF slides typically work with whatever computer is available.

2.1 Why Now?

Until now, the tools for implementing the generation of slides have been missing. One reason for this is that in the past, the presentation tools have used closed proprietary formats. The development of open XML formats for the presentation tools has made implementing such tools much easier. Both of the most popular presentation tools, Microsoft PowerPoint and OpenOffice Impress, have open XML formats, Office Open XML and Open Document Format, respectively. These languages make it possible to generate presentations for these tools with reasonable effort.

In addition, Extensible Stylesheet Language Transformations (XSLT) [2], a language designed to transform XML documents to other formats, makes transforming XML documents into the formats used by presentation tools simpler. The output of XSLT can be another XML format, HTML, or text. There are many tools available to do the XML transformations using XSLT. The most well-known XSLT processors for Java are Xalan and Saxon. This provides a simple way to implement transformations between languages.

3 Technical Description of Proof-of-concept Implementation

Our proof-of-concept implementation generates OpenOffice Impress slides visualizing the behavior of the Kruskal's minimum spanning tree algorithm. Kruskal's algorithm was chosen because it seems for some reason to be rather rarely visualized. The algorithm also uses an interesting data structure - the union-find structure usually implemented as a forest of father-linked trees.

Part of the idea behind the proof-of-concept was to use existing tools to automate parts of the process including graph layout software as well as XSL transformation packages. Figure 1 describes the architecture of our implementation. This whole process is ran using an Ant script. The first step is to execute a Java program. It first generates suitable input data for the Kruskal's algorithm and then executes the algorithm. For each state of the algorithm, the program creates graph descriptions in a format readable by the GraphViz [4] graph layout package. The graph descriptions hold information on colors, line widths, labels etc. Our program also creates the father linked trees that form the union-find structure used by Kruskal's algorithm. These are also laid out by GraphViz. The third and last part of the visualization is the sorted list of graph edges. The Java program outputs a visualization of this list directly in OpenOffice Impress format. In addition to the visualization, the program outputs supplementary information as slide notes that are later combined to the visualization steps.

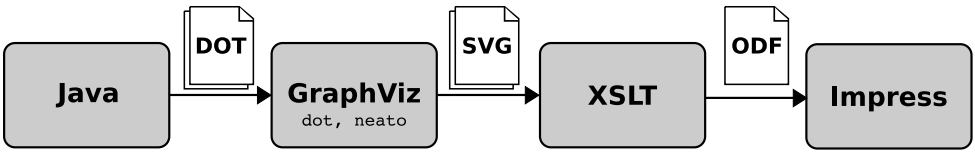

Fig. 1. Architecture of our solution

The next step in creating the visualization is running the GraphViz programs dot and neato on the input data. GraphViz is an open-source software package intended for graph drawing. The system includes several different layout algorithms suitable for different kinds of graphs. Dot generates hierarchical layouts and is used to create the union-find graphics. Neato generates spring-based layouts and is used to layout the graph. In addition, the system supports several different output formats like png, jpeg, ps, pdf, and svg. In our program, we decided to use Scalable Vector Graphics (SVG) [14] since it is easy to process in the later steps. SVG is an XML language targeted for describing graphics.

The last step converts the SVG-files into corresponding Impress snippets, which are combined with the parts created directly by the Java code. This is done with XSLT which transforms the multiple SVG-files into one Open Document Format (ODF) presentation. The XSL-stylesheet is divided into two parts that handle different parts of the input. The first one generates the main ODF-document creating the needed styles and pages. It combines each slide from the multiple input files (tree, graph, notes, and edge list). For the SVG-files (used for trees and graphs) we have another XSL-stylesheet. This stylesheet transforms the graphical elements of the SVG into ODF. It also takes care of things like coordinate system transformations, scaling etc.

The output of the process is in Open Document Format [3]. ODF is an open, XML-based file format for office applications. The format specifies an XML structure for text documents, spreadsheets, and presentations. In an ODF presentation, each slide includes the slide contents as well as the notes attached to that slide. Listing 1 gives an example of the graphical primitives in one ODF slide. The primi-

```
<draw:g xmlns:svg="http://www.w3.org/2000/svg">
  <draw:polygon draw:style-name="fillwhitestrokewhite" svg:x="1cm" svg:y="1cm" draw:points="
      0,236 0,0 203,0 203,236 0,236" svg:height="11.8cm" svg:width="10.15cm" svg:viewBox="0
      0 203 236">
    <text:p/>
  </draw:polygon>
  <draw:ellipse svg:x="6.7cm" svg:y="3.875cm" svg:width="1.4cm" svg:height="1.45cm"
      draw:style-name="fillredstrokered"/>
  <draw:frame draw:style-name="gr9" draw:text-style-name="P1" draw:layer="layout" svg:x="7.4
      cm" svg:y="4.6cm">
    <draw:text-box>
      <text:p text:style-name="P1">Joensuu</text:p>
    </draw:text-box>
  </draw:frame>
  <draw:ellipse svg:x="8.2cm" svg:y="7.425cm" svg:width="1.4cm" svg:height="1.45cm"
      draw:style-name="fillorangestrokeorange"/>
  <draw:frame draw:style-name="gr9" draw:text-style-name="P1" draw:layer="layout" svg:x="8.9
      cm" svg:y="8.15cm">
    <draw:text-box>
      <text:p text:style-name="P1">Oulu</text:p>
    </draw:text-box>
  </draw:frame>
  <draw:line svg:x1="7.7cm" svg:y1="5.25cm" svg:x2="8.6cm" svg:y2="7.5cm" draw:style-name="
      fillnonestrokeblack"/>
</draw:g>
```

Listing 1. Example of ODF graphical primitives.

tives are the same as in, for example, SVG. However, as can be seen from the listing, the attributes used are from several different namespaces. [3]

Figure 2 shows one slide in a generated example of the Kruskal's algorithm when opened in OpenOffice Impress. On the left, one can see the additional slides in the presentation. On the notes page, the presentation includes questions that the instructor can ask the students, as well as answers to the questions.

4 Conclusions

In this work, we have introduced an idea to develop tools to easily create slides for the presentation tools. Such tools would allow instructors to integrate algorithm visualizations into the tools they use on lectures instead of using separate visualization systems. We feel that this type of integration might be a solution to the problem of algorithm visualizations not used as widely as the AV research community hopes.

Our vision is to have automatic tools that output algorithm animations as presentation slides. This approach saves the teacher from switching between an algorithm visualization system and the presentation software. Compared with using, say, applets to visualize algorithms on lecture, the advantage of the slides is that the lecturer can edit them and adapt them to his/her other learning material. This can be, for example, changing terminology, changing colors, translating it to e.g. German, or adding explanations. In addition, the lecturer can alter the parameters for the tool creating the example, allowing the example to be tuned for the current audience.

[3] This caused some major problems in the implementation, since the ODF specification does not clearly state which attributes should be in which namespace. Especially specifying styles was difficult, partly due to the fact that Impress gives no error messages when using wrong namespaces, it just ignores the attributes. Another problem was with the positioning of the graphical primitives. Although the attributes are from the svg-namespace, the coordinates used are different.

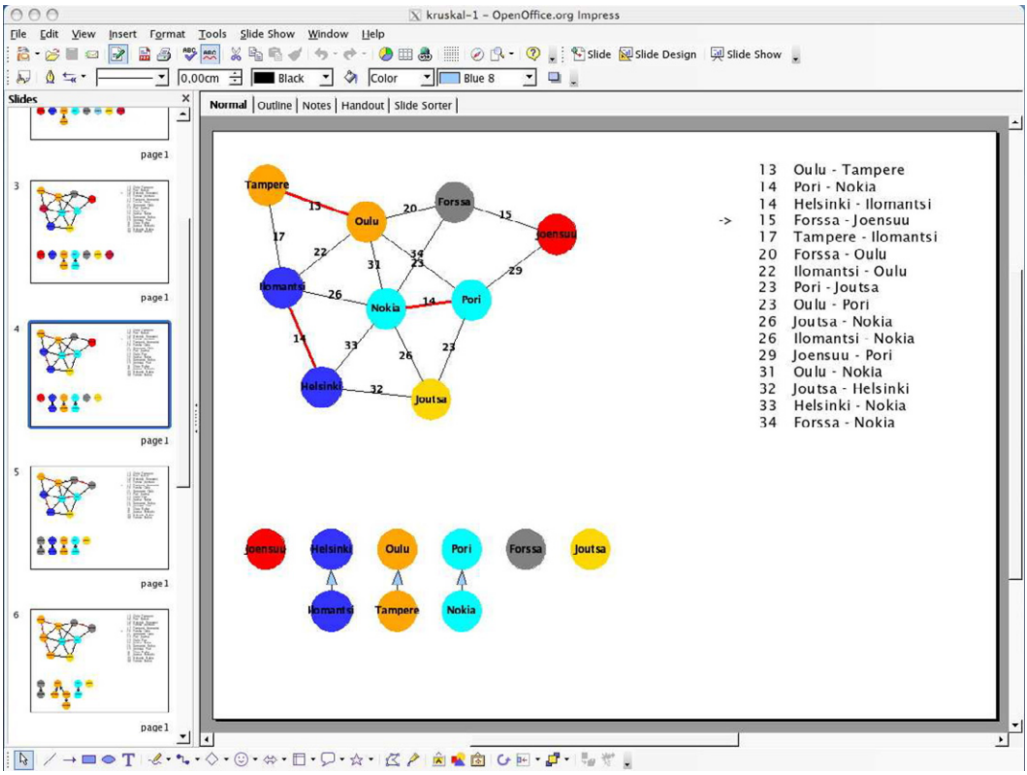

Fig. 2. A generated example in OpenOffice Impress.

Besides the visualization, automatically generated lecture material can be accompanied with dynamic documentation in the form of notes for the lecturer to use when showing the animation. These can be things to point out, questions for the audience, and such. This makes it easier for the teacher to make the lecture more interactive.

All the previous can be done in the presentation tool that the teacher is more likely to be familiar with than an algorithm animation system. Although we are algorithm animation system developers ourselves, we have to admit that the AA authoring tools are not of the same high quality as the presentation tools.

The limitations of the current proof-of-concept implementation are obvious. The implementation is for a single algorithm, the process requires several software packages to be installed, and the output is only for Open Office Impress. However, from Open Office Impress, the presentations can be exported as Flash or HTML to be easily added to web pages. In addition, they can be saved in Microsoft PowerPoint and PDF formats.

In the future, creating more such generators could be beneficial. In addition, allowing ODF export from some of the existing AV systems is an interesting direction. A very recently introduced project of OpenOffice.org, ODFDOM [4], aims to provide a Java programming API to create and modify ODF documents. The project looks

[4] http://wiki.services.openoffice.org/wiki/ODFDOM

very promising and could be the most straight-forward way to generate such documents. Furthermore, we also see that an online service for generating presentations of different topics might be a lightweight alternative which might prove popular among CS educators. Especially if the service could create slides for both Microsoft PowerPoint and OpenOffice Impress. This also eliminates the need to install the required software packages.

References

[1] Ben-Bassat Levy, R. and M. Ben-Ari, *We work so hard and they don't use it: acceptance of software tools by teachers*, in: *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education* (2007), pp. 246–250.

[2] Clark, J. (editor), *XSL Transformations (XSLT) 1.0 specification*, W3C Recommendation, World Wide Web Consortium (1999).

[3] Durusau, P. and M. Brauer, *Open document format for office applications (opendocument) v1.0 (second edition)*, Oasis committee specification, OASIS (2006).

[4] Ellson, J., E. Gansner, L. Koutsofios, N. S. C. and G. Woodhull, *Graphviz open source graph drawing tools*, Lecture Notes in Computer Science **2265/2002** (2002), pp. 594–597.

[5] Hundhausen, C. D. and S. A. Douglas, *Low-fidelity algorithm visualization*, Journal of Visual Languages and Computing **13** (2002), pp. 449–470.

[6] Hundhausen, C. D., S. A. Douglas and J. T. Stasko, *A meta-study of algorithm visualization effectiveness*, Journal of Visual Languages and Computing **13** (2002), pp. 259–290.

[7] Karavirta, V., A. Korhonen, L. Malmi and K. Stålnacke, *MatrixPro – A tool for on-the-fly demonstration of data structures and algorithms*, in: *Proceedings of the Third Program Visualization Workshop*, The University of Warwick, UK, 2004, pp. 26–33.

[8] Lahtinen, E., H.-M. Järvinen and S. Melakoski-Vistbacka, *Targeting program visualizations*, in: *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education* (2007), pp. 256–260.

[9] Lawrence, A., A. Badre and J. T. Stasko, *Empirically evaluating the use of animations to teach algorithms*, in: *Proceedings of the 1994 IEEE Symposium on Visual Languages, St. Louis, MO*, 1994, pp. 48–54.

[10] Naps, T. L., G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodgers and J. Ángel Velázquez-Iturbide, *Exploring the role of visualization and engagement in computer science education*, SIGCSE Bulletin **35** (2003), pp. 131–152.

[11] Rößling, G. and B. Freisleben, *ANIMAL: A system for supporting multiple roles in algorithm animation*, Journal of Visual Languages and Computing **13** (2002), pp. 341–354.

[12] Rößling, G., T. Naps, M. S. Hall, V. Karavirta, A. Kerren, C. Leska, A. Moreno, R. Oechsle, S. H. Rodger, J. Urquiza-Fuentes and J. A. Velázquez-Iturbide, *Merging interactive visualizations with hypertextbooks and course management*, SIGCSE Bulletin **38** (2006), pp. 166–181.

[13] Shaffer, C. A., M. Cooper and S. 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* (2007), pp. 150–154.

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