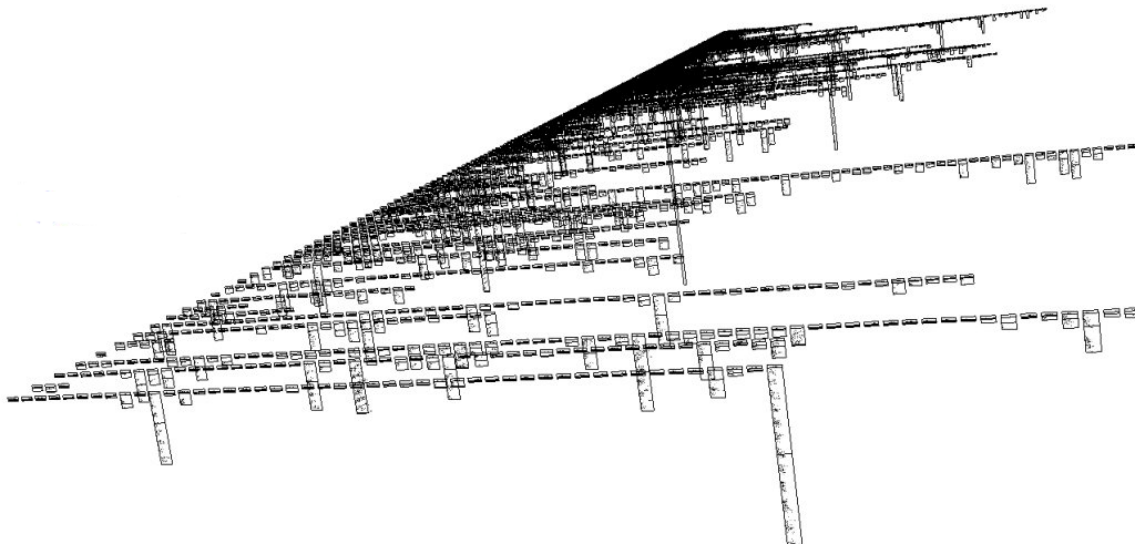


# Evaluating Extensible 3D (X3D) Graphics For Use in Software Visualisation

by

Craig Anslow



A thesis  
submitted to the Victoria University of Wellington  
in fulfilment of the  
requirements for the degree of  
Master of Science  
in Computer Science.

Victoria University of Wellington  
2008



## **Abstract**

3D web software visualisation has always been expensive, special purpose, and hard to program. Most of the technologies used require large amounts of scripting, are not reliable on all platforms, are binary formats, or no longer maintained. We can make end-user web software visualisation of object-oriented programs cheap, portable, and easy by using Extensible (X3D) 3D Graphics, which is a new open standard. In this thesis we outline our experience with X3D and discuss the suitability of X3D as an output format for software visualisation.





# Acknowledgments

I would first just like to say that my Mum and Dad have been great support over the past few years in completing this degree and without them the struggle would have been much worse. Thanks to Frank for accommodating me. My weekends would not have been as much fun or balanced without my lovely dogs, Amber-Rose and Ruby-Anne.

Thanks to my close friend Alex Potanin for just being Alex; interrupting me at the most opportune moment and asking me to do things when least expected. I know I can always rely on you. So thanks for letting me come and discuss my ideas and problems with you over the course of my degree. Even if the discussion always ended up talking about you, I knew you were always interested in what I had to say about you.

To James Noble, Robert Biddle, and Stuart Marshall, thank you very much for supervising me. Without your leadership and guidance I do not know what I would have done. It has been great to have three completely different personalities to be able to inspire me, give me advice, pointers, and direction on many aspects of my project. Your help has been much appreciated and will not be forgotten.

Through the Elvis Software Design Research Group I have had many a fine time and met some really great people including: Mike McGavin, Donald Gordon, Kirk Jackson, Pippin Barr, Rilla Khaled, Angela Martin, Matt Duignan, Chris Andreae, Stephen Nelson, Darren Willis, Jennifer Ferreira, Pia Holland, Annie Luxton, Hayden Smith, Brenda Chawner, Joerg Evermann, Robbie Morrison, Wm Leler, Gary Haggard, Craig Chambers, Tim Wright, David Pearce, and Ewan Tempero. There were also some students who had to put up with me being their office mates for many years which I know is not an easy task to do, so thanks to: Feng Lu, Nick Jamieson, Will Smart, Dean Pemberton, Jennifer Ferreira, and Rashina Hoda.

Thank you to Mike McGavin, John Fouhy, and Jonathan Aumonier-Ward, (JAWs) for introducing me to a relatively new cool sport in Wellington, Ultimate Frisbee. Without playing and thinking about Ultimate constantly I would have had no reason to stay active. I have also met some other outstanding individuals from this community and played in some really interesting teams.

During my degree I had the opportunity to attend a Web3D Consortium workshop where I had the opportunity to meet some of the pioneers in the X3D community including Don Brutzman, Leonard Daly, Alan Hudson, Tony Parisi, Chris Thorne, and Rita Turkowski. Thank you all for the wonderful advice, support and encouragement. Thanks to John Coady for helping with X3D programming.

In the journey of this thesis I have had the opportunity to meet and email various people along the way and discuss with them my research and future plans. So thanks to the following people: Liz Richardson, Liz Medford, Lyndon Hawk, Michael Richmond, Nadyne Mielke, Bruce Campbell, Ben Stephenson, Alan Blackwell, Neville Churcher, Andy Cockburn, Tim Bell, Tim Dwyer, Peter Eades, Steve Reiss, Gary Sevitsky, Ronald Bourret, Lianjiang Fu, Paul H J Kelly, Dennis Mancl, Margaret-Anne Storey, Tamara Munzner, Daniel Keim, Matt Ward, John Stasko, Ben Shneiderman, Robert Kosara, Malcolm Munro, Paul McIntosh, Michele Lanza, Orla Greevy, Dale Bent, Rasmus Koch Hansen, Rachel Ryan, Guillaume Langelier, Tilman Giese, and Keith Andrews. I can't remember everyone, so sorry if I left any one off the list.

Doing this thesis would not have been possible without some levels of funding. Thanks to Rachel Irving for employing me for a couple of years in the marketing group at Victoria University of Wellington. I had a splendid time working with a great bunch of people. Thanks to my managers at Unisys, Grant Nielsen and Tim Hogan for employing me on two vastly different projects. Thanks to Andrew Whelan for listening and providing support.

Thanks to the School of Mathematics, Statistics, and Computer Science for funding me with a scholarship and a small research grant. Thanks to the Faculty of Science for funding me with a small research grant as well. Thanks to Liz Richardson and Shona De Sain for part time employment in the Faculty of Science. Thanks to my supervisor James Noble for some funding through the DSTOOLS grant. Finally, thanks to Harris Stratex Networks for a Masters scholarship.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	2
1.2	Contributions . . . . .	3
1.3	Outline . . . . .	3
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	Information Visualisation . . . . .	6
2.1.1	2D Versus 3D Visualisation . . . . .	7
2.1.2	3D Information Visualisation Techniques . . . . .	10
2.2	Software Visualisation . . . . .	15
2.2.1	Early 3D Software Visualisation . . . . .	20
2.2.2	Source Code Visualisations . . . . .	20
2.2.3	Object-Oriented Metrics . . . . .	21
2.2.4	UML Diagrams . . . . .	23
2.2.5	Dynamic Visualisations . . . . .	23
2.3	VARE: Visualisation Architecture for REuse . . . . .	25
2.3.1	Programming Mapping Visualisation . . . . .	25
2.3.2	Architecture . . . . .	26
2.3.3	Execution Traces . . . . .	26
2.3.4	Visualisation Tools . . . . .	29
2.3.5	3D Technologies . . . . .	31
2.4	Summary . . . . .	34
<b>3</b>	<b>Exploring X3D</b>	<b>37</b>
3.1	Overview . . . . .	38
3.2	Specification . . . . .	39
3.2.1	Basic Example . . . . .	39
3.2.2	Encodings . . . . .	39
3.2.3	Profiles . . . . .	39
3.3	X3D Nodes . . . . .	43
3.3.1	Geometry . . . . .	43

3.3.2	Grouping . . . . .	44
3.3.3	Viewing and Navigation . . . . .	47
3.3.4	Lighting, Environment, and Sound . . . . .	47
3.3.5	Animation and Interaction . . . . .	48
3.3.6	Prototypes . . . . .	49
3.4	Resources . . . . .	49
3.4.1	X3D Browsers and Plug-ins . . . . .	49
3.4.2	Content Authoring and Editing Tools . . . . .	51
3.4.3	Export and Translator Tools . . . . .	55
3.5	Discussion . . . . .	55
3.5.1	Improvements to the Specification . . . . .	57
3.5.2	Future of X3D on the Web . . . . .	58
<b>4</b>	<b>X3D Software Visualisations</b>	<b>61</b>
4.1	VARE-3D . . . . .	62
4.1.1	Architecture . . . . .	62
4.1.2	Implementation . . . . .	63
4.1.3	Visualisation Transformation . . . . .	64
4.1.4	Discussion . . . . .	70
4.2	Algorithm Animations . . . . .	72
4.2.1	Shortest Path . . . . .	73
4.2.2	Heapsort . . . . .	77
4.2.3	Elementary Sorting . . . . .	80
4.2.4	Discussion . . . . .	85
4.3	UML Diagrams . . . . .	89
4.3.1	Class Diagrams . . . . .	89
4.3.2	Package Diagrams . . . . .	93
4.3.3	Sequence Diagrams . . . . .	93
4.3.4	Discussion . . . . .	95
4.4	Documentation-Related Visualisations . . . . .	97
4.4.1	Source Code Visualisation . . . . .	98
4.4.2	API Javadoc Visualisation . . . . .	98
4.4.3	Structured Video Visualisation . . . . .	98
4.4.4	Discussion . . . . .	101
4.5	Execution Trace Visualisations . . . . .	103
4.5.1	All Elements From an Execution Trace . . . . .	103
4.5.2	3D Compound Shapes . . . . .	105
4.5.3	3D Metaphors . . . . .	110
4.5.4	Discussion . . . . .	115

4.6	Summary . . . . .	116
<b>5</b>	<b>Software Visualisation Media Evaluation Framework</b>	<b>117</b>
5.1	Differences With Earlier Evaluation Model . . . . .	118
5.2	Scope* . . . . .	121
5.2.1	Requirements* . . . . .	121
5.2.2	Design* . . . . .	122
5.2.3	Method* . . . . .	123
5.2.4	Performance . . . . .	123
5.3	Form* . . . . .	123
5.3.1	Graphical Capability . . . . .	123
5.3.2	Presentation* . . . . .	125
5.3.3	Visualisation Techniques* . . . . .	126
5.4	Interaction* . . . . .	127
5.4.1	User Controls* . . . . .	127
5.4.2	User Navigation* . . . . .	127
5.4.3	User Tasks* . . . . .	128
5.5	Discussion . . . . .	128
<b>6</b>	<b>Evaluation of X3D</b>	<b>131</b>
6.1	Scope . . . . .	132
6.1.1	Requirements . . . . .	132
6.1.2	Design . . . . .	133
6.1.3	Method . . . . .	134
6.1.4	Performance . . . . .	137
6.2	Form . . . . .	138
6.2.1	Graphical Capability . . . . .	138
6.2.2	Presentation . . . . .	141
6.2.3	Visualisation Techniques . . . . .	143
6.3	Interaction . . . . .	146
6.3.1	User Controls . . . . .	146
6.3.2	User Navigation . . . . .	148
6.3.3	User Tasks . . . . .	151
6.4	Discussion . . . . .	152
6.4.1	Advantages of X3D . . . . .	153
6.4.2	Disadvantages of X3D . . . . .	154
6.4.3	Potential Improvements for X3D . . . . .	156

<b>7</b>	<b>Conclusions</b>	<b>157</b>
7.1	Contributions . . . . .	158
7.2	Future Work . . . . .	158
<b>A</b>	<b>Evaluation of X3D Summary</b>	<b>161</b>

# List of Figures

2.1	Harry Beck’s 1933 original London Underground map. Reproduced by kind permission of London’s Transport Museum ©Transport for London. . . . .	8
2.2	Tree Map — KDirStat. . . . .	12
2.3	Cone Tree [220]. . . . .	13
2.4	Hyperbolic 3D Viewer [166]. . . . .	14
2.5	Information Cube [213]. . . . .	14
2.6	Information Landscape. . . . .	16
2.7	Botanical Tree [122]. . . . .	17
2.8	Areas of software visualisation [237]. . . . .	18
2.9	Jinsight software visualisation system. . . . .	19
2.10	Object-Oriented Metrics in VRML [57]. . . . .	22
2.11	X3D-UML [163]. . . . .	24
2.12	BLOOM, Spiral views of the stack (sampled during execution) [203].	24
2.13	The Programming Mapping Visualisation Model [221, 234, 239]. . .	26
2.14	The VARE architecture [160]. . . . .	27
2.15	XTE — example execution trace of a Java program. . . . .	28
2.16	Spider, a test drive of the Java Calendar component [157, 158]. . .	29
2.17	VET displaying a sequence and an association diagram of the same data from an execution trace [162]. . . . .	30
2.18	Blur SVG visualisation tool [74]. . . . .	32
3.1	X3D content that encodes a sphere with an image of the world used as texture and the text “Hello world!” [41]. . . . .	40
3.2	An image of the world earth-topo.png [41]. . . . .	41
3.3	X3D Sample Document — Hello World. . . . .	41
3.4	The X3D nodes represented as a tree structure from the Hello World sample document. . . . .	42
3.5	X3D Baseline Profiles [265]. . . . .	42
3.6	Examples of 3D geometry definitions in X3D [41]. . . . .	45

3.7	Examples of extrusion geometry definitions in X3D [41]. . . . .	46
3.8	X3D Routing Event Model [41]. . . . .	49
3.9	X3D Browser Software Architecture [265]. . . . .	50
3.10	Octaga X3D Player Example — animating whale shark, courtesy of Marko Steffensen from Octaga ( <a href="http://www.octaga.com">http://www.octaga.com</a> ). . . . .	52
3.11	BS Contact VRML/X3D Player Example — Golden Gate Bridge, courtesy of the National Institute of Standards and Technology (NIST) ( <a href="http://www.nist.gov">http://www.nist.gov</a> ). . . . .	53
3.12	Flux Player Example — person running and then jumping, courtesy of Yilmaz Degirmenci from the US Naval Postgraduate School ( <a href="http://www.nps.edu">http://www.nps.edu</a> ). . . . .	54
3.13	X3D Editing Tools. . . . .	56
4.1	VARE-3D architecture. . . . .	63
4.2	VARE-3D — web interface for execution trace X3D software visualisations. . . . .	65
4.3	Basic X3D Execution Trace Visualisation — in this simple X3D software visualisation 10,000 events have been transformed from the XTE execution trace of the Eclipse IDE Java application by XSLT. Blue spheres represent object creation events, green boxes method calls, white cones method returns and end of an object, field accesses cyan cylinders, and field modifications as pink cylinders. . . . .	66
4.4	Example Eclipse XTE execution trace. . . . .	68
4.5	XTE execution trace XSL stylesheet. . . . .	69
4.6	UML Class Diagram XSL prototype stylesheet. . . . .	71
4.7	X3D Shortest Path Algorithm Animation — shortest path from vertex A to all other vertices. Hand crafted in X3D following the strategy of Brown and Najork [33, 168, 169]. . . . .	73
4.8	Shortest Path Algorithm — initial 2D state. . . . .	74
4.9	X3D Shortest Path Algorithm Animation — different stages of the algorithm. . . . .	76
4.10	X3D Heapsort Algorithm Animation — completed algorithm animation with united sticks like array and heap views. . . . .	78
4.11	X3D Heapsort Algorithm Animation — different stages of the algorithm. The anomalous node placements in parts 4.11(b), 4.11(c), and 4.11(d), are the results of the elements being in motion. . . . .	81
4.12	X3D Elementary Sorting Algorithm Animation — chips view. . . . .	82
4.13	X3D Elementary Sorting Algorithm Animation — blocks view. . . . .	84
4.14	X3D Elementary Sorting Algorithm Animation — blocks behind view. . . . .	84



4.15	X3D routing event model from the insertion sort algorithm animation.	86
4.16	X3D Cityscape UML class diagram — 100 classes . . . . .	90
4.17	X3D Eclipse UML class diagram — 4536 classes, showing the flexible scale of the visualisation technique. . . . .	91
4.18	UML Class diagram prototype declaration and an example prototype instance from the CityScape Java program. . . . .	92
4.19	X3D Cityscape UML package diagram . . . . .	94
4.20	X3D Cityscape UML sequence diagram . . . . .	96
4.21	Source code visualisation with a UML like class diagram and C++ source code. . . . .	99
4.22	API Javadoc Visualisation with the Cityscape UML class diagram. .	100
4.23	Structured Video Visualisation with a UML like class diagram. . . .	102
4.24	All elements from an execution trace represented as a sequence of spheres. . . . .	104
4.25	All elements from an execution trace represented as different shapes on different lines. . . . .	104
4.26	All elements from an execution trace represented as a different shapes and on the same line. . . . .	105
4.27	All elements from the CityScape execution trace represented as a cube like shape. . . . .	106
4.28	XSLT code to parse an object creation event from our execution traces	107
4.29	JavaScript layout function that calculates x, y, and z coordinates for a node. . . . .	108
4.30	Elements from the CityScape cube visualisation. . . . .	109
4.31	10,000 elements from the Eclipse execution trace. . . . .	110
4.32	100,000 elements from the Eclipse execution trace. . . . .	111
4.33	CityScape 3D Spiral. . . . .	111
4.34	Eclipse 3D Spiral. . . . .	112
4.35	All elements from an execution trace represented as an information landscape. . . . .	113
4.36	All elements from an execution trace represented as an information cube. . . . .	114
6.1	Editing X3D file in a text editor. . . . .	136
6.2	Creating an X3D software visualisation from the command line using the Apache Xalan XSLT processor. . . . .	137
6.3	Eclipse execution trace — 50,000 events. . . . .	144
6.4	Eclipse UML package diagram — 257 packages . . . . .	144
6.5	Information Cube — wireframe rendering. . . . .	147

6.6 Algorithm animation user controls. . . . . 148

6.7 X3D Navigation — navigation options by right clicking in the Oc-  
taga Player. . . . . 149

6.8 Information Landscape — showing available viewpoints. . . . . 150

# Chapter 1

## Introduction

Software visualisation is defined as the use of the crafts of typography, graphic design, animation, and cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of computer software. John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price [237].

Software visualisation has existed in many forms for a long time. Software visualisation has been used for many different purposes including algorithm animation, education, program structure, program execution, program debugging, memory leak analysis, software engineering, reverse engineering, software maintenance, and software reuse [68]. Software visualisation can be thought of as the application of information visualisation techniques in software engineering and can show the structure of software, runtime behaviour, and representation of source code. Extensible 3D (X3D) Graphics [265] is a new light weight open standard for web based 3D graphics. **The goal of this thesis is to evaluate X3D for use in software visualisation** which could potentially help with all of these purposes.

Frederick P. Brooks Jr. claims in his seminal paper *No Silver Bullet: Essence and Accidents of Software Engineering* [32] that software is invisible and unvisualisable. The reality of object-oriented software today is that it is quite complex and continually growing in complexity in a variety of ways [22]. If we want software visualisation to be a practical tool for developers, we have to develop visualisation techniques and systems that are designed to handle the complexity of tomorrow's systems, not yesterdays [208]. The large scale software systems of today makes software visualisation harder now than before.

## 1.1 Motivation

As part of an ongoing research project we have designed a Visualisation Architecture for REuse (VARE) [160] for evaluating software components over the web. The design supports visualising components in multiple languages and configurations (e.g. complete programs or just code fragments). Our architecture requires tools to develop and deliver easy to learn and easy to use visualisations from execution traces.

Our research group has created a prototype tool that can produce 2D Scalable Vector Graphics (SVG) visualisations over the web from our execution traces [74, 75]. Ware et al. [263] demonstrate that displaying object-oriented software in three dimensions instead of two can make it easier for users to understand the data. We want to create 3D software visualisations for software reuse, software maintenance, and reverse engineering that can be viewed over the web and has good XML support which is an important requirement for VARE [160]. We require a language and a tool that can produce these 3D web software visualisations.

The tools and visualisations that we are interested in building are aimed at software developers who have standard personal computers (CRT or LCD monitor, keyboard and mouse, web browser) and an Internet connection. The computer requirements include approximately 512MB-2GB of memory, a dedicated graphics / video card, and no special purpose hardware for input or output. Our visualisation architecture project [160] needs a web-based technology that meets these requirements.

Most web-based technologies that are used to create 3D software visualisations over the web use either Java3D, Java applets, or VRML. Java3D requires libraries to be installed onto users platforms and require a viewer to be created. Java applets have security restrictions, are slow to load when embedded in a web page, and sometimes have problems when working on different platforms. VRML is no longer maintained, hard to extend, and hard to inter-operate with other languages. Java3D is a binary format while VRML is a text based format that uses the old OpenInventor style syntax. All of these technologies lack support for XML.

This thesis evaluates using X3D [265] – the new open standard for web 3D graphics – for use in software visualisation. This thesis addresses the following research questions: How good is X3D for use in software visualisation? How well does X3D support the software visualisation pipeline, by creating visualisations from execution traces? How well does X3D support our software visualisation architecture?

## 1.2 Contributions

The key contribution of this thesis is evaluating how suitable X3D is for software visualisation. In more detail:

- **X3D Software Visualisation Case Studies (§4)** - this work replicates and discusses the implementation of a range of software visualisations in X3D including algorithm animations, UML diagrams, documentation-related visualisations, and execution trace visualisations.
- **VARE-3D (§4.1)** - this effort provides a prototype tool which can produce software visualisations in X3D from XML execution traces over the web.
- **Software Visualisation Media Evaluation Framework (§5)** - this work creates a framework for evaluating software visualisation media or graphics technologies for use in software visualisation based on a previous evaluation model.
- **Evaluating X3D For Use in Software Visualisation (§6)** - this work applies our software visualisation media evaluation framework to evaluate how good X3D is for software visualisation.

## 1.3 Outline

The remainder of this thesis is organised as follows:

- Chapter 2 contains related work in the areas of information visualisation, software visualisation, and describes our visualisation architecture project, VARE.
- Chapter 3 explores X3D giving an overview, describing the specification, listing some X3D resources such as browsers and tools, and discussing improvements to the specification and the future of X3D on the web.
- Chapter 4 presents our web-based prototype tool for producing X3D software visualisations, then describes our X3D software visualisation case studies. Our case studies include algorithm animations, UML diagrams, documentation-related visualisations, and execution trace visualisations.
- Chapter 5 describes a framework for evaluating software visualisation media or graphics technologies in software visualisation.

- Chapter 6 then applies our software visualisation media evaluation framework with our X3D software visualisations and our use of the X3D specification.
- Chapter 7 summarises our findings, by discussing the advantages, disadvantages and potential improvements of X3D for software visualisation. We also list our contributions of this thesis, and address future work.

# Chapter 2

## Background

### Contents

---

<b>2.1</b>	<b>Information Visualisation . . . . .</b>	<b>6</b>
2.1.1	2D Versus 3D Visualisation . . . . .	7
2.1.2	3D Information Visualisation Techniques . . . . .	10
<b>2.2</b>	<b>Software Visualisation . . . . .</b>	<b>15</b>
2.2.1	Early 3D Software Visualisation . . . . .	20
2.2.2	Source Code Visualisations . . . . .	20
2.2.3	Object-Oriented Metrics . . . . .	21
2.2.4	UML Diagrams . . . . .	23
2.2.5	Dynamic Visualisations . . . . .	23
<b>2.3</b>	<b>VARE: Visualisation Architecture for REuse . . . . .</b>	<b>25</b>
2.3.1	Programming Mapping Visualisation . . . . .	25
2.3.2	Architecture . . . . .	26
2.3.3	Execution Traces . . . . .	26
2.3.4	Visualisation Tools . . . . .	29
2.3.5	3D Technologies . . . . .	31
<b>2.4</b>	<b>Summary . . . . .</b>	<b>34</b>

---

In a recent survey [133, 134] based on questionnaires completed by 111 researchers from software maintenance, re-engineering and reverse engineering, 40% found software visualisation absolutely necessary for their work and another 42% found it important but not critical. 7% think that it is at least relevant and 6% that they can do without but it is nice to have. Only 1% believe software visualisation is not an issue at all. Finally, 4% did not answer the question. From the same survey relatively few people consider software visualisation their primary research (11%) or at least a substantial part of their research (18%). Many people are doing software visualisation research every now and then (20%), however most people are primarily using or integrating existing software visualisation tools developed by others (33%).

We believe that applying visualisation techniques to software will help to assist developers to understand software. Understanding the shape of existing software is a crucial first step to understanding how software systems have been built [22]. Developers face the task of understanding software when they want to reuse, maintain, reverse engineer, or re-engineer a piece of software. Visualising the source code and run-time of software can give a greater insight into the structure and behaviour of software, and will be able to help developers in these tasks [237].

The main reasons for wanting to reuse and maintain software are to save on time, effort, and costs in both development and maintenance of quality software [193]. For software reuse the developer will not have to implement a new solution to an old problem. For software maintenance the refactoring of code and fixing bugs will be reduced. The reason for reverse engineering is to break a piece of software down to understand it to either build a copy of the software or to improve the software [39]. Re-engineering is the subsequent modification of the software once it has been reverse engineered, usually to add new functionality or to correct errors.

Next we describe the nature of information visualisation, and techniques for visualising information. We then describe the nature of *software visualisation*, and look at various software visualisation systems that have used 3D graphics. Finally, in the last section, we describe the software visualisation architecture being developed by our research group.

## 2.1 Information Visualisation

Card et. al [45] describe information visualisation as the use of computer-supported, interactive, visual representations of abstract data to amplify cognition. Even after producing a visual representation, the following issues must be addressed: ex-



ploration, navigation, and interpolation of the data [46]. Several overviews on information visualisation exist [23, 45, 50, 101, 231, 232, 258].

The theory of the visual display of quantitative information [253] consists of principles that generate design options and that guide choices among the design options. Tufte [253] describes graphical excellence of quantitative information, as the well designed presentation of interesting data - a matter of substance, of statistics, and of design. Graphical excellence consists of complex ideas communicated with clarity, precision, and efficiency. This results in a visualisation displaying the greatest number of ideas, in the shortest time and in the smallest space possible.

An early example of graphical excellence is the original London Underground map designed by Harry Beck in 1933, see Figure 2.1. Typically most people will use the map as a visualisation tool for planning a journey from one station to another and a feasible route between them. People may memorise their route by colour or the direction of the lines involved and any intermediate stations. The internal model created by memorising a route is known as a cognitive map [232]. Beck based the map on electrical circuit diagrams which does not reflect the geography of the city above. The revolutionary design with minor modifications and additions still remains today. Further information on the history of the London Underground Map is located in Garland [88].

Ben Shneiderman [228] created a visual design guideline called the visual information seeking mantra which says show an overview first, then zoom and filter, and finally show details-on-demand. He then proposed a task by data type taxonomy which has seven data types (1-, 2-, 3-dimensional data, temporal and multi-dimensional data, and tree and network data) and seven tasks which a user can perform (overview, zoom, filter, details-on-demand, relate, history, and extract). This mantra is one of the very few methodical guidelines for designing information visualisations and it is the most widely cited [64]. There are, however, other user task heuristics [4, 5, 142, 254, 278, 279] but they are not as useful for evaluating usability, focus on low level tasks, or are domain specific. Each of these other user task heuristics have components which overlap Shneiderman's [228] mantra.

### 2.1.1 2D Versus 3D Visualisation

Visualisation in 2D has been heavily explored [45]. What benefits 3D representation is over 2D still remains to be answered for information and software visualisation. The goal of this thesis is not to determine if 3D software visualisations are better than 2D software visualisations. Instead we are going to evaluate a 3D technology – X3D, an open standard for web 3D graphics – for use in software visualisation.



Figure 2.1: Harry Beck's 1933 original London Underground map. Reproduced by kind permission of London's Transport Museum ©Transport for London.

We now give a brief overview of 2D versus 3D for information visualisation.

A 3D world gives users an extra degree of freedom, but sometimes this leads to confusion and complications with understanding and navigating a visualisation compared to 2D visualisations [50]. Ware et al. [16, 260, 261] compared 2D and various different 3D techniques for comprehending 3D graphs using fish-tank (no flat screen) virtual reality [214, 244, 259]. Fish-tank virtual reality uses expensive special purpose hardware such as head-mounted 3D displays, mounted projectors, 3D input devices, and tracking devices. The 3D techniques considerably outperformed the 2D displays, errors were reduced when using 3D, and being able to move or rotate the graph were key features of the 3D techniques. They have since abandoned this mode of viewing because graphics technology today have remarkably improved and cheaper since their earlier work. They now use 3D graphics technologies that don't require head mounted displays or head tracking devices.

Hubona et al. [109] claim understanding of a 3D structure over 2D improves when a user can manipulate the structure. Another study by Irani and Ware et al. [110, 111, 112] found that information will be easier to read when 2D UML information structures are mapped into connected structures built with 3D primitives called *geon diagrams*. Geon diagrams are made out of cones, spheres, cylinders, and boxes.

Risden et al. [218] compared 2D and 3D visualisations of web content and the results indicated that there were no reliable differences in overall user performance or satisfaction with the visualisations. Hicks et al. [106] conducted some empirical evaluations on 2D and 3D representations which presented customer behaviour information on telecommunication usage. The results indicated a performance advantage for the 2D display compared with both the 3D representations. Accordingly the merits about 3D are mixed as there are a number of conflicting research studies.

Moving from 2D to 3D user interfaces will not necessarily enhance a user's performance through natural support for spatial memory, i.e. a user remembering where objects are located in space. Robertson et al.'s [219] 3D Data Mountain allowed users to arrange thumbnail images of web pages on an inclined plane along the Z dimension. The results showed that the Data Mountain improved retrieval times and reduced error rates in comparison to Internet Explorer's Favourites. Tavanti and Lind [245] conducted experiments using 2D and 3D interfaces for remembering alphanumeric characters. The results of the experiments confirmed that the 3D display better supported the task of correctly locating the characters on the depth level.

Some contrasting similar studies on spatial memory were carried out by Cock-

burn et al. [58, 60, 61, 62] where they replicated both of the two previous examples by Robertson et al. [219] and Tavanti and Lind [245]. The results showed no significant difference between 2D and 3D when using computer-supported systems, but 2D performed better when using actual physical systems.

Shneiderman [229] gives evidence that disorientation and occlusions of information are major weaknesses of 3D visualisations. Shneiderman claims that 3D information visualisation interfaces have the potential for novel social, scientific, and commercial applications if designers go beyond mimicking 3D reality. In order to achieve good 3D information visualisations, the following features must be supported: rapid situation awareness through effective overviews, reduced number of actions to accomplish tasks, and prompt and meaningful feedback for user actions. Bowman et al. [29, 30] provide an excellent introduction to 3D user interfaces.

Chen [50] claims that current empirical studies suggest that increasing an interface from 2D to 3D is unlikely to be enough to boost the task performance, unless additional functions are provided so that users can have greater control of objects within a 3D visualisation.

Nielsen [172, 173] claims that “3D is never going to make it big-time in user interfaces and especially web user interfaces until we get true 3D control devices”. This is because 3D images are displayed on a 2D computer screens and are controlled through 2D input devices, the mouse and keyboard. Spence [231] also claims for 3D to be useful one has to be able to move the data in a visualisation.

To summarise the 2D versus 3D visualisation debate, one should not assume that a 3D visualisation is automatically superior to a 2D visualisation. Answering the question “which is better, 3D or 2D?” is difficult, if not impossible [84]. When deciding on using 3D there should be clear important subtasks for which 3D is clearly beneficial over 2D [258]. Simply increasing a visualisation from 2D to 3D is unlikely to improve task performance unless extra and greater controls of 3D data in the visualisation are created.

### 2.1.2 3D Information Visualisation Techniques

We now discuss some 3D information visualisation techniques and metaphors. We are particularly interested in 3D information visualisation techniques that could be used for creating X3D software visualisations. A broader range of 3D information visualisation techniques can be found elsewhere [51, 274].

One concept for information visualisation is Focus + Context [45] which requires the user having a view of the whole data available (context), while pursuing detailed analysis of a part of it (focus). A distorted view, a focus + context tech-

nique, is created by applying a transformation function to an undistorted image. Creating a distorted view magnification in one place occurs at the expense of compression in another. Leung and Apperley [143] describe a taxonomy and a unified theory of graphical distortion-oriented presentation techniques for spatial problems.

The Polyfocal Display [117] is a one-dimensional form that has a transformation and magnification function where the highest peak is the focus of display. Fish-Eye views [87] are a way to generate a small display of a large structure in a wide angle or fish-eye lens where distance and level of detail can be defined. Sarkar and Brown [224] extended Furna's fish-eye concept by creating the graphical fish-eye view.

The Bifocal Lens [233] contains one part of information in detail while the other parts in limited detail. Carpendale et al. [46] have created a 3D Bifocal Fish-Eye view. The Perspective Wall [149] is a conceptual descendant of the bifocal display where the main difference is that of the out-of-focus regions. The Table Lens [198] displays information based on a table format where a specified number of cells in a table are the main focus.

The Hyperbolic Browser [136], for visualising large hierarchies, distorts space so when a user moves around in the display a virtual camera follows and constantly redisplay the view onto Euclidean space on the screen. Essentially the information in the hyperbolic browser is stretched and squeezed as the user moves position or moves objects.

The traditional way of representing tree structures such as XML documents is to have a directed graph with a root node at the top then child nodes below connected to parent nodes by links. The major problem with this traditional tree representation is that it occupies a great deal of space, usually to the extent that for more than three levels and more than 50 nodes at the third level, the tree is too large for an effective presentation on a display screen [231]. We now discuss some improved techniques for displaying trees.

Shneiderman created the Tree Map [116, 227] which displays tree structures as a rectangular 2D display in a space-filling manner, where 100% of the designated display space is utilised. The Tree Map was designed to gain a better representation of the utilisation of storage space on a hard disk as viewed from the perspective of a multiple level directory of subdirectories and files. Figure 2.2 shows a practical use of Tree Maps with an image of KDirStat<sup>1</sup> which is a graphical disk usage utility for KDE that is based on SequoiaView [256, 268] an implementation of a Tree Map.

Bladh et al. [28] created a 3D tree map application, Step Tree. They conducted

---

<sup>1</sup><http://kdirstat.sourceforge.net>

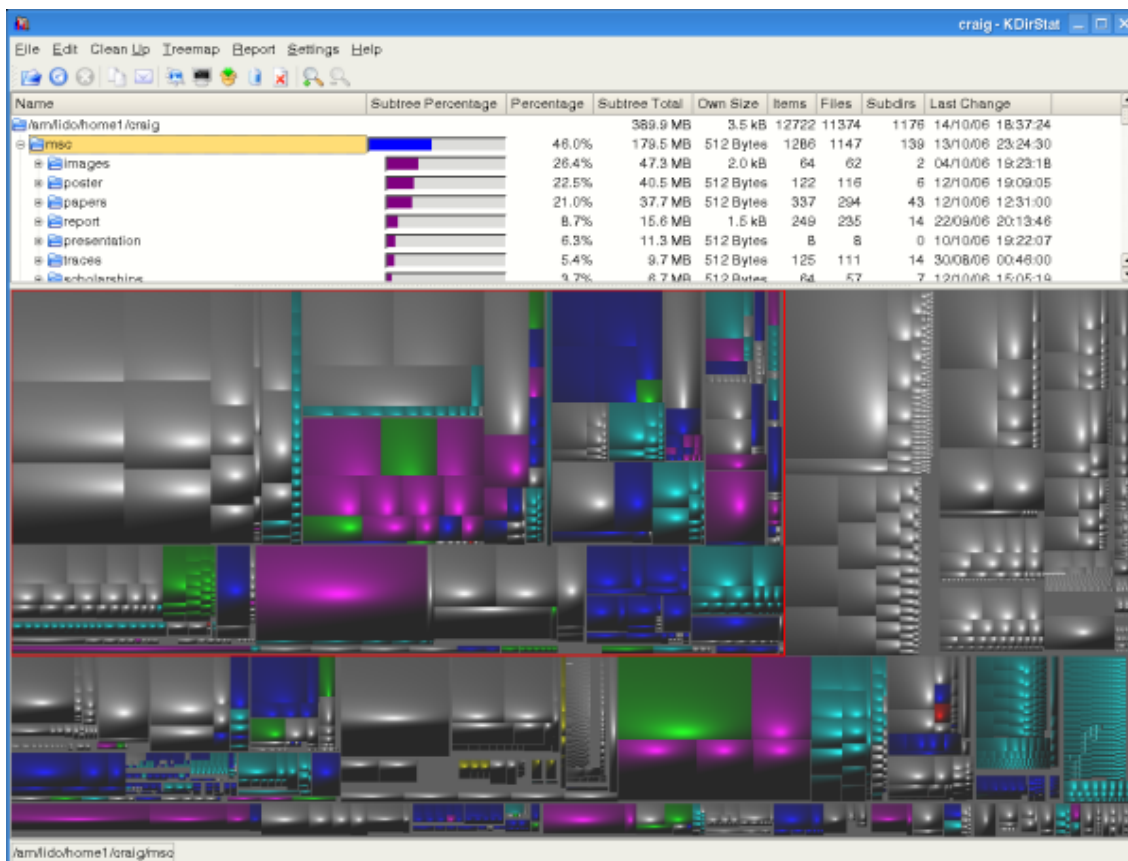


Figure 2.2: Tree Map — KDirStat.



usability studies and found that users perform significantly better on tasks related to interpreting structural relationships when using Step Tree as opposed to Tree Maps. The display of hierarchical depth was a clear advantage over Tree Maps.

Cone Trees [220] are a technique for visualising hierarchical information structures vertically, see Figure 2.3. The hierarchy is presented in 3D to maximise the effective use of available screen space and enable visualisation of the whole structure. Cam Trees [220] are an alternative layout which are horizontally oriented. Munzner et al. [166, 167] extended the Cone Tree layout for 3D hyperbolic space by placing children on a hemisphere around the cone mouth instead of on its perimeter, see Figure 2.4.

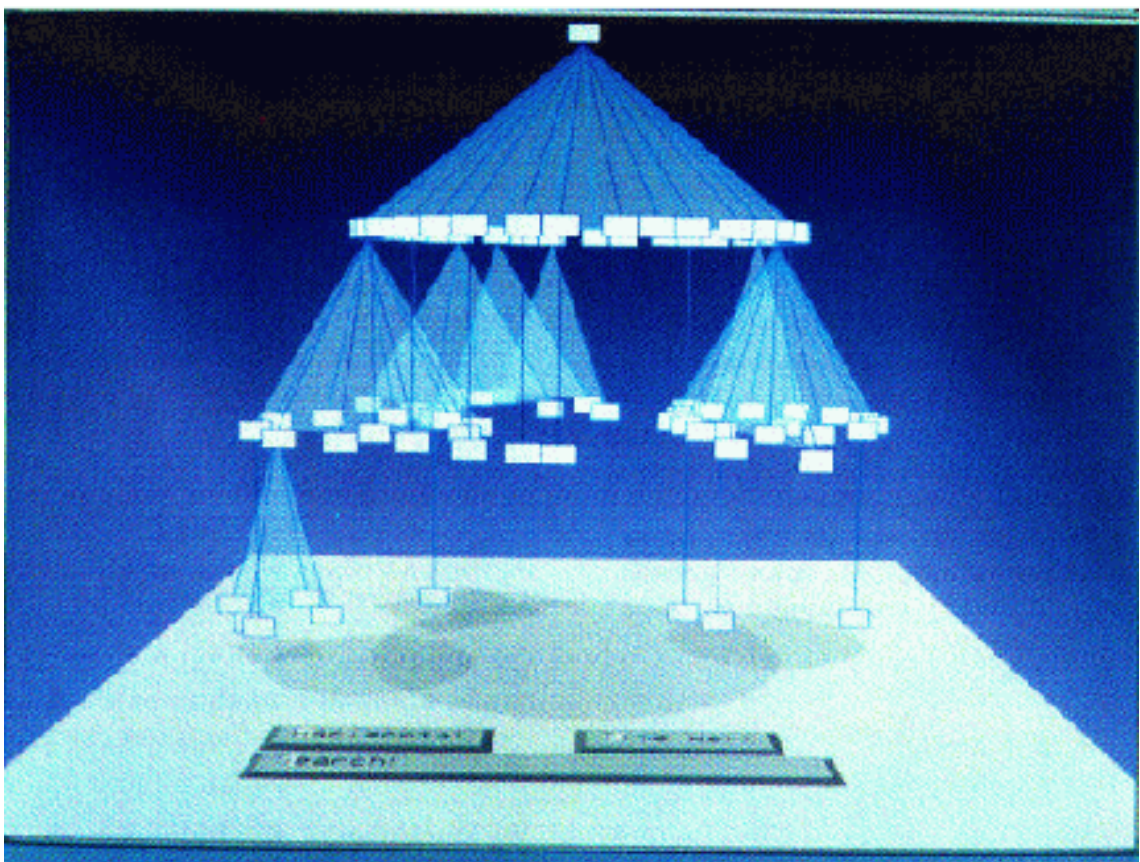


Figure 2.3: Cone Tree [220].

Cockburn and MacKenzie [59] conducted some empirical studies to demonstrate the performance and effectiveness of users browsing documents through a Cone Tree interface versus a Explorer-like tree normal browser. The normal tree browser performed better for deep and shallow browsing due to the increased density of information in the Cone Tree. The subjects stated that the Cone Trees provided a better feel for the structure of the data space. Subjects also felt that the lack of experience using the Cone Tree hindered their performance and with more

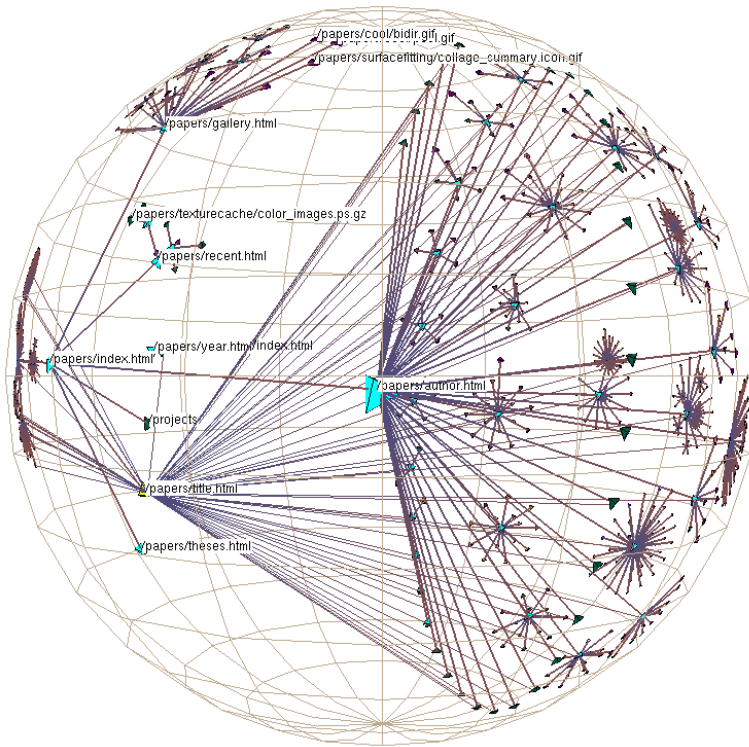


Figure 2.4: Hyperbolic 3D Viewer [166].

time different results maybe achieved.

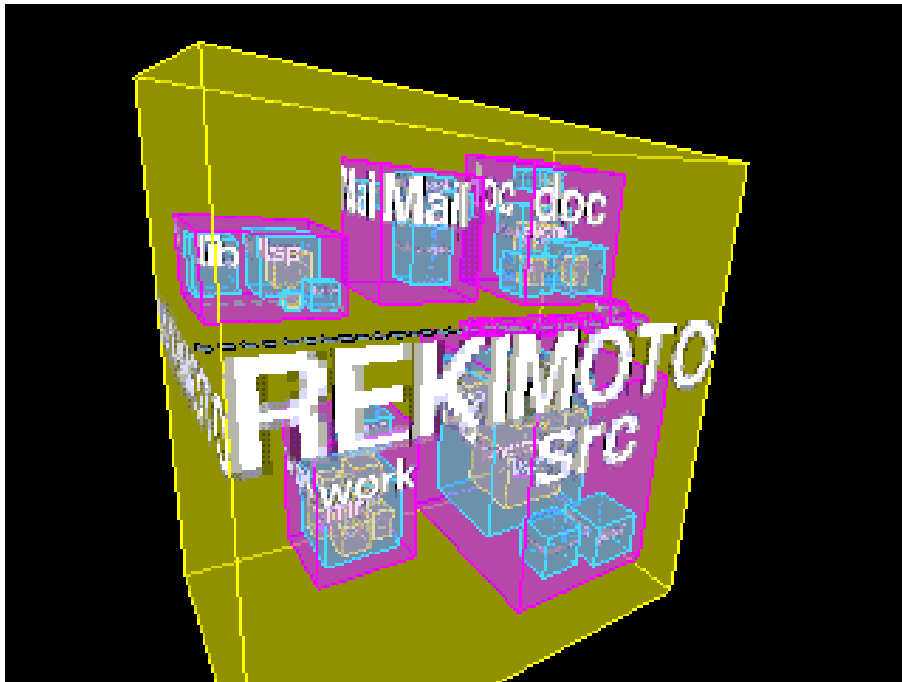


Figure 2.5: Information Cube [213].



The Information Cube [213] is a 3D visualisation technique of hierarchical information visualised as nested cubes, see Figure 2.5. The outermost cube corresponds to the top level data, while the next level data are represented as the cubes in the outer most cube. Each second level cube contains third level cubes etc. Each cube has the title it is representing on its surface. Terminal data are presented as tiles with labels on their surfaces.

The Information Landscape is a  $2\frac{1}{2}$ D visualisation technique of hierarchical information visualised on a single plane, see Figure 2.6. The information landscape has been used to visualise the Unix file system [247] and web pages [7].

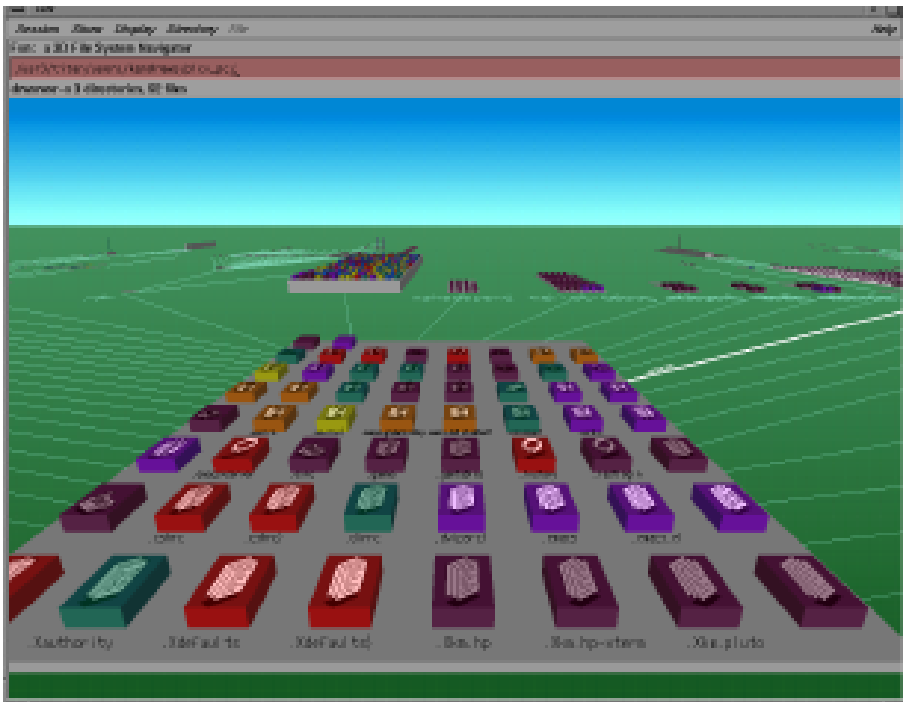
Wiss et al. [270] implemented and evaluated the Information Landscape, Cam Tree and Information Cube. They found that if it were possible to predict what the structure of the data will be, then this will help determine which information visualisation design is most applicable. They suggest one strategy to avoid predicting the structure of data is to create a system where data sets can be viewed by alternative visualisations in the same application. They then conducted an empirical study [271] comparing the three information visualisation designs. The results indicated that the subjects were significantly faster with the Information Landscape when compared with both other visualisations. The Cam Tree was significantly faster than the Information Cube. They found that local and global overview and custom navigation are important factors when creating 3D user interfaces.

The Botanical Tree [122], see Figure 2.7, is a visually intriguing way to visualise huge hierarchies. Even though the visualisation is nice to look at it is not clear that it adds further information than that of other tree layout visualisations. Jarke van Wijk [255] questions whether these kind of visualisations are good or not. He makes an attempt at determining how the value of visualisation can be assessed. He concludes that there is no single answer, but that it depends on the point of view one adopts.

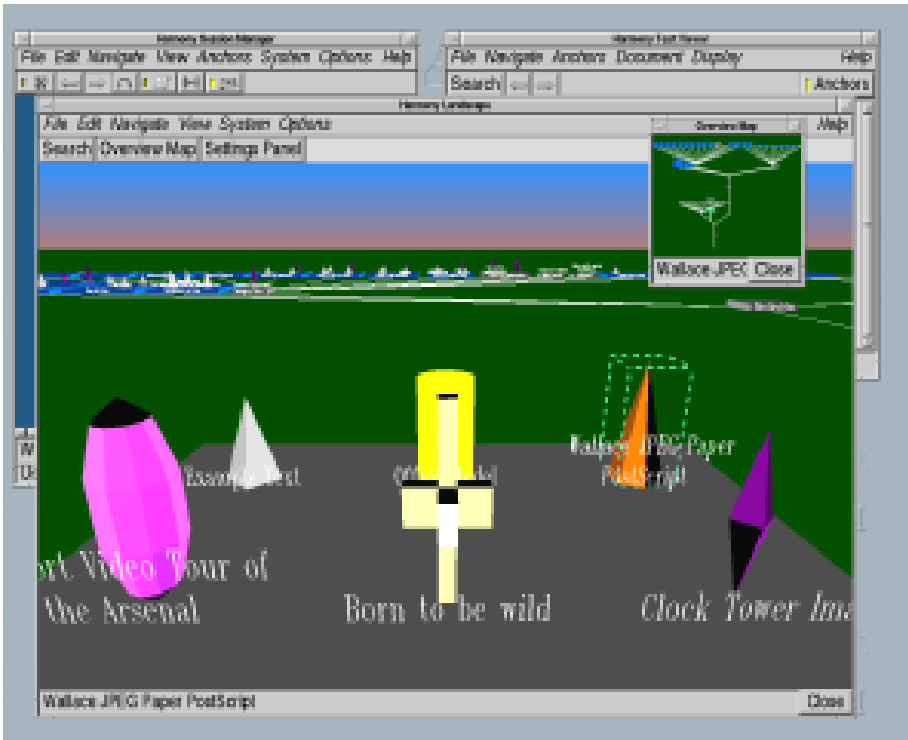
We are interested in applying these tree-like visualisation techniques for visualising software. We now discuss what software visualisation is and look at systems that use 3D visualisation techniques.

## 2.2 Software Visualisation

Software visualisation is the application of information visualisation in software engineering and can show the structure of software, runtime behaviour, and representation of source code. Software visualisation is essentially situated at the intersection of information visualisation, software engineering, human computer



(a) 3D File System Navigator [247] — Unix file system.



(b) Harmony Browser [7] — web pages.

Figure 2.6: Information Landscape.

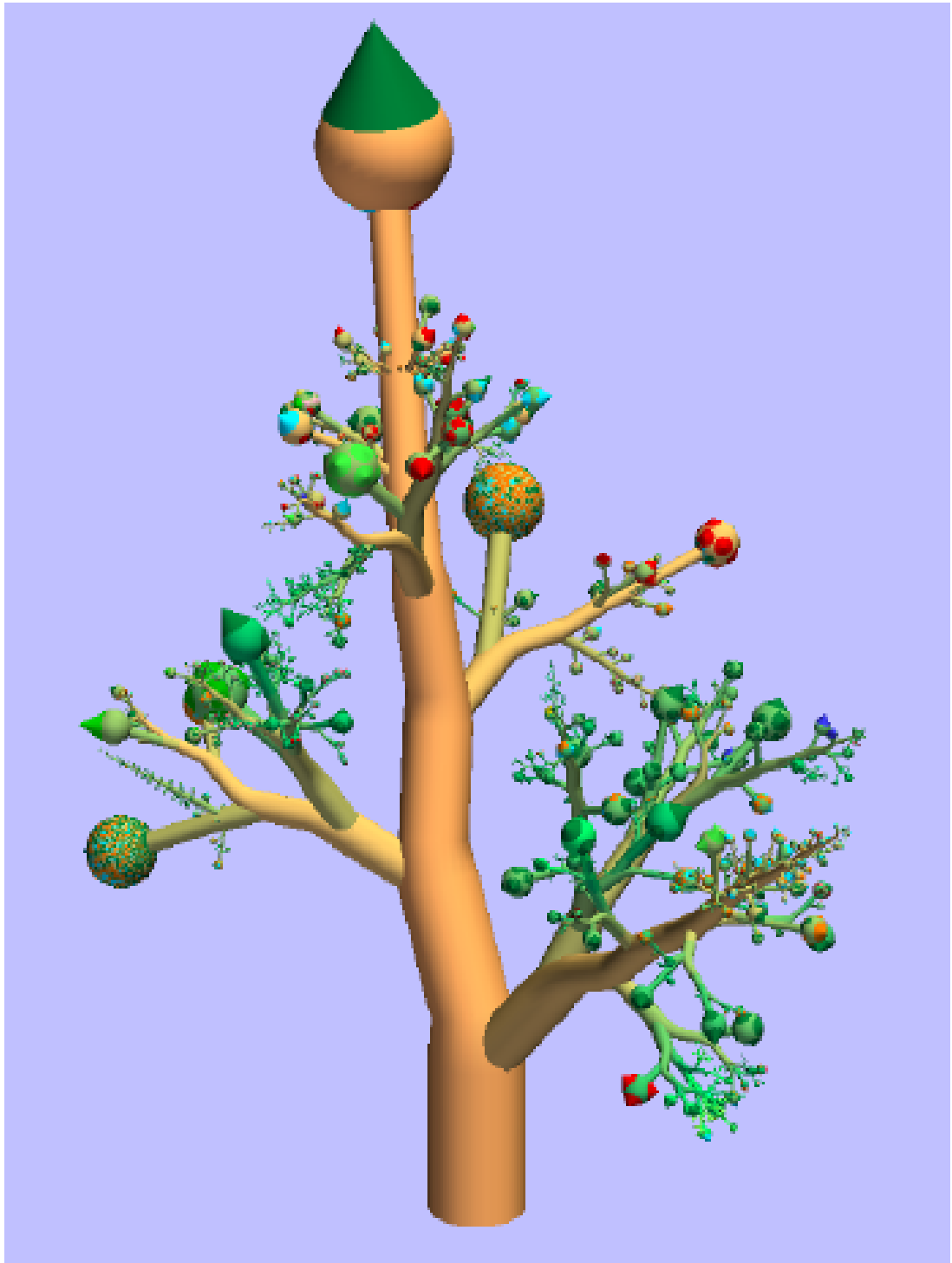


Figure 2.7: Botanical Tree [122].

interaction, graphics, and cognitive psychology [154]. Software Visualisation is comprised of the following sub-fields algorithm visualisation, visual programming, programming by demonstration, and program visualisation [237]. Figure 2.8 shows the relationships between the different types of software visualisation fields.

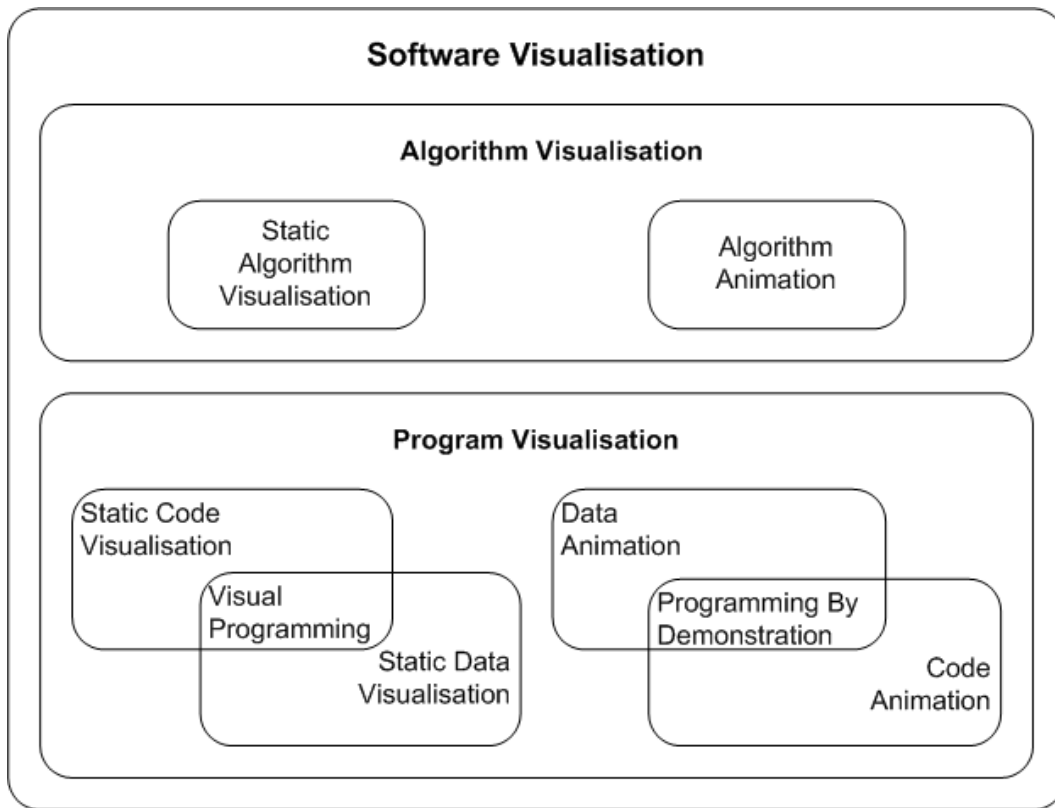


Figure 2.8: Areas of software visualisation [237].

Algorithm visualisation is the visualisation of the higher level abstractions which describe software, where as algorithm animation [119, 194] is dynamic algorithm visualisation. Algorithm animation communicates how an algorithm works by graphically displaying its fundamental operations. Brown and Hershberger [36] claim that creating effective visualisations is an art, not a science. Flowcharts are an example of static algorithm visualisations while visualising sorting algorithms are algorithm animation. Algorithm animation has been quite useful for education and for research into the design and analysis of algorithms [237]. *Sorting Out Sorting* [17, 18] was the first teaching film on algorithm animation and described nine sorting algorithms.

Program visualisation is the visualisation of actual program code or data structures in either static or dynamic form. There are various software visualisation systems that have been produced over the years and some early examples include Balsa [34, 38] (the first real-time interactive algorithm animation system), Zeus [35]

(follow up to Balsa), Tango [235, 239], Polka [238] (a follow up system to Tango), Pavane [222], and Tarraingim [174, 175, 176] (a tool for visualising Self programs produced by James Noble from our research group).

One notable recent software visualisation example is Jinsight [182, 183, 184, 185, 186, 187, 226] stemming from a wide range of work from IBM. Jinsight is a tool for visualising and analysing the execution of Java programs and is useful for performance analysis, memory leak diagnosis, debugging, or any task in which a user needs to better understand what a Java program is really doing. Figure 2.9 shows a brief overview of some of the visualisations produced by Jinsight.

Several overviews on software visualisation exist which describe some more recent software visualisation systems [43, 68, 69, 78, 115, 237, 276] and the state of the art [80, 95, 189, 267].

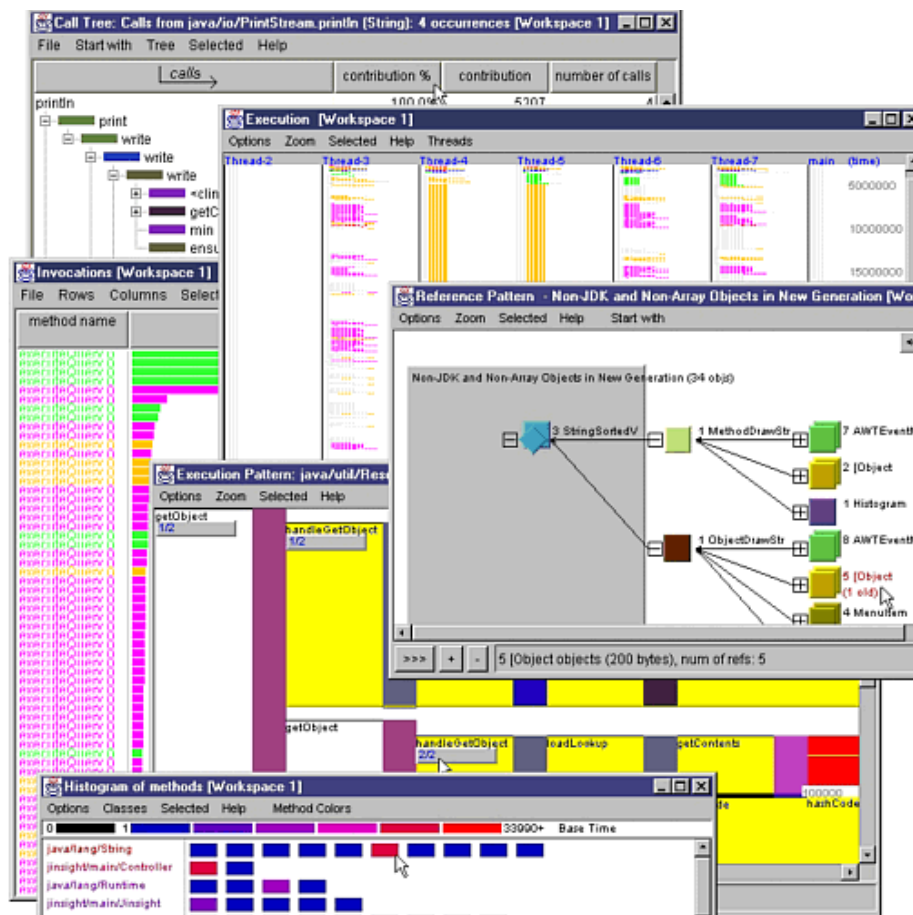


Figure 2.9: Jinsight software visualisation system.

We are mainly interested in systems that can produce 3D visualisations of software. We now look at various areas of software visualisation and focus on what 3D technologies they use and what interface they use to display the visualisations. We first begin with early work on 3D software visualisation, followed by source

code visualisations, object-oriented software metrics, UML diagrams, and finally, dynamic visualisations.

### 2.2.1 Early 3D Software Visualisation

Stasko and Wehrli [240] identified the need for three-dimensional graphics in software visualisation. They list the basic requirements for 3D computation visualisation, define three categories for characterising visualisations, and discuss their system for supporting 3D animation development by programmers.

Koike [128, 129, 130, 131] described the significance of visualising software information in three dimensional space and the problems of 2D visualisation. This work also introduced the concept of a 3D class library browser to show method inheritance. The class hierarchy was represented as a tree in the X-Y plane and methods of each class were shown in the Z axis with the same X-Y coordinates. Koike also looked at visualising large trace files in 3D of computer processes from a number of computers running in parallel and communicating with each other.

Other early research has been done in 3D for visualising Lisp programs [146], different features of a program [199, 200, 201], the layout and structuring of object oriented software in three dimensions as directed graphs (GraphVisualizer3D [84, 262, 263] and NestedVision3D [181]), web-enabled visualisations of complex SELF programs [72, 73, 105], and the Virtual Reality Modeling Language (VRML) [264] for visualising call graphs [275], design patterns [44], and software architectures [82].

Several systems have explored the use of 3D graphics for algorithm animation. Some of these systems include Pavane [63, 222], Polka3D [240], Zeus3D [37], 3D-AAPE [93], JCAT [168, 169], and Alice [65].

### 2.2.2 Source Code Visualisations

sv3D [154, 155, 156] is a framework for visualising source code and related attributes in 3D. For visualisations, the framework uses 3D metaphors based on the SeeSoft [79] pixel representation and the 3D File Maps [202]. The framework uses transparency, elevation, and special 3D manipulators to overcome occlusion. They apply Shneiderman's [228] seven high level user needs that an information visualisation application should support to the framework. There is no support for extraction and querying features, however. sv3D is implemented using Qt for the user interface and Open Inventor for the rendering components. They also show how sv3D can be combined with an information retrieval tool [273] to enrich source code searching and browsing in MS Visual Studio. Previous work of theirs

was done using immersive environments [150, 151] whereupon they created a visual language which defines a formal mapping from object-oriented languages to a visualisation in virtual reality. The language only supports syntactic and static features of a program.

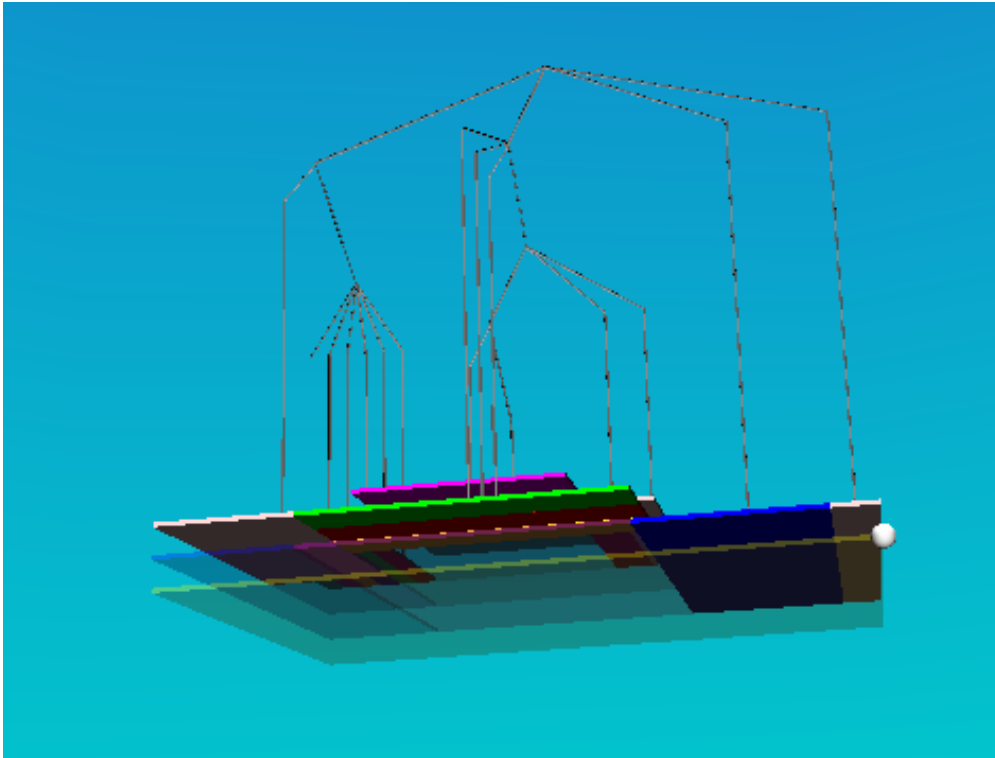
Numerous other systems have looked at visualising the static structure of software in 3D as well. J3Browser [1, 2, 3] explores Java class relations in VRML while their other tool VisMOOS [85, 86] is an Eclipse plug-in that uses Java3D. Hierarchical Net [20, 21] visualises the structure of large software systems as software landscapes. VizzAnalyzer [147] is a framework designed for reverse engineering and can create visualisations in Java3D. VizzAnalyzer also has a built-in tool Vizz3D [179], which can be used as standalone for visualising class and package interaction, program evolution, and program quality as Java3D or OpenGL visualisations. Telea et al. [246] provide an open toolkit for visualising telecommunications software for the purposes of reverse engineering using Open Inventor. CCVisu [25, 26, 27] uses a method for computing clustering layouts of software systems for which the change history is available in VRML and SVG. WhiteCoats [164] visualises the evolution of software from CVS repositories using VRML. Finally, the Rube [108, 121] framework uses VRML for finite state machines, but does not actually look at any specific source code.

Some researchers have even explored different visualisation metaphors for source code comprehension. These metaphors include 3D cities (Software World [124, 125, 126], Component City [48], 3D City [178], and CodeCity [266] which uses OpenGL), a 3D solar system metaphor [96], 3D self organizing maps [31], and 3D computer game engines (Quake2 [123] and Quake3 [135]).

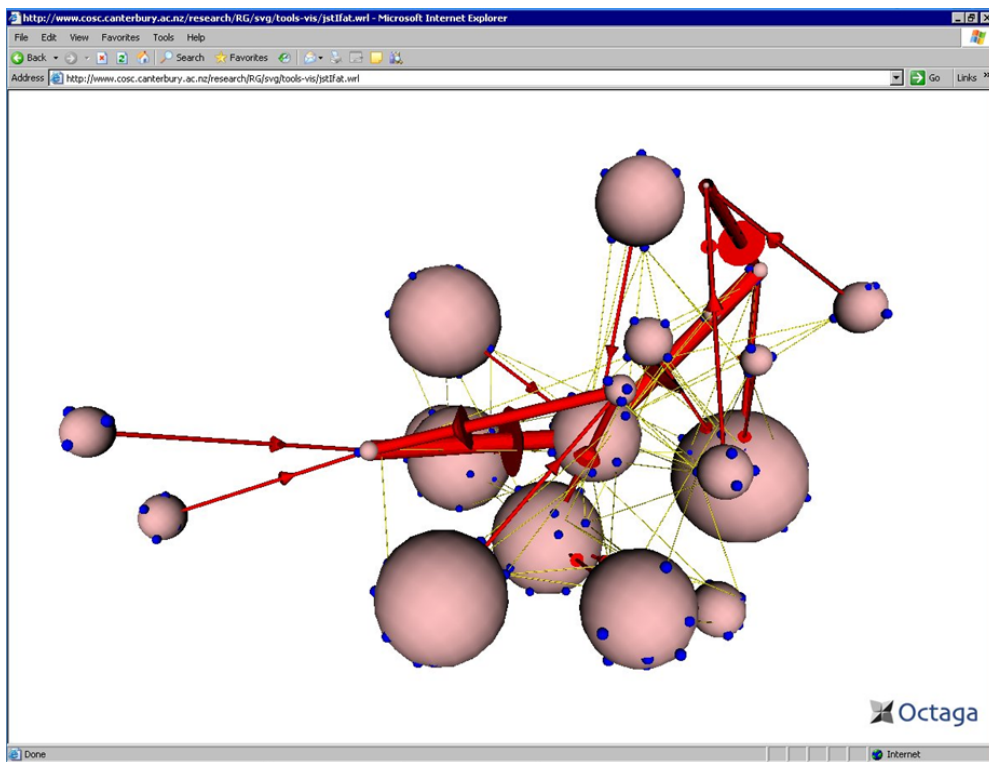
### 2.2.3 Object-Oriented Metrics

Churcher et al. [57, 118] use VRML for software visualisation and mainly focus on object-oriented metrics. They visualise inheritance structures with cone trees, inheritance structures with metrics, hierarchies with tree maps, web sites [102], class cohesion [56], and object-oriented metrics and class clusters [114]. Figure 2.10(a) shows a Compound Tree Map which is an extension to the Tree Map [116, 227]. Each layer represents different weighted information such as the size or age of the components. The original tree structure is also included to emphasise the relationship with the weighted tree map. Figure 2.10(b) shows a node-link diagram of object-oriented metrics focusing on class cohesion. Other key areas they explore are the layout of virtual worlds using 3D layout algorithms using a force directed approach [53, 55], XML in the visualisation pipeline [113], and the software visualisation design process in terms of a design pipeline [54]. The

design pipeline is essentially an architecture for visualising software.



(a) 3D Compound Tree Map [57].



(b) Class Cohesion [57].

Figure 2.10: Object-Oriented Metrics in VRML [57].



Various other systems and research groups have also looked at visualising object-oriented metrics in 3D including CrocCosmos [144, 145] (OpenGL and VRML), MetaViz [215, 216, 217] (Java3D), and Langelier et al. [137, 138] (Direct3D).

#### 2.2.4 UML Diagrams

Visualisations of various UML diagrams such as class, object, sequence, and collaboration diagrams have been explored in 3D using Java3D [66, 76], VRML [90, 92, 197] and X3D [163]. Displaying UML in 3D – which is intended to be drawn on 2D surfaces – does not scale well once there are many nodes in a world. Text is also hard to render in 3D. When text is rotated in 3D it is hard for a user to view what is meant to be displayed. Rather than representing strict UML diagrams in 3D, some research has been conducted that represents UML diagrams as 3D geon diagrams [110, 111, 112]. The geon diagrams are made from 3D primitives such as cones, spheres, cylinders, and boxes. There also exists a software visualisation tool based on the geon UML representation and implemented in Java3D [47].

Figure 2.11 shows approximately 700 classes from the Java3D API implemented by McIntosh et al. [163] in X3D. The class information is parsed using JavaML which creates an XML file for each class. Each XML class file is then transformed using XSLT into separate X3D files. The root X3D visualisation file contains include statements to all the other X3D class files. The layout of each of the X3D class files in the root X3D visualisation file is not automated and is hard coded. They found that their UML class visualisations only worked in one X3D browser. This is the only other research group that we are aware of that is using X3D to create UML software visualisations.

#### 2.2.5 Dynamic Visualisations

BLOOM [202, 203, 204, 205, 209, 210, 211] is a system for understanding software through visualisation, see Figure 2.12. BLOOM provides facilities for static and dynamic data collection and offers a wide range of data analysis. The system includes a visual query language for specifying what information should be visualised. All these are used in conjunction with a back end that supports a variety of 2D and 3D visualisation strategies. Some more recent systems by the same researchers include JIVE [206, 207] and JOVE [212] for dynamic visualisations of real applications, however, these systems do not create 3D visualisations.

Pounamu [277] is a system which can produce static or dynamic software architecture notation in XML. They have built a number of tools that convert the

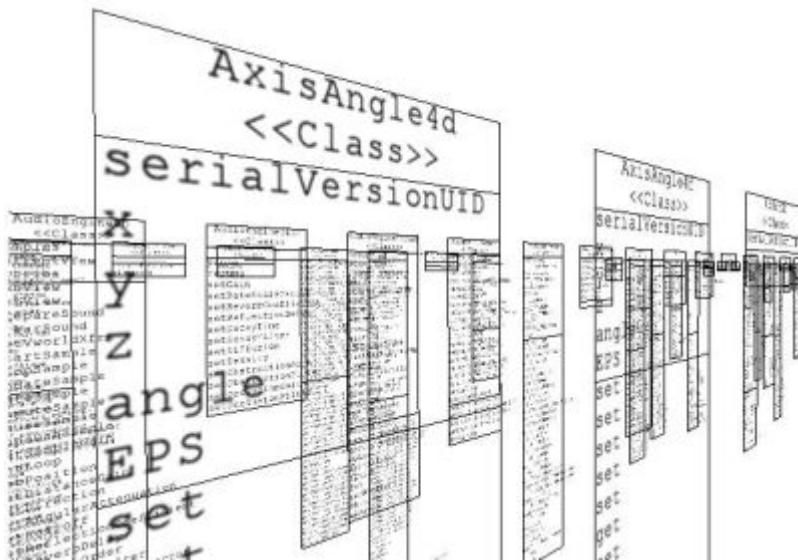


Figure 2.11: X3D-UML [163].

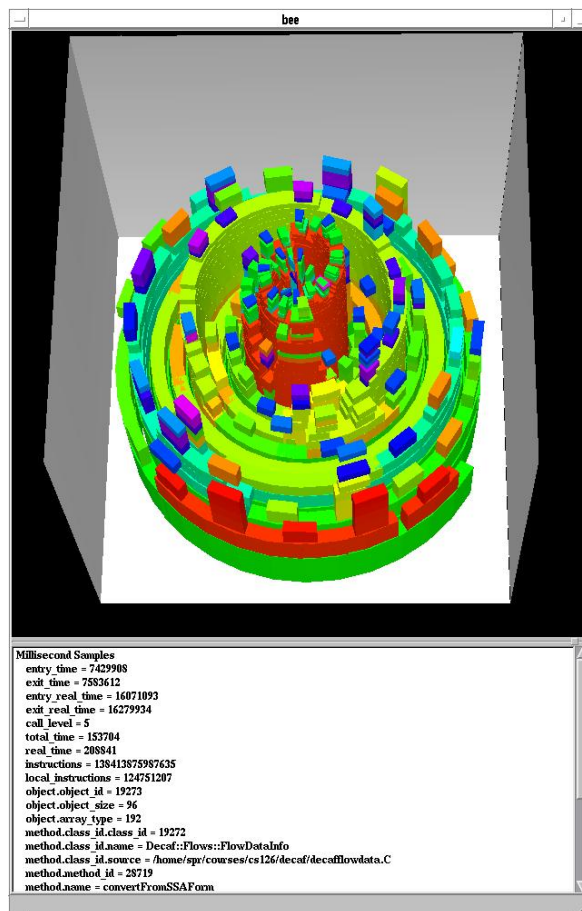


Figure 2.12: BLOOM, Spiral views of the stack (sampled during execution) [203].

XML notation about a program into the Graph eXchange Language (GXL)<sup>2</sup> [269] and can then convert the GXL information into SVG or VRML [241]. Knight and Munro [127] have also explored using GXL to create visualisations for program comprehension.

CodeCrawler [139, 140, 141] and the associated tools TraceCrawler [97, 98] and CCJUN [272] explore visualising feature traces in 3D for the purposes of object-oriented metrics. These tools use the Jun 3D framework to render the visualisations. Jun 3D is implemented in Smalltalk and rendered in OpenGL.

Storer et al. [242] has developed a tool for teaching object-oriented programming concepts to introductory level computer science courses. The tool provides Java3D visualisations of the execution of Java programs including representation of classes, objects, references, and method execution.

Charters et al. [49] question whether the time has come when no new advances are being made in software visualisation and all advances are just variants on old themes. We now discuss our approach to advancing the field of software visualisation.

## 2.3 VARE: Visualisation Architecture for REuse

The Visualisation Architecture for REuse (VARE) [160] project is an ongoing effort in our software design research group for generating web-based visualisations of programs to support code reuse. The design of VARE supports multiple programming languages and provides user control for the different parts in the visualisation process.

### 2.3.1 Programming Mapping Visualisation

VARE is based on the Programming Mapping Visualisation (PMV) Model, which is a conceptual model for describing program visualisation systems. The model was developed by Stasko [234, 239] and Roman and Cox [221] and has three separate elements: the program component, the mapping component and the visualisation component, see Figure 2.13.

The role of the program component is to collect information about a program either from the source code or while a program is executing, and passes this information to the mapping component. The mapping component transforms or changes either all of this information or a subset into an appropriate format and gives the transformed information to the visualisation component. The visualisation component then turns the transformed information into a visualisation.

---

<sup>2</sup><http://www.gupro.de/GXL/>

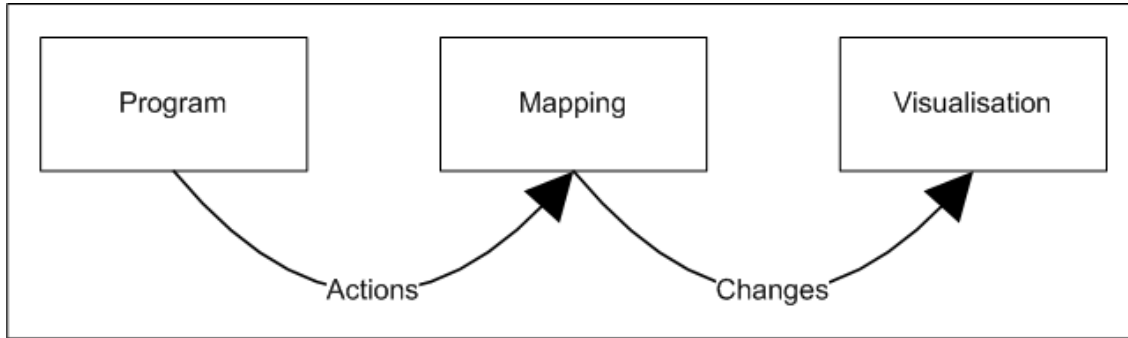


Figure 2.13: The Programming Mapping Visualisation Model [221, 234, 239].

### 2.3.2 Architecture

VARE is a client-server architecture, see Figure 2.14. The server contains repositories and processes. Dashed lines represent test drive traces or visualisation input or output, and solid lines represent control, queries or responses. On the client side, the user manages the activities associated with creating and viewing a visualisation. The component repository interface lets the user select a component from the repository to create a component set. Once this is created, the user can select an engine type from the engine repository to control the *test driving* of these components. Test driving is defined as specifying a sequence of method invocation and field access/modifications and then executing the sequence on a component [157, 158]. The engine represents the program component from the PMV model.

The engine generates an execution trace as output [12], which is stored in the execution trace repository [9, 10]. An execution trace is then used as input to a transformer. The transformer component then modifies the execution trace into an appropriate form for a visualisation [74]. The transformer repository interface lets the user select the transformer to use and the execution trace to use with it. The transformer represents the mapping component from the PMV model.

Finally, the finished visualisation is stored in the visualisation repository. The visualisation interface lets a user choose a particular visualisation and control its presentation. The visualisations represent the visualisation component from the PMV model.

### 2.3.3 Execution Traces

Visualisations of the behaviour of software are important because the dynamic behaviour is typically ephemeral. We are interested in capturing the runtime information of a program into an execution trace and then visualising subsets

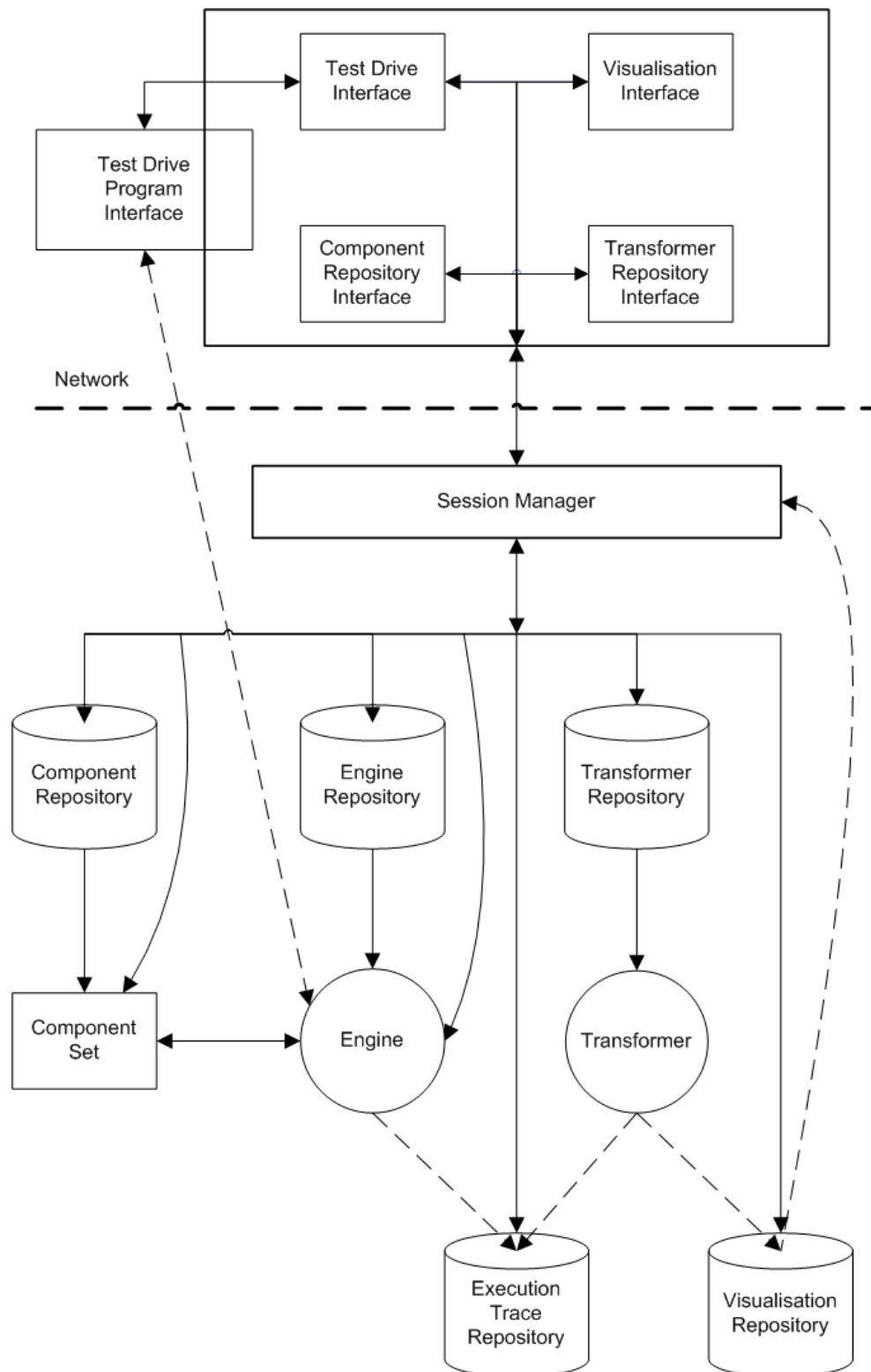


Figure 2.14: The VARE architecture [160].

of the information contained within an execution trace. Tracing the execution of a component involves catching information such as method calls, method returns, field accesses, field modifications, object creation and object deletion in an execution trace. Static information such as classes, super-classes, methods, and fields can also be gathered from source files.

We have created a list of requirements for the kinds of information we would like to see in an execution trace language and have created two XML execution trace languages. Information on the requirements [159] and the languages [12], the Process Abstraction Language (PAL) [161] and Reusable Component Descriptions (RCD) for static information and eXtensible Trace Executions (XTE) for dynamic information [157] can be found elsewhere.

Figure 2.15 shows an example XTE execution trace from a Java program. The example shows three events, the creation of an object, a method call, and a method return. Each event has an event id and an id for the thread the event belongs to. Lines 1–7 show the StoreView object being created. Lines 8–13 show the getUtilities method being called, and then lines 14–19 show the return of the getUtilities method.

```

1 <xte:objectcreation xte:eventid="event56" xte:objectid="object22"
2   xte:threadid="thread1">
3   <xte:complextype>
4     <xte:typename>nz.ac.vuw.mcs.comp301.view.gui.StoreView
5   </xte:typename>
6 </xte:complextype>
7 </xte:objectcreation>
8 <xte:methodcall xte:eventid="event57" xte:receiver=""
9   xte:sender="object22" xte:threadid="thread1">
10  <xte:methodname>getGUIUtilities</xte:methodname>
11  <xte:typename>nz.ac.vuw.mcs.comp301.view.gui.GUIUtilities
12 </xte:typename>
13 </xte:methodcall>
14 <xte:methodreturn xte:eventid="event58" xte:threadid="thread1">
15  <xte:objectvalue xte:objectid=""/>
16  <xte:methodname>getGUIUtilities</xte:methodname>
17  <xte:typename>nz.ac.vuw.mcs.comp301.view.gui.GUIUtilities
18 </xte:typename>
19 </xte:methodreturn>

```

Figure 2.15: XTE — example execution trace of a Java program.

### 2.3.4 Visualisation Tools

Some recent visualisation tools that use the VARE architecture will now be described. Spider [157, 158] is a prototype for exploring and documenting reusable components in a web environment. Figure 2.16 shows a test drive of the Java Calendar component. Spider documents components with Reusable Component Descriptions (RCD) and execution traces with eXtensible Trace Executions (XTE) by interpreting information stored in a component, detecting events in the run-time environment, and interrogating the runtime environment's state. Spider uses the Java Debugger Interface [165] to extract information.

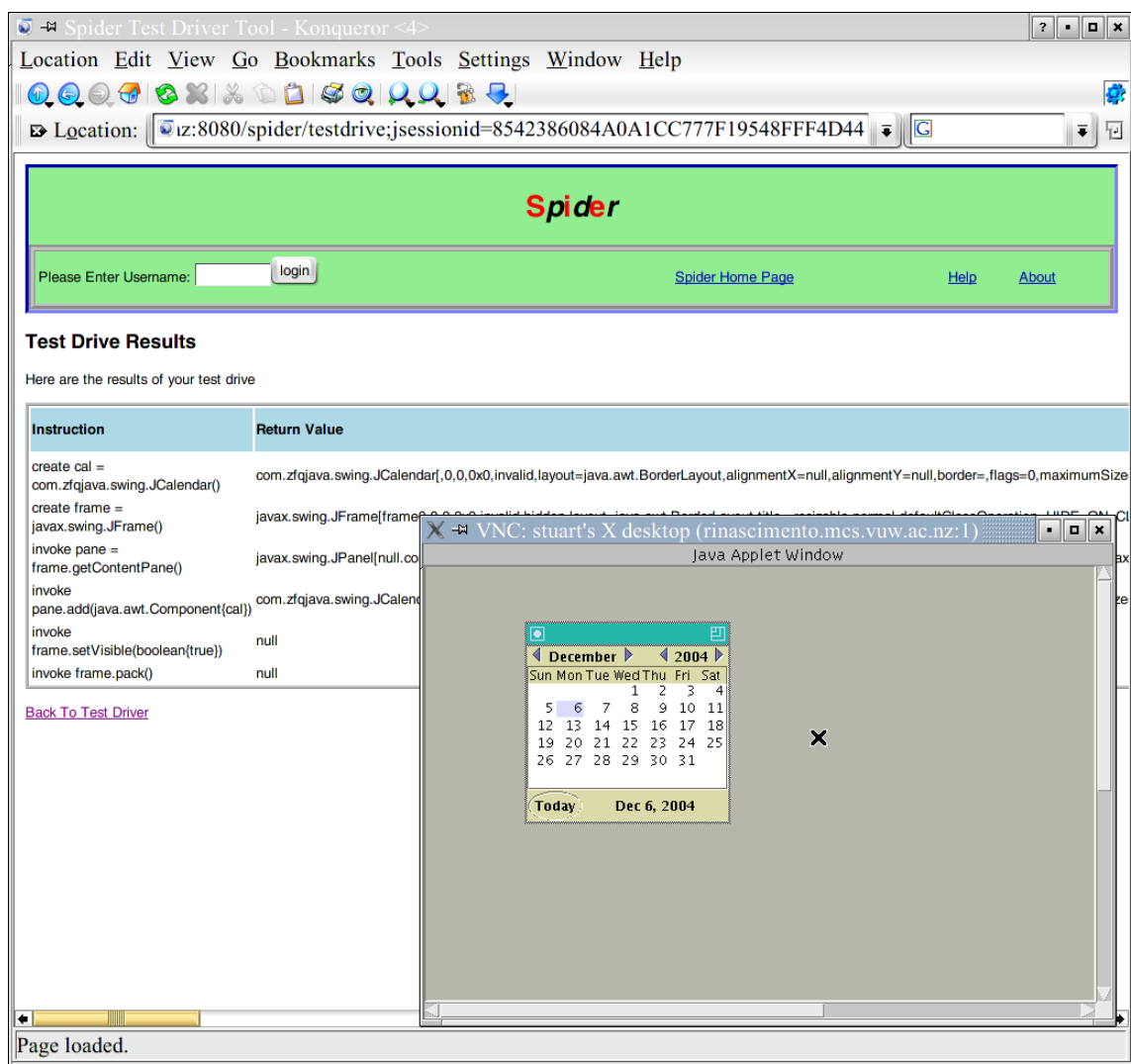


Figure 2.16: Spider, a test drive of the Java Calendar component [157, 158].

Visualisation of Execution Traces (VET) [162] is a prototype visualisation application that lets users interact with and understand execution traces. VET follows the methodology of show all data of an execution trace, then let users filter out

unwanted data, and provide details on demand. VET uses filters so that users can directly adjust the information being displayed by the visualisations. VET is a plug-in based architecture which allows users to design their own visualisation plug-ins, so that a user can decide what to draw and when particular events occur. Users can also design their own filters. Figure 2.17 shows two visualisations, the top one is a sequence diagram while the bottom one is an association diagram. The two visualisations are synchronised at the same location within the execution trace and also with the same information that is displayed.

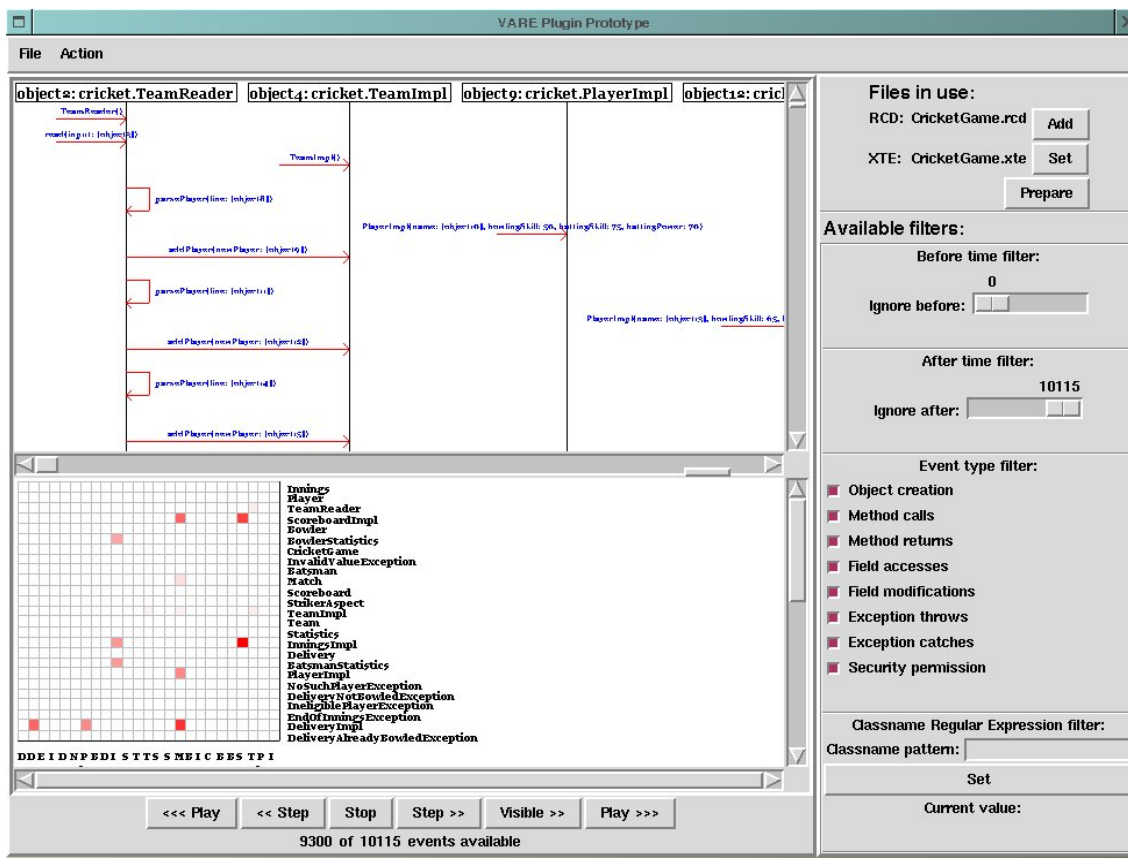


Figure 2.17: VET displaying a sequence and an association diagram of the same data from an execution trace [162].

Blur [74, 75] is a prototype for visualising execution traces by transforming them into SVG visualisations that represent UML class or sequence diagrams. Blur is implemented as a Java Servlet running in Apache Tomcat and can be accessed over the web. Figure 2.18(a) shows a SVG UML interactive class diagram. When the mouse covers a piece of code in the right hand side frame, the left hand side highlights the appropriate class or method in the UML diagram. This is a helpful tool for developers, because they can have a clear understanding of the class diagram, and how the code works. Figure 2.18(b) shows a SVG UML sequence



diagram. This is also helpful for the developer, as they can see the sequence of interactions during the execution of a program.

Our research group have previously explored a few 2D technologies for visualising our execution traces in VARE. Blur [74, 75] identified several reasons why SVG failed to meet expectations for software visualisation. We are now interested in seeing if there are any possible web-based 3D technologies that could be used to complement our SVG visualisations.

### 2.3.5 3D Technologies

The aim of this thesis is to evaluate X3D for use in software visualisation. In this section we briefly review some alternative 3D technologies. A more complete exploration would require measuring each of the alternative solutions against our framework for evaluating software visualisation media (§5) and compare with our results from evaluating X3D (§6).

#### VRML

The Virtual Reality Modeling Language (VRML) [6, 103, 257, 264] is a standard file format for representing 3D interactive vector graphics, over the web. The first version of VRML was specified in November 1994. The current version was created in 1997 and is referred to as VRML97. VRML is a text file format where a 3D polygon can be specified along with for example the surface colour, image-mapped textures, shininess, and transparency. Animations, sounds, lighting, and other aspects of the virtual world can interact with the user or may be triggered by external events. VRML files are called worlds and have the .wrl extension.

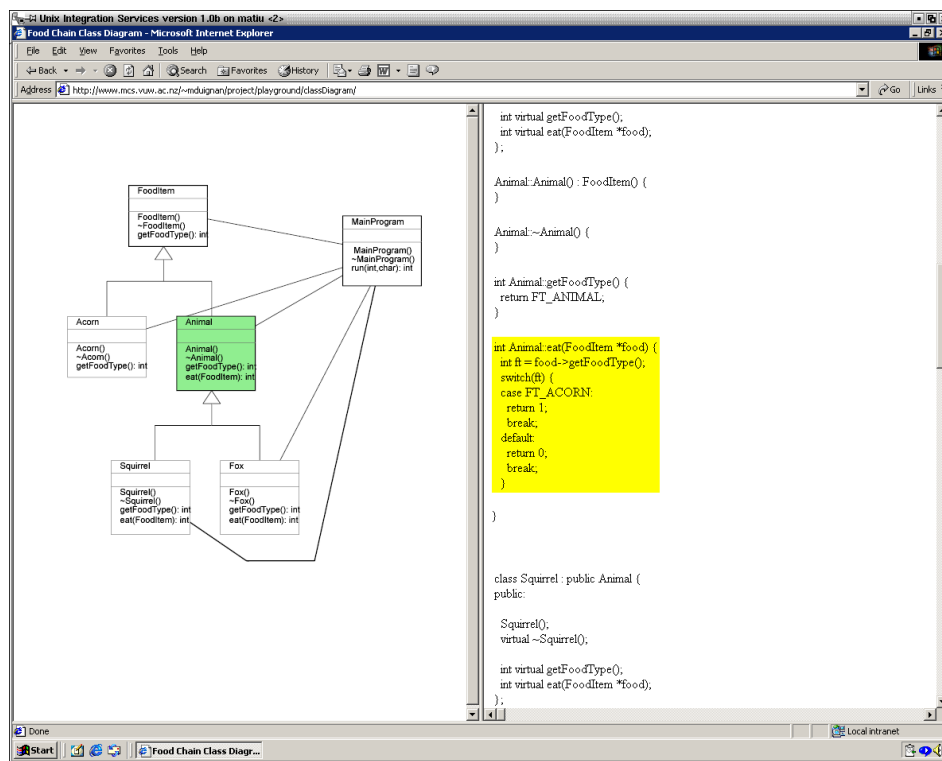
Other VRML specifications include an XML Schema version of VRML<sup>3</sup> and an object-oriented language which extends VRML [67]. Neither of these applications are well supported or sponsored by the Web3D Consortium. VRML was not used for this project as X3D has replaced it and it is not easy to incorporate VRML with other applications. Of note is that the Classic VRML encoding and the immersive profile of X3D are directly similar to VRML97. Thus, any approach applied to X3D should also work for VRML97 as well.

#### Java 3D

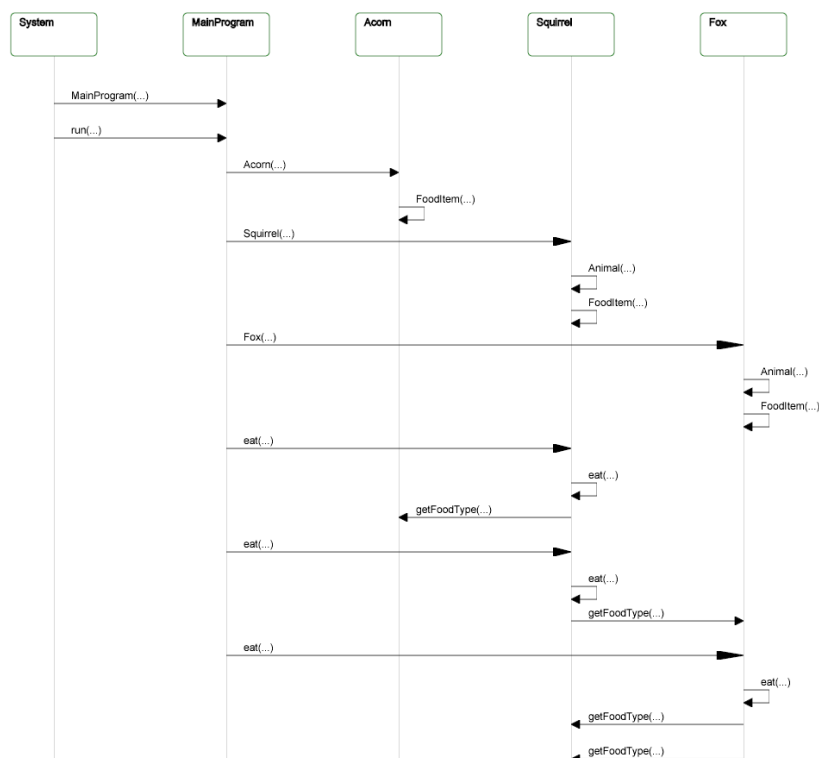
Java 3D [225] provides a set of object-oriented interfaces that support a simple, high-level programming model you can use to build, render, and control the behaviour of 3D objects and visual environments. The language is not lightweight

---

<sup>3</sup><http://www.xvrm1.net>



(a) SVG interactive class diagram [74].



(b) SVG sequence diagram [74].

Figure 2.18: Blur SVG visualisation tool [74].

as it is a binary format and needs a viewer to display the technology. Java3D does not work with commercial web browsers unless used with Java applets, basic graphics need to be encoded, and navigation options have to be developed.

### **OpenGL**

The Open Graphics Library (OpenGL)<sup>4</sup> is a specification defining a cross-language cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of about 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL is used in CAD, virtual reality, scientific visualisation programs, information visualisation, and video game development. OpenGL is a very heavyweight solution and requires writing all functionality as a software program to produce a visualisation. Other programming languages make use of OpenGL by having language bindings for rendering.

### **Linden Scripting Language**

Second Life<sup>5</sup> has a scripting language called the Linden Scripting Language (LSL) [223] named after the company that created Second Life, Linden Labs. LSL is a programming language for users to create virtual worlds in Second Life and has a syntax that is similar to C. The scripts are compiled to byte-code before run-time execution in a virtual machine on a Linden Lab server. LSL is event driven, features states, 3D variable types, and has functions for manipulating physics and avatar interaction.

### **COLLABorative Design Activity (COLLADA)**

COLLABorative Design Activity (COLLADA) [14] is an intermediate or interchange file format designed for interactive 3D applications. COLLADA is an open XML standard for exchanging digital assets that store their assets in incompatible proprietary formats.

COLLADA was originally created by Sony Computer Entertainment as the official format for PlayStation 3 and PlayStation Portable development [15]. COLLADA is now owned by the Khronos Group<sup>6</sup> which also manages the OpenGL specification.

COLLADA was designed as a format for transporting data from one content authoring tool to another including: Maya, 3D Studio Max, Softimage XSI, and

---

<sup>4</sup><http://www.opengl.org>

<sup>5</sup><http://secondlife.com>

<sup>6</sup><http://www.khronos.org/collada>

Blender. Maya and 3D Studio Max are both owned by Autodesk. Some applications use the COLLADA format including the Unreal Tournament game engine<sup>7</sup> and Google Earth<sup>8</sup>.

### Extensible Application Markup Language (XAML)

The Extensible Application Markup Language (XAML)<sup>9</sup> is a new declarative XML language that defines objects and their properties. The syntax of XAML focuses on defining the user interface for the Windows Presentation Foundation (WPF) or Silverlight and is separate from the application code. XAML supports 3D and control features. XAML requires the .Net framework to be installed on a users machine. Since XAML is very new, a Microsoft technology, not cross platform, and primarily used for user interfaces it is not a standard worth considering yet for our software visualisations.

### Graph Drawing Systems

One final kind of alternative technology would be to use specific graph drawing [250] systems designed for drawing large 3D directed graphs. Some systems include WilmaScope [77] (Java3D) and Walrus<sup>10</sup>. Most of these systems do not run in a web browser and would require lots of customisation to work with our software visualisation system.

## 2.4 Summary

We are interested in understanding what software looks like to help with software reuse, software maintenance, and software re-engineering. We believe that capturing the static (source code) and dynamic (run-time) information about software in execution traces, and then applying information visualisation techniques to the execution traces, will help to assist developers to understand the structure and behaviour of software.

We have an existing software visualisation architecture [160] that requires tools and visualisations to be created to understand software. The interface to our tools are web based and we have already created a 2D visualisation tool that can produce SVG software visualisations [74, 75]. Ware et al. [261] demonstrate that displaying object oriented software in three dimensions instead of two can make it

---

<sup>7</sup><http://www.unrealtournament3.com>

<sup>8</sup><http://earth.google.com>

<sup>9</sup><http://www.xaml.net>

<sup>10</sup><http://www.caida.org/tools/visualization/walrus/>

easier for users to understand the data. We want to create software visualisations over the web in 3D.

Very little research has been done on the applicability of using X3D [265] – the Web3D Consortium’s open standard for web 3D graphics – for software visualisation. We are aware of one other research group using X3D for visualising UML class diagrams [163], see Figure 2.11 (§2.2.4). We believe it is imperative to evaluate X3D to determine how applicable X3D is for use in software visualisation. The benefit of this evaluation will be that it will help other developers to determine whether or not X3D is an appropriate media for use in the development of their software visualisation systems.

Most existing 3D software visualisation systems visualise source code, object-oriented metrics, UML diagrams, or execution traces. Most of these systems are not web based and are usually standalone desktop applications. The visualisations produced from these system usually render the visualisations in languages that are either no longer supported (VRML), are binary (Java3D), and are very heavy weight (OpenInventor, OpenGL).

X3D is more light-weight, is the open standard for web 3D graphics, has an XML encoding, and can be used over the web. X3D alleviates the problems of other 3D languages (§2.3.5) as it is not a heavy-weight solution, not a binary format, and not designed as an intermediary format. In this thesis we have decided to evaluate X3D [265] for use in software visualisation to see if these characteristics of X3D meet the requirements for our ongoing software visualisation research project (§2.3).

This thesis aims to answer the following research questions:

- How good is X3D for use in software visualisation?
- How well does X3D support the software visualisation pipeline, by creating X3D visualisations from execution traces?
- How well does X3D support our software visualisation architecture?

In the next chapter we explore the X3D language, while the following chapter shows our 3D visualisation tool and the X3D software visualisations that can be produced from the tool.



# Chapter 3

## Exploring X3D

### Contents

---

<b>3.1 Overview</b>	<b>38</b>
<b>3.2 Specification</b>	<b>39</b>
3.2.1 Basic Example	39
3.2.2 Encodings	39
3.2.3 Profiles	39
<b>3.3 X3D Nodes</b>	<b>43</b>
3.3.1 Geometry	43
3.3.2 Grouping	44
3.3.3 Viewing and Navigation	47
3.3.4 Lighting, Environment, and Sound	47
3.3.5 Animation and Interaction	48
3.3.6 Prototypes	49
<b>3.4 Resources</b>	<b>49</b>
3.4.1 X3D Browsers and Plug-ins	49
3.4.2 Content Authoring and Editing Tools	51
3.4.3 Export and Translator Tools	55
<b>3.5 Discussion</b>	<b>55</b>
3.5.1 Improvements to the Specification	57
3.5.2 Future of X3D on the Web	58

---

X3D [265] is a royalty-free open standards file format and run-time architecture to represent and communicate 3D scenes and objects over the web. X3D is an International Standards Organization (ISO) standard that provides a system for the storage, retrieval, interactive, and playback of real time graphics content embedded in applications, all within an open architecture to support a wide array of domains and user scenarios [265]. The aim of this chapter is to give an overview of X3D, highlighting key components from the specification, describing some available tools, and discussing the prospects and the viability of X3D on the web in the future.

### 3.1 Overview

X3D is the successor to the Virtual Reality Modeling Language (VRML) [264] from the same standards authority, the Web3D Consortium<sup>1</sup>. There exists a good overview of X3D fundamentals [42] and a book on X3D by Don Brutzman and Leonard Daly [41]. Complete details on VRML and early details on X3D are available in the literature [6, 257].

There is a broad range of application areas where X3D is currently being applied. Some areas include interactive marketing (entertainment and educational titles), building architecture and urban planning, geography, scientific visualisation, industry engineering (oil and gas, automotive, e-learning, virtual training and simulation for military and navy), mobiles and PDAs, and education [52].

The basic structure of X3D documents is very similar to any other XML document. Scene graphs<sup>2</sup>, nodes, and fields (in X3D terminology) correspond to documents, elements, and attributes (in XML terminology) [41]. X3D documents combine both geometry and the run-time behaviour into a single XML file. The scene graph is the run-time environment of an X3D file. Nodes within the scene graph can have descriptive fields and can contain one or more child nodes.

X3D content maybe presented in a native X3D browser, a web browser that has an X3D plug-in or transformed and delivered to a VRML browser. X3D content can be created from DTDs or XML Schemas and edited using authoring tools, text editors, or transformed using Extensible Stylesheet Language Transformations (XSLT) [191, 192]. X3D allows scripts to be embedded such as JavaScript, or refer to external JavaScript or Java binary files. The X3D browsers allow full 3D navigation.

---

<sup>1</sup><http://www.web3d.org>

<sup>2</sup>A scene graph is a directed, acyclic graph containing objects represented as nodes and the relationships between objects in the 3D world [89].



## 3.2 Specification

The first X3D draft specification was developed in 1999 and launched as an open standard in 2001. The specification was first submitted to the ISO in 2002 and became official in 2004 as ISO/IEC 19775:2004 - Extensible 3D (X3D) [89] or X3D version 3.0. The current version is 3.2.

### 3.2.1 Basic Example

Figure 3.1 shows the basic Hello World X3D example [41]. The X3D code declares the text “Hello world!”, a sphere, and applies an image of the world as texture from Figure 3.2 to the sphere. Figure 3.3 displays the X3D content in Mozilla Firefox version 2.0 using the Octaga X3D web browser plug-in. Figure 3.4 represents the tree structure of the X3D nodes from Figure 3.1.

A user can rotate the globe in Figure 3.3 or zoom-in/zoom-out to find countries of interest. Lines 1-7 in Figure 3.1 are default X3D information which define the XML version, the X3D DTD and XML Schema, Immersive profile, and the X3D root node. Lines 8-10 define the meta data, which there can be more of. Lines 11-28 define the contents of the X3D scene. Line 12 defines a specific viewpoint which will be the default view that the X3D browser loads initially. Lines 14-20 define the sphere with the image of the world used as texture. Lines 21-26 define the text to display and define what colour the text is. Line 29 is the closing tag of the X3D root element.

### 3.2.2 Encodings

There are specifications for encoding X3D in XML, binary, and a VRML derived encoding called X3D classic. X3D files using the XML encoding are saved with the file extension .x3d, .x3dv for X3D VRML classic encoded, or .x3db for X3D binary. VRML files have the extension of .wrl which stands for world. There are also language bindings for ECMAScript (implemented as JavaScript) and Java as well as node prototyping, which together provide support for scene graph extensions and new language functionality [41].

### 3.2.3 Profiles

X3D is organised as a set of components where each component describes a set of related functionality. Profiles are built from components and are standardised sets of extensions to meet specific application needs. There are four main profiles in ascending functionality order, see Figure 3.5:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.2//EN"
3   "http://www.web3d.org/specifications/x3d-3.2.dtd">
4 <X3D profile="Immersive" version="3.1"
5   xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
6   xsd:noNamespaceSchemaLocation="http://www.web3d.org/
7     specifications/x3d-3.1.xsd">
8   <head>
9     <meta content="HelloWorld.x3d" name="title"/>
10  </head>
11  <Scene>
12    <Viewpoint description="hello, world!" position="0 0 10"/>
13    <Group>
14      <Shape>
15        <Sphere/>
16        <Appearance>
17          <ImageTexture url='"earth-topo.png"
18            "http://www.web3d.org/x3d/content/examples/earth-topo.png"' />
19        </Appearance>
20      </Shape>
21      <Shape>
22        <Text string='"Hello" "world!"' />
23        <Appearance>
24          <Material diffuseColor="1 1 1"/>
25        </Appearance>
26      </Shape>
27    </Group>
28  </Scene>
29 </X3D>
```

Figure 3.1: X3D content that encodes a sphere with an image of the world used as texture and the text “Hello world!” [41].

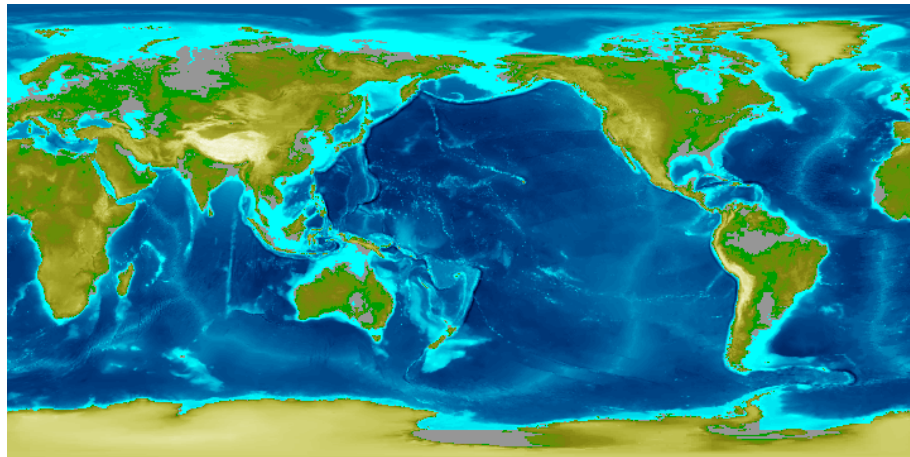


Figure 3.2: An image of the world earth-topo.png [41].

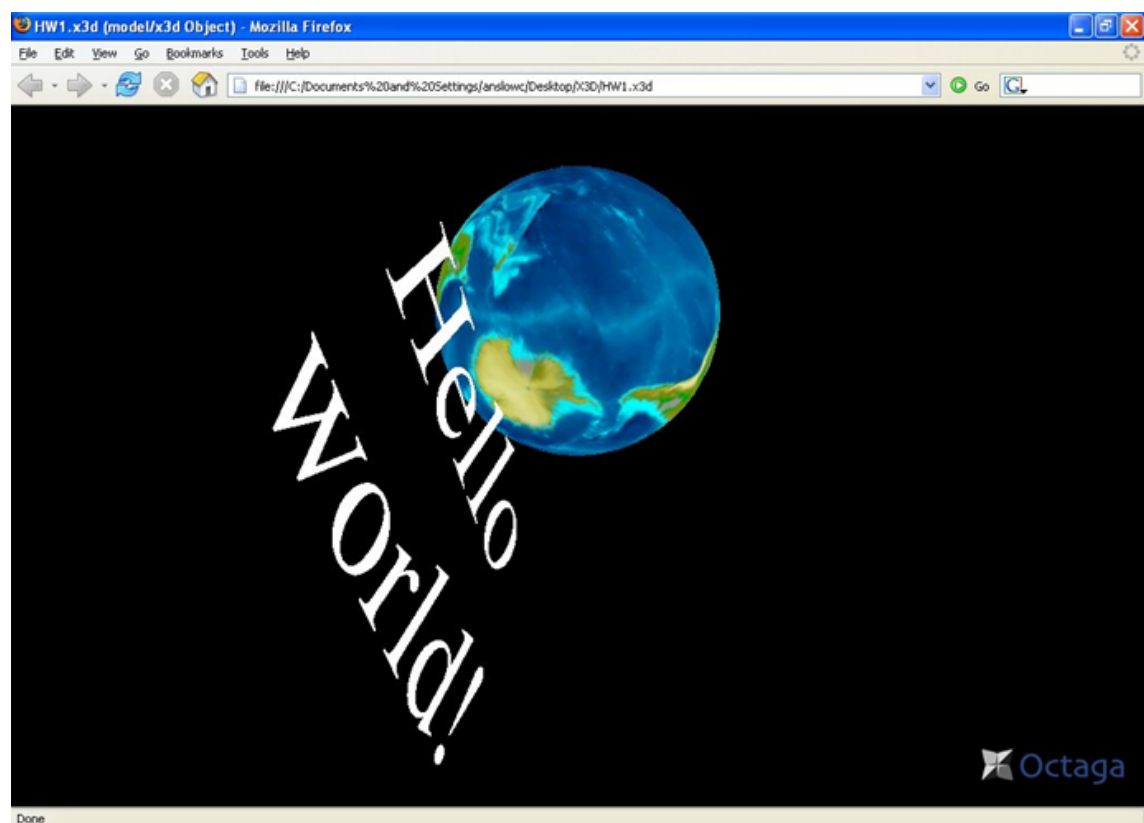


Figure 3.3: X3D Sample Document — Hello World.

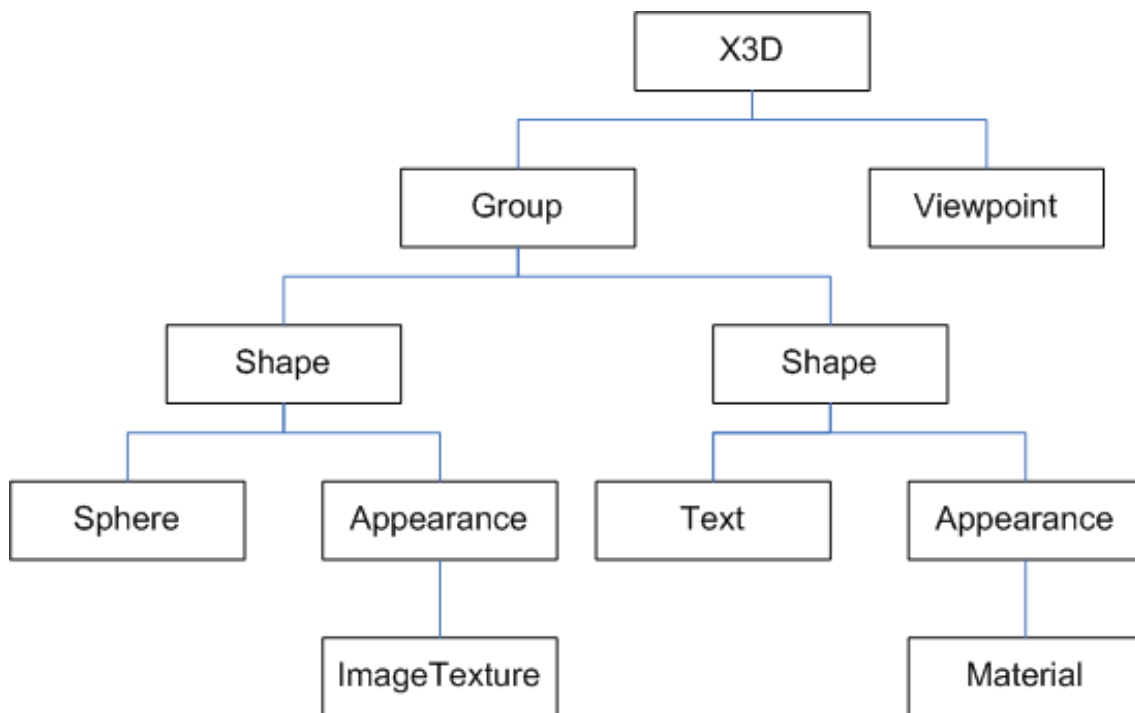


Figure 3.4: The X3D nodes represented as a tree structure from the Hello World sample document.

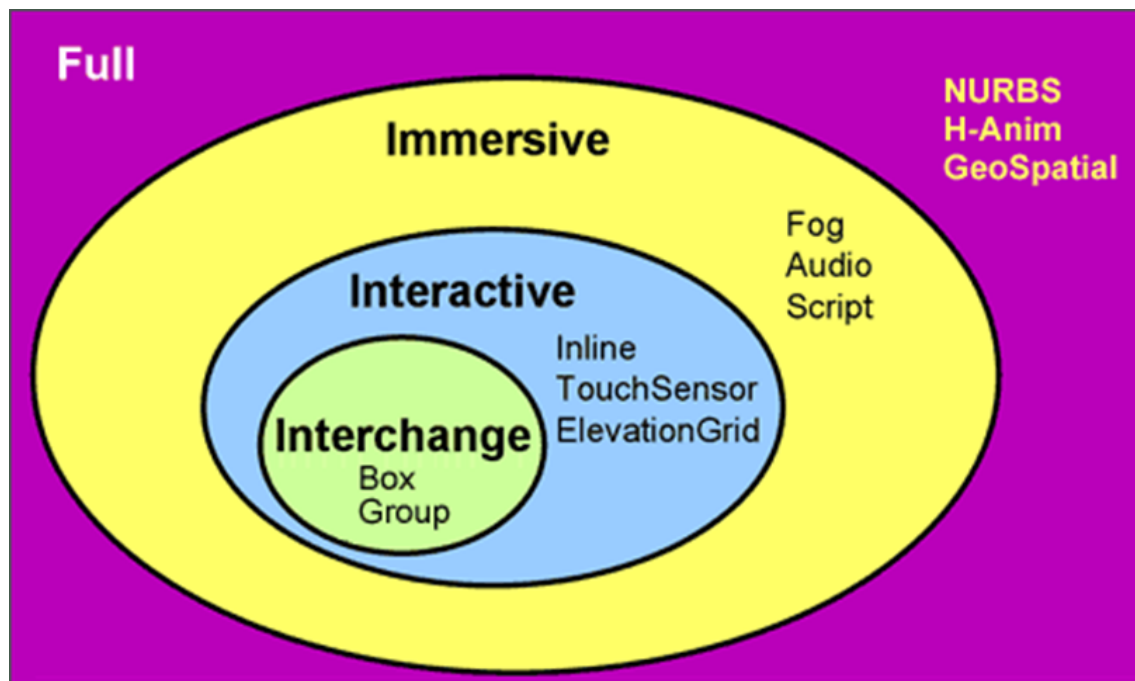


Figure 3.5: X3D Baseline Profiles [265].

- Interchange - is the basic profile for communicating between applications, and supports geometry, texturing, basic lighting, and animation.
- Interactive - enables basic interaction with a 3D environment by adding various sensor nodes for user navigation and interaction, enhanced timing, and additional lighting.
- Immersive - enables full 3D graphics and interaction, including audio support, collision, fog, and scripting.
- Full - includes all defined nodes in the X3D specification and incorporates other advanced components such as NURBS<sup>3</sup>, H-Anim<sup>4</sup> and GeoSpatial<sup>5</sup>.

There are other additional profiles that complement the main four profiles which include MPEG-4 Interactive and CAD Distillation Format (CDF). MPEG-4 Interactive is a small subset of the Interactive profile designed for broadcast, handheld devices and mobile phones. CDF enables translations of CAD data to an open format for publishing and interactive media.

### 3.3 X3D Nodes

There are a number of nodes that can be used in an X3D scene including: geometry, grouping, appearance, material, textures, viewing, navigation, animation, interaction, lighting, environment, sound, and prototypes. We will now give a brief overview of some of the nodes. Further detailed information can be found elsewhere [41].

#### 3.3.1 Geometry

X3D supports 2D and 3D geometry. The 2D geometry nodes include: arcs, closed arcs, circles, polylines, polypoints, rectangles and triangles. The 3D geometry nodes include: boxes, cones, cylinders, spheres, indexed face sets, indexed line sets, elevation grids, and extrusion (see Figure 3.6).

Boxes, cones, cylinders, and spheres are specified as typical 3D shapes. Indexed face sets and indexed line sets specify coordinate points which are then indexed to draw shapes with faces or lines. Elevation grids are used to draw physical world objects like terrains, grounds, hills, and skies. Extrusion nodes, see Figure 3.7,

---

<sup>3</sup>NURBS is short for non-uniform, rational B-spline and is used for generating and representing curves and surfaces.

<sup>4</sup>Humanoid Animation is an abstract representation for modeling 3D human figures.

<sup>5</sup>The GeoSpatial component can be used for geographic and geospatial applications.

can be used to create a wide variety of shapes and are much like what happens when molten plastic is extruded through a hole pattern [41]. The extrusion node starts with a planar cross-section outline and stretches it out around a series of line segments called a spine. The spine is a list of 3D points for a piecewise-linear curve that form a series of connected vertices. Finally, text can be displayed with the text node, while the font style node defines the size, font face, layout, and style of the text.

Graphical properties can be applied to shapes by defining an appearance node. Lines 16–19 in Figure 3.1 show an example of the appearance node. The appearance node specifies the visual properties of geometry in an X3D scene. The appearance node has elements for defining a shape’s fill properties (filled or patterns), line properties (solid, dashed, dotted), texture (images, sound, movies, pixels), and material (ambient, colour, transparency). The material node specifies surface material properties for associated geometry nodes and is used by the X3D lighting equations during rendering.

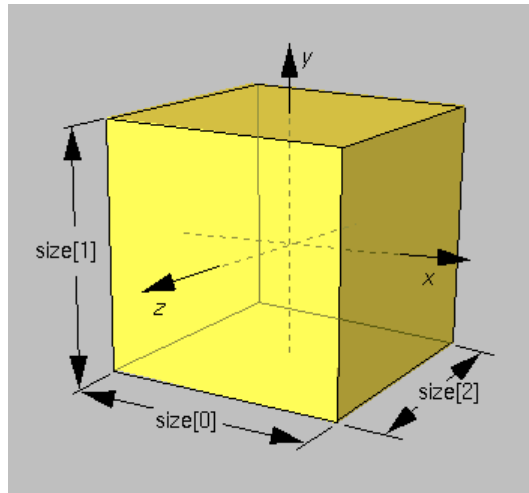
Nodes have built-in fields which have a field name, access type, and type of field. The field name is a textual description. The access type defines how the field is accessed and there are four possible values: `initializeOnly`, `inputOnly`, `outputOnly`, and `inputOutput`. The type of field can be a boolean, string, colour, integer, float, double, image, node, or a vector. Each field can be either a single or multiple (array) field.

The DEF and USE attributes are used for defining and copying a node, multiple nodes, or even groups of nodes. The DEF attribute is a textual identification label in a file, whereas the USE attribute refers back to a node with a DEF name. Essentially the DEF attribute is used to label a node and then the USE attribute is used in a subsequent location in the file which contains the same features of the defined node. These attributes together can make X3D files less verbose.

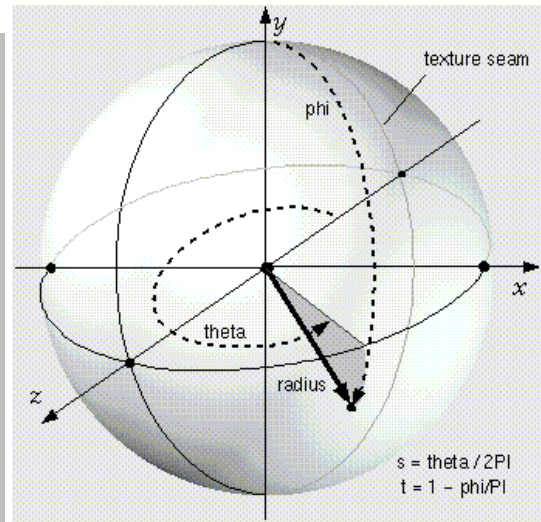
### 3.3.2 Grouping

Shapes can be grouped in an X3D scene in many different ways. The group node is used to collect related objects into a single parent in the scene graph hierarchy. The transform node is a grouping node that defines a coordinate system for its children that is relative to the coordinate systems of its ancestors. The transform node specifies a translation and a rotation field which has values for the X, Y, and Z coordinates. The inline node is essentially an include statement and can bring in nodes to the scene from another X3D file, referenced by a URL.

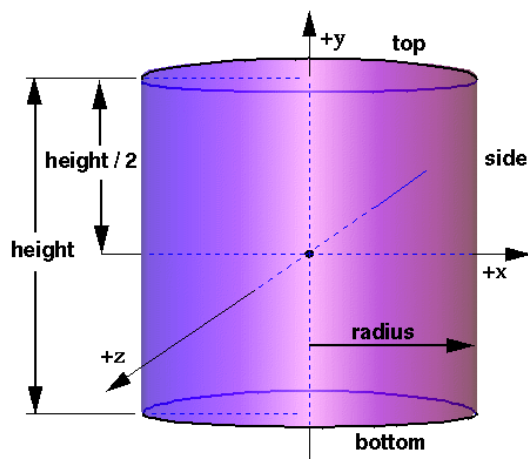
The level of detail (LOD) node is a grouping node that allows various levels of detail or complexity to be displayed for an object, and provides hints allowing



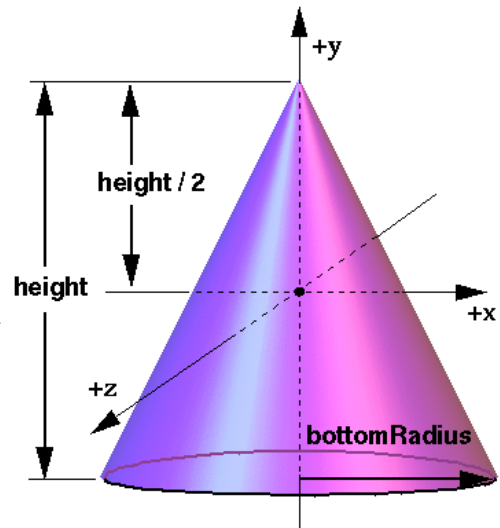
(a) X3D Box.



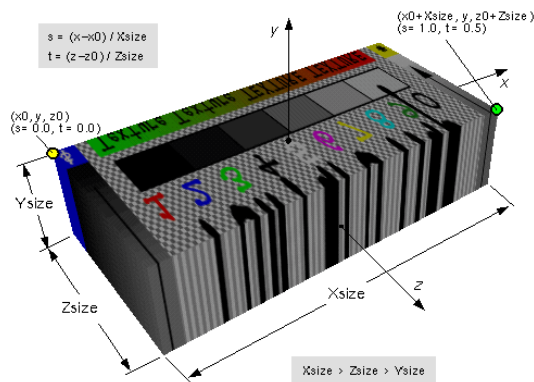
(b) X3D Sphere.



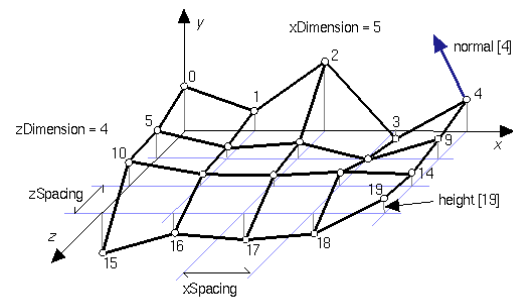
(c) X3D Cylinder.



(d) X3D Cone.

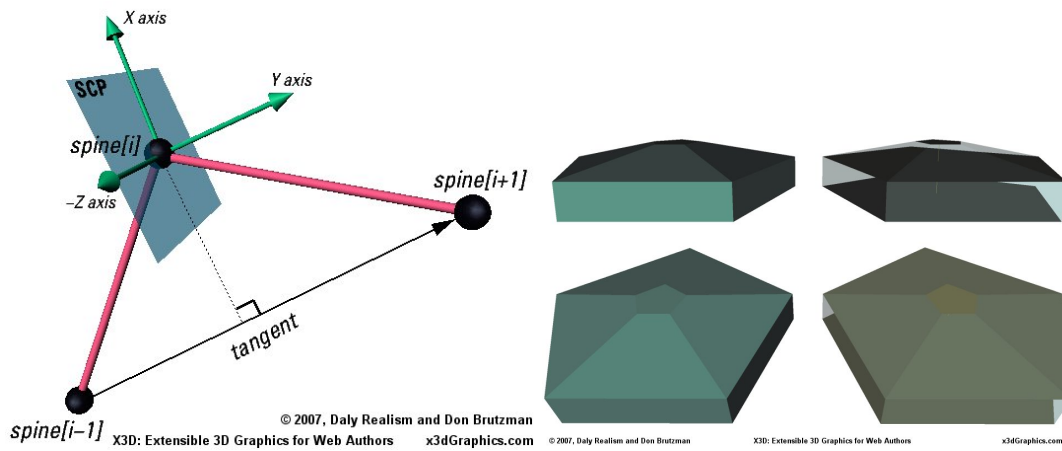


(e) X3D Indexed Face Set.



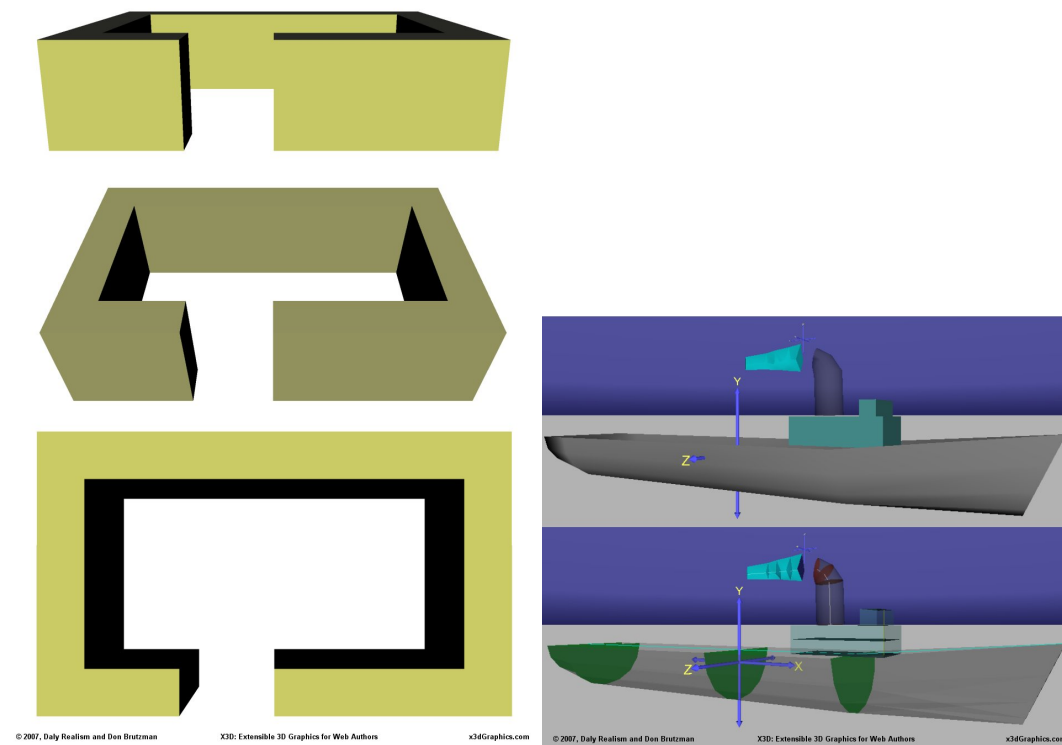
(f) X3D Elevation Grid.

Figure 3.6: Examples of 3D geometry definitions in X3D [41].



(a) Points making up a spine define a local Spine-aligned Cross-section Plane (SCP) where cross-sections are repeated.

(b) Example pentagon extrusion views showing default rendering on left and SCP on right.



(c) Extrusion example constructing the walls of a building.

(d) Extrusion example constructing the hull, superstructure, and smoke trail of a ship.

Figure 3.7: Examples of extrusion geometry definitions in X3D [41].



browsers to automatically choose the appropriate version of the object based on the distance from the viewer. The switch node can be used to render different child nodes (zero or more nodes) that are grouped together. The switch node can be used for animating geometry and providing customised level of detail. The rendering of the order of the child nodes can also be controlled.

### 3.3.3 Viewing and Navigation

X3D allows end-users full 3D navigation in a scene and supports the following navigation types: any, walk, examine, fly, look-at, none, slide, and pan. The navigation info node contains information describing the physical characteristics of the viewer's avatar and viewing model. The navigation info node can define the default navigation type, navigation speed, and rotation speed for a scene that will be used when a user loads an X3D file.

Users can navigate to predefined locations and viewing orientations using viewpoints. The viewpoint node defines a specific location in the local coordinate system from which a user may view an X3D scene. Viewpoints can also have textual descriptions to distinguish them. The anchor node allows geometry to be linked to other viewpoints in the scene or to external content (e.g. X3D or HTML files). The anchor node is similar to the HTML anchor tag.

Scenes can be more responsive for users when the billboard and collision nodes are used. The billboard node can be used to adjust shapes such as text to always be readable in the current viewpoint by a user. The billboard node is a grouping node which modifies its coordinate system so that the billboard node's local Z-axis turns to point at the viewer and the children geometry nodes are also rotated to change position. The collision node defines object collision detection properties for nodes. The collision node can be disabled to allow users to pass through objects.

### 3.3.4 Lighting, Environment, and Sound

Various lighting nodes can be used to make an effect on the X3D scene. Directional light illuminates the environment in a single direction. The headlight node is turned on by default as directional light and is fixed at the location and direction of the current user viewpoint. The point light node provides a single light source that is spread evenly in all directions, while the spotlight node highlights geometry within a cone-shaped beam.

The background and fog nodes can be used to provide important environmental effects. The background node can be used to colour the ground and sky with an array of colours or image textures. The fog node gradually replaces the colour

reflected by objects according to the distance from the viewer.

The load sensor node can be used for tracking the download progress of external files which can delay the start of animations until all downloads have completed. The proximity sensor node is used to detect changes in the current viewer position and orientation, and can be used to create heads-up displays (HUDs) such as a dashboard in first-person computer video games. The visibility sensor node detects whether a specified volume of space is visible from the current user view. The difference between the proximity and visibility sensor is that the visibility sensor does not detect whether the current user's view is within a predefined region.

The sound node identifies the source, location, intensity, direction, and spatial characteristics for a sound source in an X3D scene. The audio clip node provides and controls the source for the sound node. X3D supports the uncompressed wavefile (.wav), and compressed MIDI (.midi) and MP3 (.mp3) sound file formats.

### 3.3.5 Animation and Interaction

Animation and user interaction are controlled by sensors and interpolators in conjunction with the routing event model, see Figure 3.8. A user first clicks (touch sensor) or drags (plane, cylinder, or sphere sensor) a geometry node, or types (key sensor) which activates a clock node (time sensor node). The time sensor then sends values to an interpolator node (scalar, colour, position, orientation, and coordinate), which in turn outputs values whereupon some other geometry in the scene is modified (e.g. colour, position, orientation). At each step in the animation, events are passed from one node's output field to another node's input field via a ROUTE directive. This animation process is referred to as the routing event model. The identification of each node in the routing event model is located via the DEF attribute of the node.

Figure 3.8 shows the X3D routing event model where a user can click or drag a node which is detected by mouse touch and drag sensors. The touch and drag sensors first activate a timer which then activates the frames of an interpolator to change a variable in the target node. Depending on the visualisation, user interaction is not always required to initiate animations. Instead the animation can be activated by only a time sensor. ROUTE directives are used and act as the glue to link the sensors, interpolators, and target nodes together.

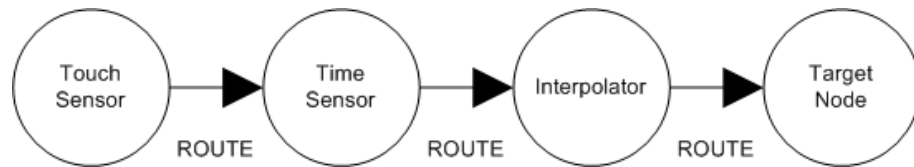


Figure 3.8: X3D Routing Event Model [41].

### 3.3.6 Prototypes

Prototypes provide a way to extend the X3D language by creating new nodes from other shapes that can be reused in an X3D scene or other scenes. Prototypes are a customised way to create new nodes without relying on a component to be added through the X3D specification approval process. Prototypes allow developers to define fields and even embed scripts which means they can be helpful for creating customised X3D objects when needed.

First a prototype definition is declared (ProtoDeclare) which is used to build a new node from other nodes. The prototype definition is then used to create prototype instances (ProtoInstance). Each new prototype instance is traversed and processed in the X3D browser run-time. The prototype declarations can be defined in the X3D file or reused if defined in a separate external file. If the prototypes are declared externally they can be retrieved when necessary using the the ExternProtoDeclare statement.

A prototype declaration defines a prototype interface and prototype body. The interface defines the fields while the body defines the nodes, which are instantiated when a prototype is created. The fields have the same kind of properties as X3D node fields such as the type of field, access type, and if any default value is required. The prototype nodes can have the same kind of functionality as that of built-in X3D nodes.

## 3.4 Resources

We now briefly look at some X3D browsers, digital content authoring tools, developer toolkits and libraries, and finally, file translators and utilities.

### 3.4.1 X3D Browsers and Plug-ins

The interpretation, execution, and presentation of X3D scenes occurs in an X3D browser. Figure 3.9 shows a representative example of the architecture of an X3D browser [265]. The X3D browser either reads or writes the scene whereupon a

parser interprets the file format of the scene. Nodes are next created and sent to the scene graph manager which then draws these nodes as images with appropriate geometry, appearance, positioning, and orientation. The scene graph can receive events by animation or scripting nodes which can change the values of the rendered images or manipulate the geometry in the scene. The Scene Authoring Interface (SAI)<sup>6</sup> defines how the scripting code works which allows developers to create code that can work across different operating systems and browsers. Finally, X3D plug-in browsers can be embedded into HTML web pages or widely-used web browsers.

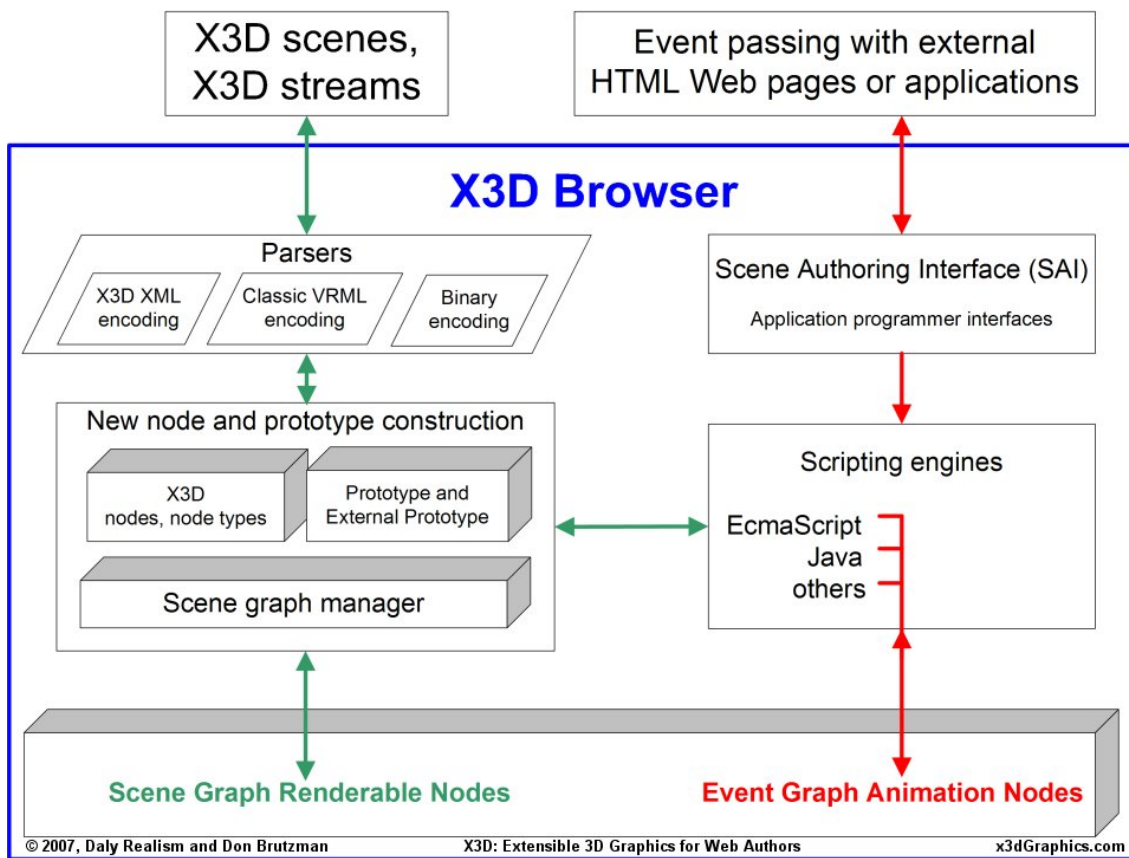


Figure 3.9: X3D Browser Software Architecture [265].

Three X3D browsers that operate on the Windows platform and can be plugged into Microsoft Internet Explorer or Mozilla Firefox include the Octaga Player, Flux Player, and BS Contact VRML/X3D Player. The Octaga Player<sup>7</sup> (5MB download) is a high-performance, standards-compliant viewer created by Octaga Incorporated. The Octaga Player Acrobat plug-in for Windows can display X3D models embed-

<sup>6</sup>The Scene Authoring Interface (SAI) is used either for manipulating the browser and the scene graph from an external application or from inside the scene graph through the X3D script node.

<sup>7</sup><http://www.octaga.com>

ded in PDF documents. Figure 3.10 shows an example animation of a whale shark using the Octaga Player. The BS Contact VRML/X3D Player<sup>8</sup> (6MB download) is a high-performance X3D player created by Bit Management. Figure 3.11 shows an example of the Golden Gate Bridge in San Francisco using the BS VRML/X3D Contact Player. The Flux Player<sup>9</sup> [180] (1.5MB download) is a plug-in produced by Media Machines. Figure 3.12 shows a person running from one side of the screen to the other and then jumping using the Flux Player.

The BS Contact VRML/X3D Player provides features which the other players do not, for capturing screen shots (.bmp and .jpeg formats) and creating videos (.avi format). All three browsers have slightly different names for the same user control and navigation options, and each display their tool-bars in a different places inside the stand-alone browser. Only the Flux player displays the tool-bar inside a web browser while the others let a user right click to get to the tool-bar. All three browsers also render VRML and have professional versions of their browsers. Being able to render VRML files as well as X3D is important as there are a number of existing VRML software visualisations which software developers would also like to be able to reuse or refer to when creating X3D software visualisations.

Xj3D<sup>10</sup> (12MB download) is the Web3D Consortium's open source project which is a toolkit for writing VRML and X3D content and is developed in Java. The current version is 1.0 and was released in April 2006. Xj3D has its own stand-alone web browser implemented in Java and does not have a plug-in for Microsoft Internet Explorer or Mozilla Firefox, which is a part of our software visualisation requirements for integration with our VARE architecture project (§2.3). FreeWRL<sup>11</sup> is an open source VRML and X3D browser that operates on Linux and MacOSX. We have not evaluated either Xj3D or FreeWRL browsers for this project as we want the X3D browsers to be able to be plugged into either Microsoft Internet Explorer or Mozilla Firefox, and operate on a Windows operating system.

### 3.4.2 Content Authoring and Editing Tools

Since X3D has text encodings, basic text editors can be used to create and edit X3D documents. There are also various open source and free tools specifically designed for editing X3D including: X3D-Edit, Flux Studio, and Seamless 3D.

The X3D-Edit Authoring Tool [40]<sup>12</sup> is a GUI application that operates as a

---

<sup>8</sup><http://www.bitmanagement.de>

<sup>9</sup><http://www.mediamachines.com>

<sup>10</sup><http://www.xj3d.org>

<sup>11</sup><http://freewrl.sourceforge.net>

<sup>12</sup><https://savage.nps.edu/X3D-Edit/>

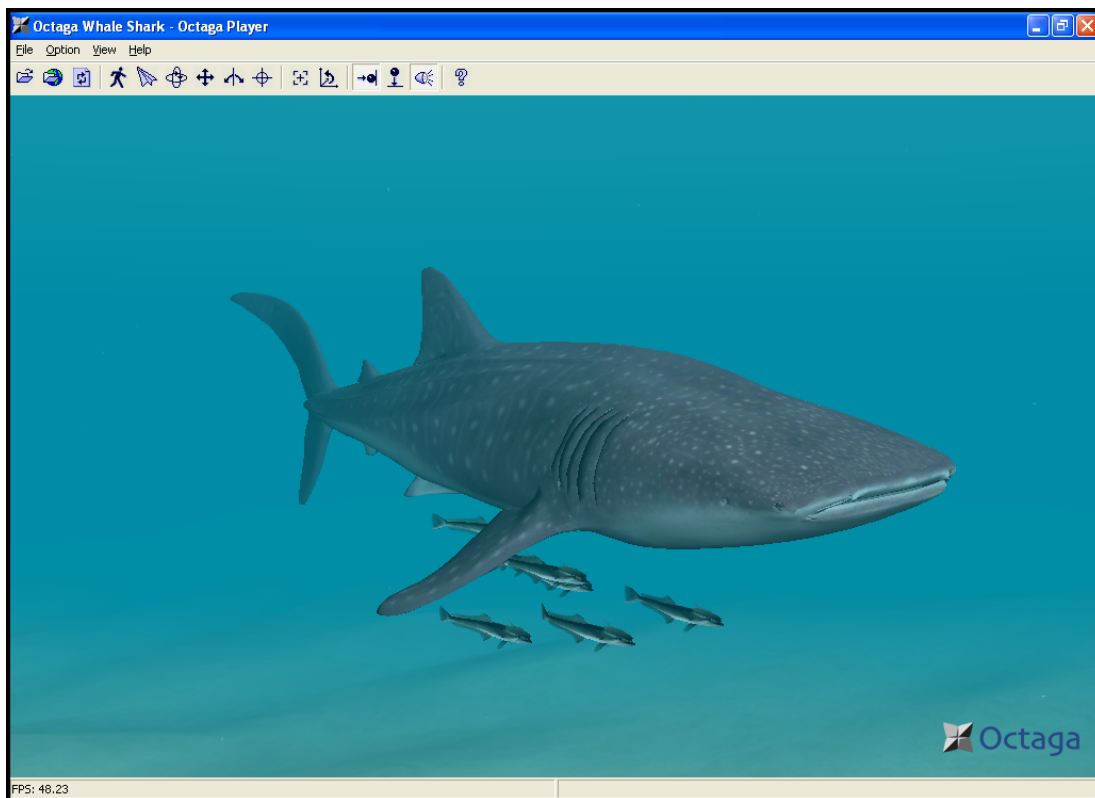


Figure 3.10: Octaga X3D Player Example — animating whale shark, courtesy of Marko Steffensen from Octaga (<http://www.octaga.com>).

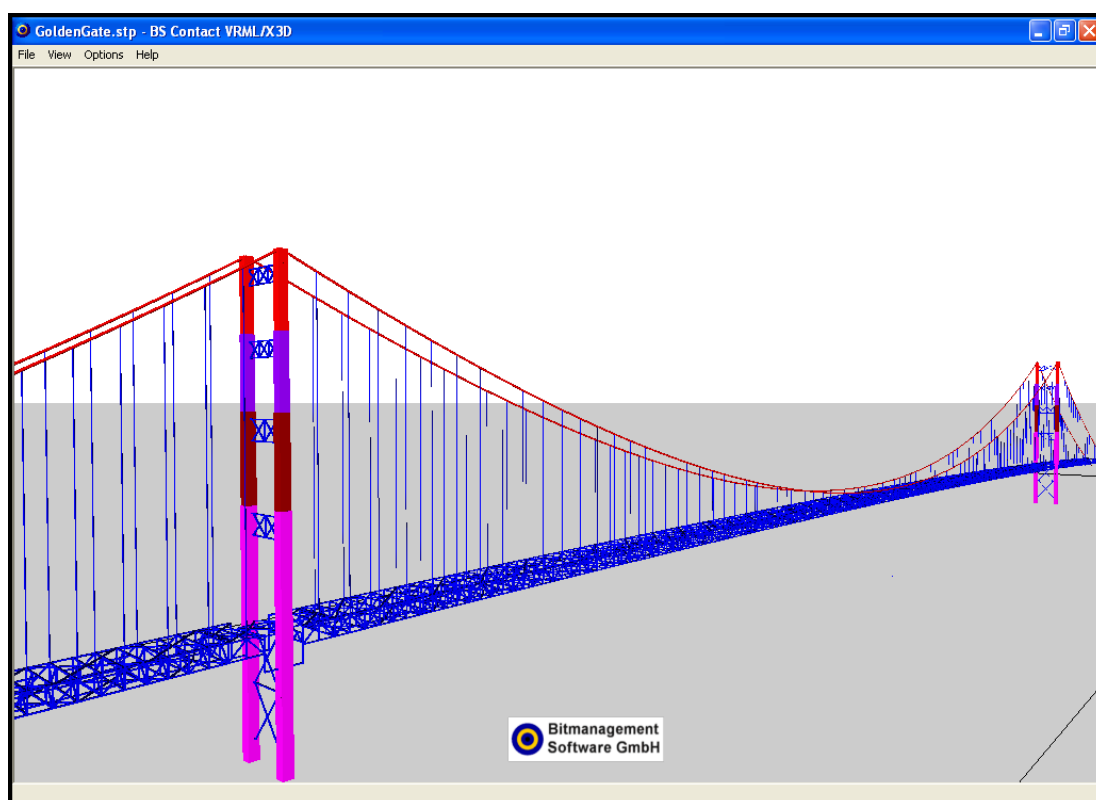


Figure 3.11: BS Contact VRML/X3D Player Example — Golden Gate Bridge, courtesy of the National Institute of Standards and Technology (NIST) (<http://www.nist.gov>).

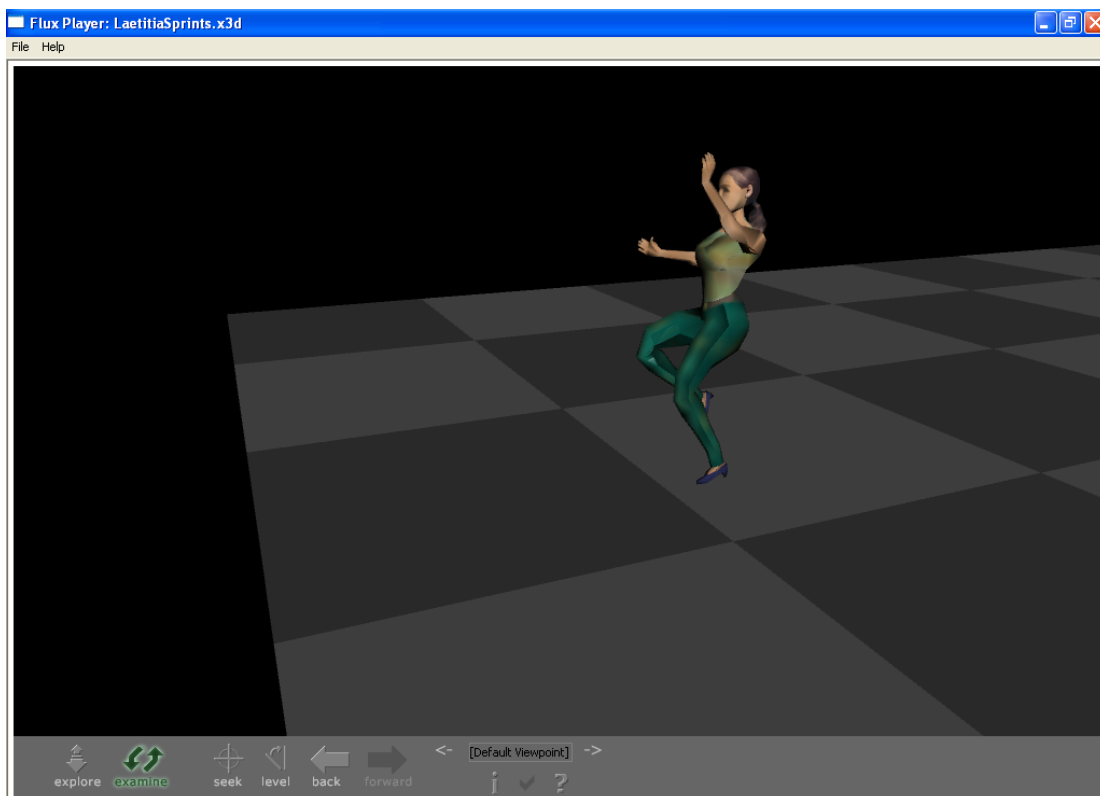


Figure 3.12: Flux Player Example — person running and then jumping, courtesy of Yilmaz Degirmenci from the US Naval Postgraduate School (<http://www.nps.edu>).



plug-in to NetBeans<sup>13</sup>. Flux Studio [180] is a GUI application which has a design interface to graphically see what you are doing rather than working with raw XML. Flux Studio saves the data in its own format. Flux allows multiple views of the same drawing, exporting to X3D, and quick preview of your drawing in X3D using the Flux Player. Figure 3.13 is a screen shot of X3D-Edit and Flux Studio editing an X3D document. There are also some commercial X3D editing tools including: Octaga Professional, Wirefusion, SwirlX3D, and NSS X3D Modeler.

### 3.4.3 Export and Translator Tools

Some common digital content creation tools not designed for X3D have plug-ins or features that export their native file format into X3D. These tools include: 3D Studio Max, Maya, Blender, MilkShape 3D, SoftImage XSI, Modo, and AC3D. Some of these digital content creation tools can even write and import X3D files. A few browser companies have supplied exporter tools to digital content creation tool companies for producing X3D that is compatible with their browsers such as Octaga Exporter with 3D Studio Max. The advantage of using digital content creation tools is that they allow people who are not programmers or have no computer science or programming backgrounds to create X3D content.

There are translator tools for converting other files into X3D files including the 3D Object Converter, KML2X3D (Keyhole Markup Language which is used as the modeling format for Google Earth), and UnrealToX3D [13] (converts Unreal Tournament computer game scenes).

There are many existing software visualisations implemented in VRML. Since X3D has replaced VRML it is important to preserve these software visualisations so that they can still be viewed. Any file encoded in the the VRML97 standard should also work with X3D since X3D is backwards compatible. There are however some tools for converting VRML files into X3D files. There is a stand-alone Java-based software package for translating VRML files into X3D files and vice-versa created by the National Institute of Standards and Technology (NIST)<sup>14</sup>. Xj3D has a separate command line translator independent of the Xj3D browser for converting files both ways.

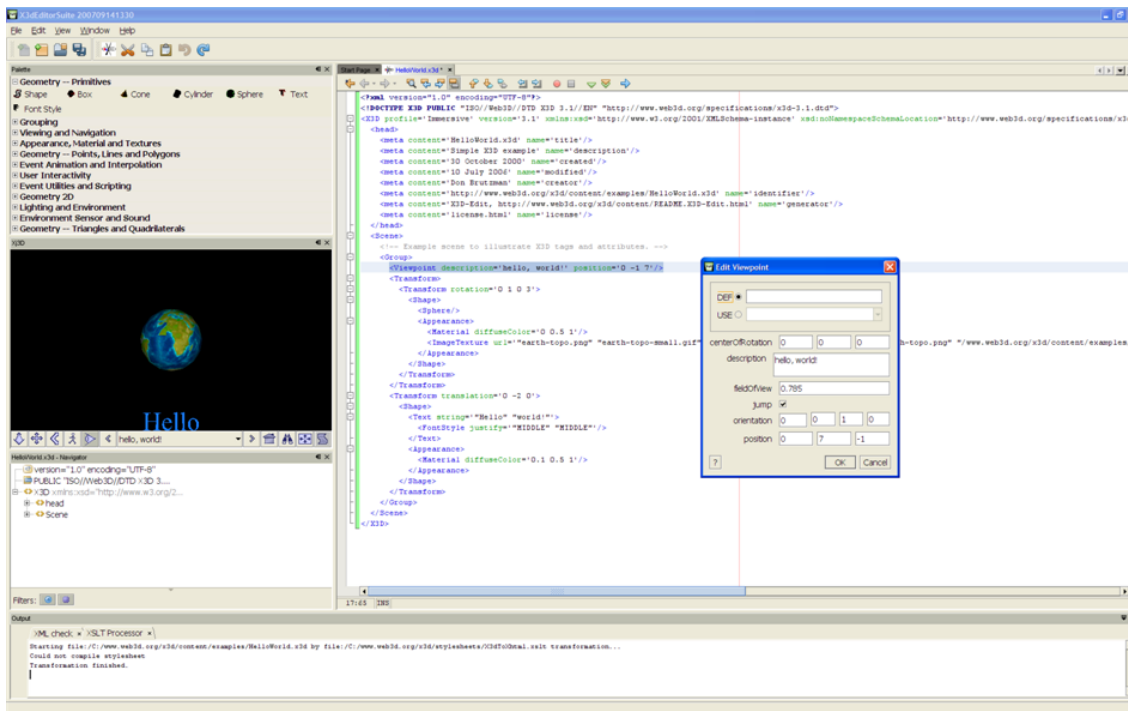
## 3.5 Discussion

There are various issues to discuss regarding the X3D language in terms of improvements to the specification, and the future of X3D on the web.

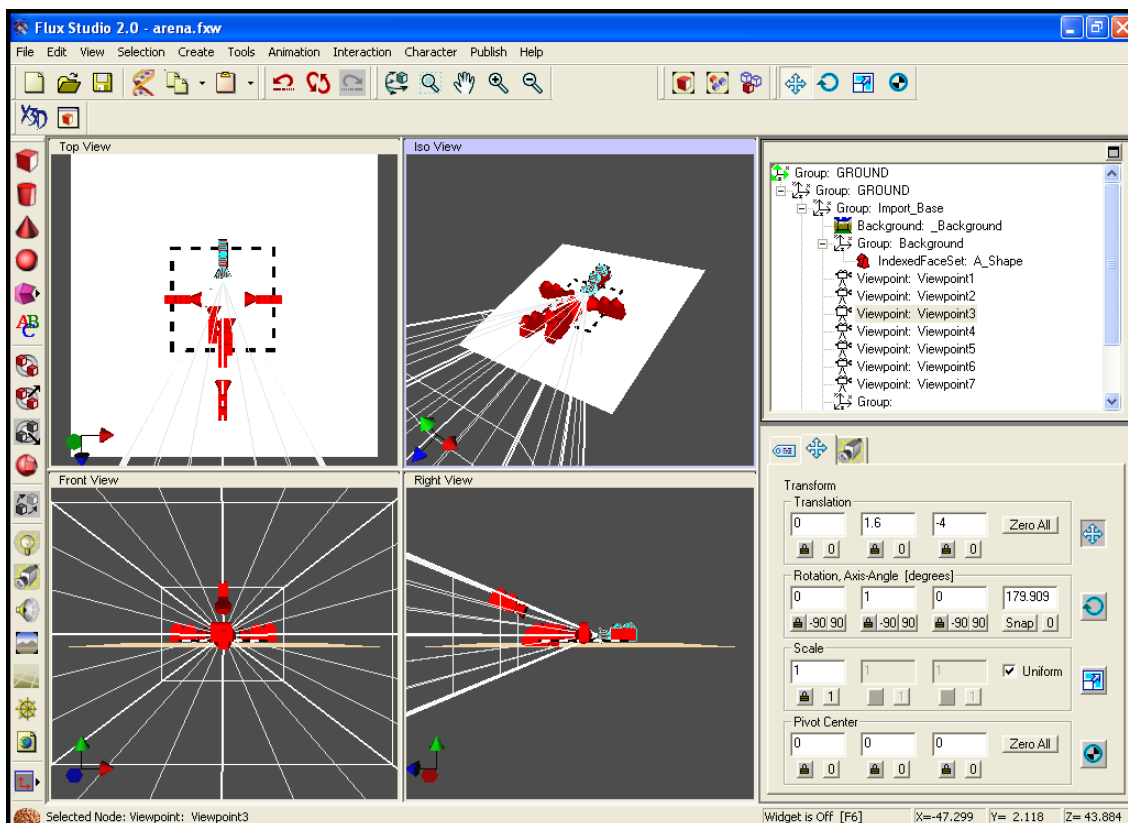
---

<sup>13</sup><http://www.netbeans.org>

<sup>14</sup><http://ovrt.nist.gov/v2.x3d.html>



(a) X3D Edit [40].



(b) Flux Studio [180].

Figure 3.13: X3D Editing Tools.

### 3.5.1 Improvements to the Specification

The Web3D Consortium has several working groups that research and propose solutions to specific technical problems relating to X3D. Some of these groups are explored next.

An important part of X3D is being able to network with other applications or services hosted on remote servers. A couple of approaches already exist including a web browser supported JavaScript approach and an internal script node networking approach. The first approach uses Ajax3D<sup>15</sup> where the SAI (§3.4.1) is used to interface with the X3D run-time system and JavaScript or the DOM is used to pass information between a 3D world and a networked server application. The disadvantage of this approach is that it relies on third party applications and protocols, can have a significant performance overhead, and a potentially high JavaScript code maintenance overhead. The second approach uses either a Java or C++ class inside a script node to communicate via the network. The disadvantage of this approach is that X3D browsers and other X3D applications must have built-in support for these languages. The advantage of the first approach (Ajax3D) is for low frequency communications, while the second approach (internal script node) is best for the more demanding high frequency, throughput, payload, and low lag communication.

The aim of the X3D networking group is to strengthen the open standards networking capability for X3D. This group has proposed a different approach to the previous ones above, which uses a direct networking approach where a node interface is used to directly connect the X3D event system to the network. The advantage of this approach is that it minimises the use of third party applications and protocols and avoids the portability problems of using different programming languages. There is also a low performance overhead for high frequency communications such as required for movement tracking in games or exploration in 3D software visualisations.

There is a conformance program whereby X3D implementations (X3D browsers and tools) are assessed to see if they meet certain X3D standards before they can claim to be true X3D implementations or use the Web3D Consortium's X3D trademarks. The conformance working group manages this process. The advantage of this group is that it encourages vendors to implement X3D browsers and tools that are consistent with the X3D specification.

Other groups range from supporting general UI functionality in X3D content, creating an open source X3D-Earth application that uses 3D spatial data to model

---

<sup>15</sup><http://www.ajax3d.org>

plant earth (an earlier open source X3D project is Planet Earth<sup>16</sup> [249]), CAD engineering, programmable X3D shaders, modeling 3D human figures, representing human anatomy for surgical training, patient education and medical visualisation & modeling. Finally, there is a group focused on developing implementations and utilities for the X3D specification.

### 3.5.2 Future of X3D on the Web

There are various reasons why the predecessor to X3D, VRML, failed to become a popular open standard for 3D on the web. For X3D to survive and to be a success on the web there must be a number of applications implemented with X3D which are used by the greater public.

Many 3D web applications are becoming extremely popular with the wider computer community [71]. Some large scale example applications include Google Earth<sup>17</sup> for 3D maps and Second Life<sup>18</sup> [223] for online virtual worlds. Neither of these applications use X3D nor do they use the same standard.

The potential of 3D environments is attracting major attention and investment. IBM<sup>19</sup> recently announced that it will invest US\$100m over two years to pursue 10 ideas that came from a collaborative innovation brainstorming session. One of these projects is to take the best of virtual worlds and gaming environments to build a standards based 3D Internet. In order for a 3D Internet to work, all devices, platforms and connection speeds would need some open standard protocols that will operate efficiently over TCP/IP and UDP with additional support. Sun Microsystems also have a project called Wonderland<sup>20</sup> which is a 3D scene manager for creating collaborative virtual worlds.

Rasmus [100] investigated why 3D technologies for the web are not widely used on the Internet today. The focus of the report was on end user product and service web sites. Rasmus concluded that for 3D technologies to be successful, the web development industry needs to focus more on the people who are going to use the web sites rather than the technology itself.

In our research we are not focusing on end user web sites for the general user population. Instead, we are interested in seeing if X3D would be a viable solution for use within our software visualisation architecture. Our software visualisations are aimed at software developers who we expect to be more sophisticated web users than the general web end-user because of their technical background.

---

<sup>16</sup><http://planet-earth.org>

<sup>17</sup><http://earth.google.com>

<sup>18</sup><http://secondlife.com>

<sup>19</sup><http://www-03.ibm.com/press/us/en/pressrelease/20605.wss>

<sup>20</sup><http://lg3d-wonderland.dev.java.net>

The next chapter demonstrates some software visualisation case studies we have built using X3D.



# Chapter 4

## X3D Software Visualisations

### Contents

---

<b>4.1</b>	<b>VARE-3D . . . . .</b>	<b>62</b>
4.1.1	Architecture . . . . .	62
4.1.2	Implementation . . . . .	63
4.1.3	Visualisation Transformation . . . . .	64
4.1.4	Discussion . . . . .	70
<b>4.2</b>	<b>Algorithm Animations . . . . .</b>	<b>72</b>
4.2.1	Shortest Path . . . . .	73
4.2.2	Heapsort . . . . .	77
4.2.3	Elementary Sorting . . . . .	80
4.2.4	Discussion . . . . .	85
<b>4.3</b>	<b>UML Diagrams . . . . .</b>	<b>89</b>
4.3.1	Class Diagrams . . . . .	89
4.3.2	Package Diagrams . . . . .	93
4.3.3	Sequence Diagrams . . . . .	93
4.3.4	Discussion . . . . .	95
<b>4.4</b>	<b>Documentation-Related Visualisations . . . . .</b>	<b>97</b>
4.4.1	Source Code Visualisation . . . . .	98
4.4.2	API Javadoc Visualisation . . . . .	98
4.4.3	Structured Video Visualisation . . . . .	98
4.4.4	Discussion . . . . .	101
<b>4.5</b>	<b>Execution Trace Visualisations . . . . .</b>	<b>103</b>
4.5.1	All Elements From an Execution Trace . . . . .	103

4.5.2	3D Compound Shapes . . . . .	105
4.5.3	3D Metaphors . . . . .	110
4.5.4	Discussion . . . . .	115
4.6	Summary . . . . .	116

---

We want to see if X3D can support a range of 3D software visualisation techniques to determine if X3D is a viable solution for use in software visualisation. We want to experiment with automatically creating X3D software visualisations over the web, evaluate X3D's animation and interactivity aspects, examine the text, layout, and extensibility features, test the integration capabilities of X3D, and finally analyse the performance display capabilities.

In this chapter we replicate a representative sample of the most common software visualisation techniques throughout the software visualisation literature [237]. We first describe our prototype tool for producing X3D software visualisations over the web. We then describe our representative case studies for algorithm animations, UML diagrams, documentation-related visualisations, and visualisations from execution traces.

## 4.1 VARE-3D

We are interested in evaluating if X3D is able to create and display visualisations over the web for developers to understand software components for software reuse, maintenance, and re-engineering. We have developed a light-weight prototype web-based software visualisation tool called VARE-3D [11]. VARE-3D implements a transformer component from our visualisation software architecture (§ 2.3). VARE-3D follows on from other lightweight web-based tools that colleagues of our research group have built for visual applications [94], sequence diagrams [120], class diagrams [148], UML SVG diagrams [75], and execution trace visualisations of sequence and class association diagrams [162]. We next describe the architecture, implementation, and visualisation transformation process of VARE-3D. Finally, we finished with a discussion of our prototype tool.

### 4.1.1 Architecture

Figure 4.1 displays the VARE-3D architecture which is based on the VARE architecture (§2.3). Users make queries from a web browser. Users can test drive software components by specifying a sequence of method invocation and field access/modifications and then execute the sequence on a component (§2.3.4). The output of a



test drive is an XML execution trace (§2.3.3) which is stored in a database. Users can transform XML execution traces using XSLT into X3D software visualisations and store the visualisations in the same database but a different namespace. Users can then display the X3D software visualisations in a web browser that has an X3D browser plug-in.

The XML execution traces contain static and dynamic information of software components including the events that happened during the execution of a component (§2.3.3). Separating the test driving and the creation of visualisation steps allows users to test drive components and then create visualisations in the future. Users can also view stored X3D visualisations without having to test drive remote software or transform XML execution traces.

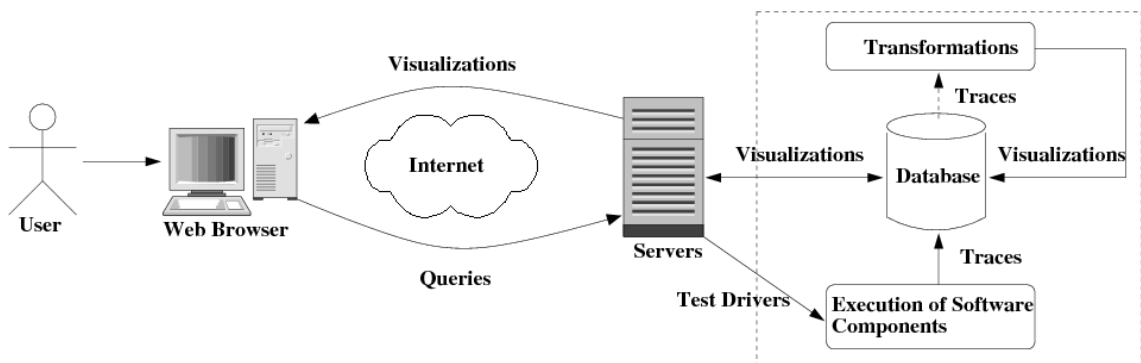


Figure 4.1: VARE-3D architecture.

### 4.1.2 Implementation

In VARE-3D we use the BS Contact Player, Octaga Player, and Flux Player plug-ins (§3.4.1), which are all compatible with both Mozilla Firefox and Internet Explorer for displaying the X3D software visualisations. The Apache Xalan<sup>1</sup> XSLT processor is used for transforming the XML execution traces into X3D software visualisations. Apache Tomcat<sup>2</sup> is used to host the JavaServer Pages (JSP) and Java Servlet web front end.

The current implementation status of VARE-3D allows visualisations of traces but the prototype is not yet fully integrated with the architecture. The main page of VARE-3D consists of a JSP page that has a web form where a user can input a data source (XML execution trace file) and select a visualisation type (stylesheet). When the visualisation creation web form is submitted the Xalan servlet (XSLTServletWithParams) is executed which takes an XML and XSL file as

<sup>1</sup><http://xalan.apache.org>

<sup>2</sup><http://tomcat.apache.org>

input and transforms the XML file into an X3D software visualisation. Once the servlet has successfully transformed the XML file, the X3D software visualisation file is then delivered to the client and displayed in the web browser.

The input XML files are either Reusable Component Descriptions (RCD) or eXtensible Execution Traces (XTE) (§2.3.3). Since we have not integrated VARE-3D with a test driving component we use our existing visualisation tools (§2.3.4) to generate our XML execution trace files. The visualisation types in VARE-3D are only applicable to certain types of execution traces, for example UML class and package diagrams for static RCD description files (§4.3) and execution trace visualisations for dynamic XTE execution traces (§4.5).

Figure 4.2 shows the VARE-3D visualisation creation interface where the user has input the EclipseXTE.xml execution trace file and selected the 3D shapes visualisation type. The EclipseXTE.xml execution trace file contains all the events from a test drive of the Eclipse IDE Java application. The events represent creating an object, method calls, method returns, field accesses, and field modifications.

Figure 4.3 shows a simple 3D shapes visualisation which is the output from the web form submission of Figure 4.2. The visualisation shows 10,000 events from the Eclipse IDE Java application, where each event is represented as different 3D shapes and displayed in a sequence along the X axis. Blue spheres represent object creation events, green boxes method calls, white cones method returns and end of an object, field accesses cyan cylinders, and field modifications as pink cylinders. The different shapes show the sequence of events that happened during the execution of Eclipse and could potentially show interesting behaviour patterns such as one object dominating the application.

We are yet to implement the control components from the VARE architecture or integrate a test driving tool and backend database. VARE-3D currently stores the XML execution traces and XSL stylesheets on the file system and web server. We have, however, explored using the eXist<sup>3</sup> and Ipedo<sup>4</sup> native XML databases for storing and retrieving XML execution traces in previous work [9, 10]. Storing and retrieving X3D software visualisations with these kind of native XML databases should work similar to storing and retrieving execution traces since they are both XML formats.

### 4.1.3 Visualisation Transformation

The main functionality of VARE-3D is the transformation of the RCD and XTE files into X3D software visualisations. We will now elaborate on this transformation

---

<sup>3</sup><http://exist.sourceforge.net>

<sup>4</sup><http://www.ipedo.com>

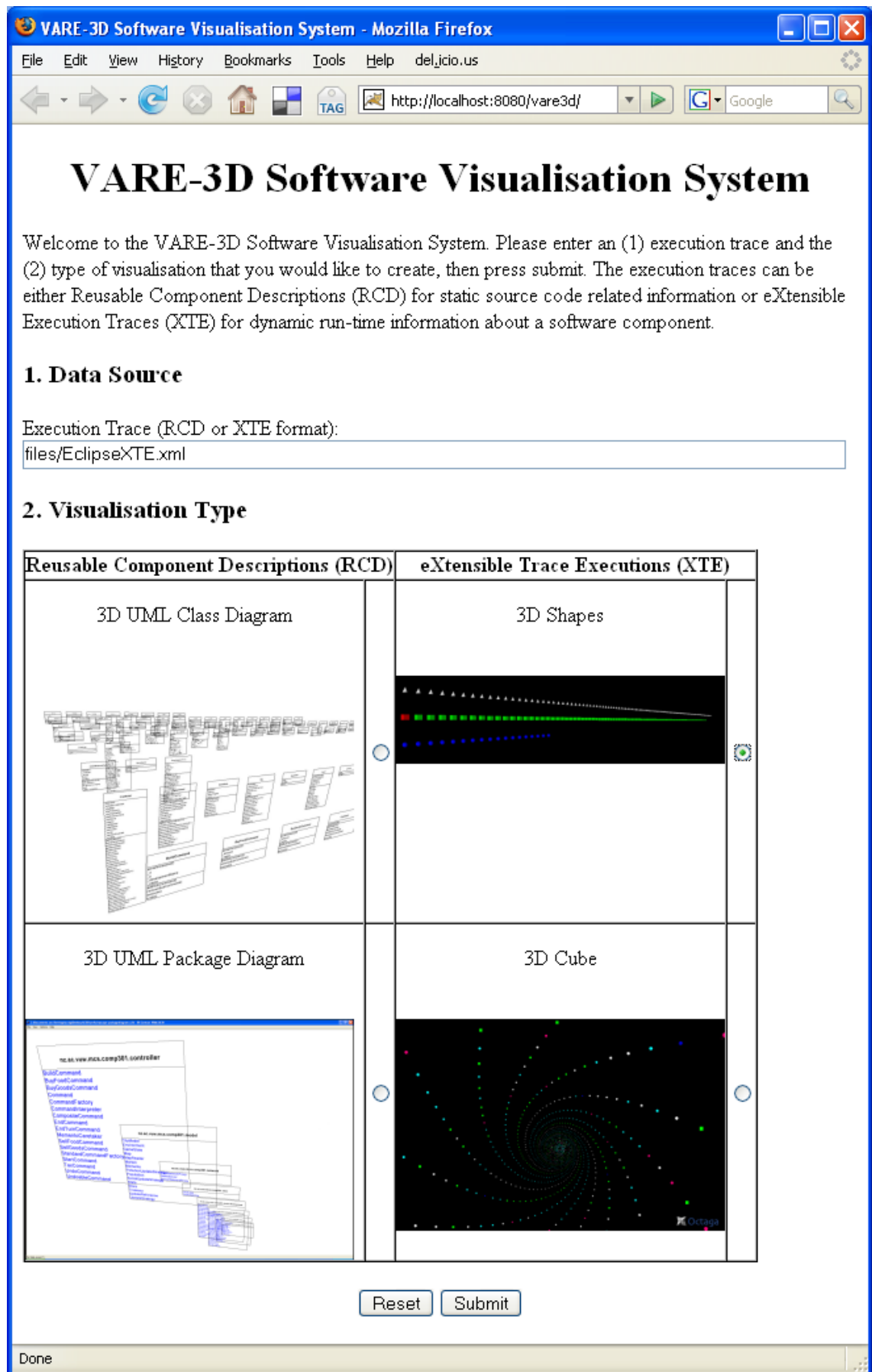


Figure 4.2: VARE-3D — web interface for execution trace X3D software visualisations.

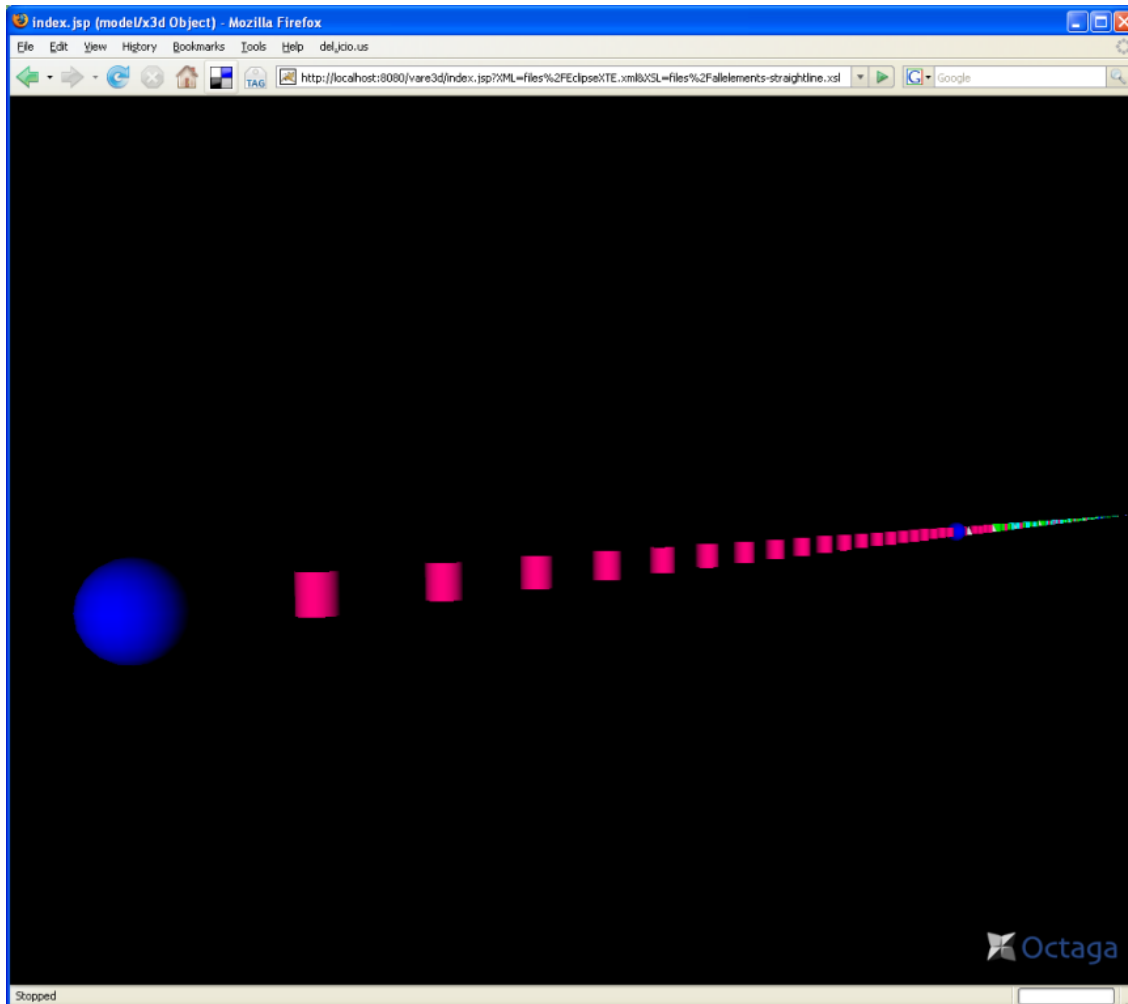


Figure 4.3: Basic X3D Execution Trace Visualisation — in this simple X3D software visualisation 10,000 events have been transformed from the XTE execution trace of the Eclipse IDE Java application by XSLT. Blue spheres represent object creation events, green boxes method calls, white cones method returns and end of an object, field accesses cyan cylinders, and field modifications as pink cylinders.

process. We process the execution traces using our stylesheets and look for specific elements in the traces. For each RCD file we match the following elements: `rcd:packagename`, `rcd:classname`, `rcd:fieldname`, and `rcd:methodname`. We then draw different shapes for each element we match according to the visualisation type selected by a user. For each XTE file we match the following elements: `xte:objectcreation`, `xte:methodcall`, `xte:methodreturn`, `xte:fieldmodification`, and `xte:fieldaccess`. The transformation process is quite straight forward since the execution trace files and the output X3D software visualisations are both in XML format which allows us to use existing XSLT processors.

Figure 4.4 shows an example from one of our XTE execution traces with one object creation (Lines 2-6) and three field modification events (Lines 7-37). Figure 4.5 shows the stylesheet that transforms this XTE execution trace into the X3D software visualisation displayed in Figure 4.3. Lines 1-6 of Figure 4.5 are the standard XSL tags that declare that the stylesheet is an XML file, uses the XSL namespace and the XTE namespace (<http://www.mcs.vuw.ac.nz>), and the output of processing the stylesheet is an XML file. The stylesheet processes the creator, date, and object elements (lines 7-9) in the XTE execution trace, but since these elements are not important for the software visualisation they are not used. This additional data could be included in the metadata header of the X3D software visualisation file if required.

Lines 11-24 show the main body of the stylesheet which matches the `xte:execution` element. Once the XSLT processor reaches the execution element it outputs the X3D DOCTYPE, X3D root, scene, navigation info, viewpoints, and background tags. These tags need to be wrapped inside XSL text tags so that the processor knows to output them as verbatim text. The critical line in the stylesheet is line 20 which applies all other event templates in the stylesheet.

Lines 26-38 show the object creation template. Line 27-28 outputs the transform node for where the object creation node will be positioned in the X3D scene. Line 29 calculates the X axis value of the translation field based on the position of the element in the execution trace. Line 30 uses zero for the X and Y axes values in the translation field. Lines 31-36 draw a blue sphere for the object creation event and then line 37-38 closes the transform node and object creation template. The other templates in the stylesheet are very similar except that different shapes are drawn. Once the object creation template is complete the next element in the XTE execution trace is matched. Depending on what type of event the next element is the appropriate template is processed.

A similar kind of stylesheet is used to create our more complex visualisations that use prototypes (§3.3.6), see Figure 4.6. In this stylesheet we are creating a UML class diagram where we define a prototype declaration which has the following

```
1 <xte:execution>
2 <xte:objectcreation xte:eventid="1" xte:objectid="obj1"
3   xte:threadid="main">
4 <xte:complextypename><xte:typename>org.eclipse.core.launcher.Main
5   </xte:typename></xte:complextypename>
6 </xte:objectcreation>
7 <xte:fieldmodification xte:eventid="2" xte:objectid="obj1"
8   xte:threadid="main">
9 <xte:fieldname>debug</xte:fieldname>
10 <xte:oldvalue>
11 <xte:primitivevalue><xte:value>1</xte:value>
12 </xte:primitivevalue>
13 </xte:oldvalue>
14 <xte:newvalue>
15 <xte:primitivevalue><xte:value>1</xte:value></xte:primitivevalue>
16 </xte:newvalue>
17 </xte:fieldmodification>
18 <xte:fieldmodification xte:eventid="3" xte:objectid="obj1"
19   xte:threadid="main">
20 <xte:fieldname>bootLocation</xte:fieldname>
21 <xte:oldvalue>
22 <xte:primitivevalue><xte:value>1</xte:value></xte:primitivevalue>
23 </xte:oldvalue>
24 <xte:newvalue>
25 <xte:primitivevalue><xte:value>1</xte:value></xte:primitivevalue>
26 </xte:newvalue>
27 </xte:fieldmodification>
28 <xte:fieldmodification xte:eventid="4" xte:objectid="obj1"
29   xte:threadid="main">
30 <xte:fieldname>installLocation</xte:fieldname>
31 <xte:oldvalue>
32 <xte:primitivevalue><xte:value>1</xte:value></xte:primitivevalue>
33 </xte:oldvalue>
34 <xte:newvalue>
35 <xte:primitivevalue><xte:value>1</xte:value></xte:primitivevalue>
36 </xte:newvalue>
37 </xte:fieldmodification>
```

Figure 4.4: Example Eclipse XTE execution trace.

```

1 <?xml version="1.0"?>
2 <xsl:stylesheet version="1.0"
3 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4 xmlns:xte="http://www.mcs.vuw.ac.nz">
5 <xsl:output method="xml" version="1.0" encoding="utf-8"
6 indent="yes" />
7 <xsl:template match="xte:creator"/>
8 <xsl:template match="xte:date"/>
9 <xsl:template match="xte:object"/>
10
11 <xsl:template match="xte:execution">
12 <xsl:text disable-output-escaping="yes">
13 <![CDATA[ <!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
14 "http://www.web3d.org/specifications/x3d-3.0.dtd">
15 <X3D profile="Immersive" version="3.0">
16 <Scene>
17 <NavigationInfo type=' "EXAMINE"' speed="64"/>
18 <Viewpoint description="Overview" position="112 -5 150"/>
19 <Background skyColor="1 1 1"/>]]></xsl:text>
20 <xsl:apply-templates/>
21 <xsl:text disable-output-escaping="yes">
22 <![CDATA[</Scene>
23 </X3D>]]></xsl:text>
24 </xsl:template>
25
26 <xsl:template match="xte:objectcreation">
27 <xsl:text disable-output-escaping="yes">
28 <![CDATA[<Transform translation=""]></xsl:text>
29 <xsl:value-of select="5*position()" />
30 <xsl:text disable-output-escaping="yes"><![CDATA[ 0 0">
31 <Shape>
32 <Appearance>
33 <Material diffuseColor="0 0 1"/>
34 </Appearance>
35 <Sphere radius="2"/>
36 </Shape>
37 </Transform>]]></xsl:text>
38 </xsl:template>

```

Figure 4.5: XTE execution trace XSL stylesheet.

fields: classname, attributes, operations, height, and width (Lines 1–9).

We first match and apply the package template which creates a group tag and uses the package name as the DEF attribute (Lines 12–17). Next we match all the classes in the package (Lines 19–38) and group each class inside a transform node to position the class in the scene (Lines 20–25). Prototype instances are then created (Lines 26–36) based on the prototype declaration defined earlier in the file. Each prototype instance has the attribute name of the prototype declaration (Line 26, e.g. class diagram, package diagram) and the value for each prototype instance field which is selected from the element in the execution trace (Lines 28–30, e.g. classname field). The fields and methods templates are next processed which get all the fields and methods for each class (Lines 31–32). We then have two fields height and width, that are fixed values and determine the size of the class node prototype instance (Lines 34–35). Finally, we close the prototype instance, transform node for each class, group tag for each package, and package template (Lines 36–41).

Section 4.3 shows how we use the prototype stylesheet in Figure 4.6 to produce the X3D prototype file in Figure 4.18. Figure 4.16 and Figure 4.17 show the eventual X3D software visualisations generated from this stylesheet and prototype generation.

#### 4.1.4 Discussion

The design of VARE-3D makes it easy to add new visualisation types by simply adding a new stylesheet to the system without recompiling any parts of the system. The only part that requires changing is the web page that allows a user to upload a visualisation type and the subsequent page that lets a user select this new visualisation type. We are yet to implement this user visualisation control feature and other VARE architecture controls and test driving components.

To implement these features the next development steps would be to integrate our XML database component which we have documented elsewhere [9, 10] and some test driving tools (§2.3.4). The database component will be of value to an end user as they will be able to see what execution traces exist, look at the contents of the execution traces (if applicable), upload their own visualisation types and display X3D software visualisations previously created without relying on a filesystem. Therefore, integrating both of the database and test driving components will help provide a complete end to end software visualisation system for a user.

The VARE-3D visualisation transformation process allows execution trace data to be encoded using two different approaches. The first approach (Figure 4.5)



```

1 <ProtoDeclare name="ClassDiagram" url="x3d-classdiagram.x3d">
2   <field accessType="initializeOnly" name="classname" type="MFString"/>
3   <field accessType="initializeOnly" name="attributes"
4     type="MFString"/>
5   <field accessType="initializeOnly" name="operations"
6     type="MFString"/>
7   <field accessType="initializeOnly" name="height" type="SFFloat"/>
8   <field accessType="initializeOnly" name="width" type="SFFloat"/>
9 </ProtoDeclare>
10 <xsl:apply-templates/>
11
12 <xsl:template match="rcd:package">
13   <xsl:variable name="yposition" select="-20*position()" />
14   <xsl:text disable-output-escaping="yes">
15     <![CDATA[<Group DEF="]]></xsl:text>
16   <xsl:value-of select="rcd:packagename"/>
17   <xsl:text disable-output-escaping="yes"><![CDATA[">]]></xsl:text>
18
19   <xsl:for-each select="rcd:class">
20     <xsl:text disable-output-escaping="yes">
21       <![CDATA[<Transform translation="]]></xsl:text>
22     <xsl:value-of select="20*position()" />
23     <xsl:text disable-output-escaping="yes"><![CDATA[ 0 ]]></xsl:text>
24     <xsl:value-of select="yposition"/>
25     <xsl:text disable-output-escaping="yes"><![CDATA[">
26     <ProtoInstance name="ClassDiagram">]]></xsl:text>
27     <xsl:text disable-output-escaping="yes">
28       <![CDATA[<fieldValue name="classname" value="]]></xsl:text>
29     <xsl:value-of select="rcd:classname"/>
30     <xsl:text disable-output-escaping="yes"><![CDATA["/>]]></xsl:text>
31     <xsl:apply-templates select="rcd:fields"/>
32     <xsl:apply-templates select="rcd:methods"/>
33     <xsl:text disable-output-escaping="yes"><![CDATA[
34     <fieldValue name="height" value="15"/>
35     <fieldValue name="width" value="15"/>
36     </ProtoInstance>
37     </Transform>]]></xsl:text>
38   </xsl:for-each>
39   <xsl:text disable-output-escaping="yes">
40     <![CDATA[</Group>]]></xsl:text>
41 </xsl:template>

```

Figure 4.6: UML Class Diagram XSL prototype stylesheet.

transforms an execution trace into X3D geometry that uses JavaScript functions for layout, while the second approach (Figure 4.6) uses prototypes (§3.3.6). Each approach has its merits depending on the type of visualisation that is displayed. Section 4.5 uses the X3D geometry JavaScript function approach while Section 4.3 uses the prototypes approach. In Section 4.2 we investigate how we can encode the data inside the actual software visualisation without relying on JavaScript functions for layout nor prototypes for data display independence.

VARE-3D operates on both Mozilla Firefox and Microsoft Internet Explorer. Our strategy for doing all of the visualisation transformation in XML allows us to leverage high powered XML tools such as the Apache Xalan XSLT processor and Java XML parsing libraries. Our smaller execution traces take less than a few seconds<sup>5</sup> to generate a X3D software visualisation while our larger traces (10-50MB) take less than two minutes to produce 10,000-100,000 nodes. The longest time spent by VARE-3D for creating a software visualisation is actually rendering the end software visualisation rather than the stylesheet transformation. It takes less than 10 seconds to render about 10,000 nodes, three minutes for 50,000 nodes, but up to 10 minutes to render 100,000 nodes.

The next sections discuss some of the X3D software visualisations that we have created, some via VARE-3D and others manually. We want to see how viable X3D is for use in software visualisation. In particular we want to evaluate X3D's animation and interactivity aspects, examine the text, layout, and extensibility features, test the integration capabilities of X3D, and analyse the performance display capabilities.

## 4.2 Algorithm Animations

In this section we explore the animation and interaction capabilities of X3D for algorithm animation. We replicate a variety of 3D algorithm animation techniques from the literature. Our X3D algorithm animations are manually hand crafted as we want to see if X3D can support creating complex software visualisations by hand and encode the data inside the visualisation.

According to Brown and Najork [33] 3D graphics for algorithm animation is significant and mostly unexplored. Brown and Najork [33, 168, 169] were the first to identify several reasons for integrating 3D graphics into an algorithm animation system. The third dimension can be used for expressing fundamental information about structures that are inherently 2D, uniting multiple views of an object, and

---

<sup>5</sup>Performance transformation timings of both client and server were measured using a Dell 610 laptop with Windows XP and 512MB of RAM.

capturing the history of a 2D view. There also exists some empirical evidence that 3D algorithm animation promotes understanding of distributed algorithms [252].

We now show three replicated example algorithm animations from Brown and Najork [33, 168, 169] hand crafted in X3D which unite multiple views, capture the history of execution, and display additional information. We also show how we have extended each of the examples as well.

### 4.2.1 Shortest Path

Given a directed graph with weighted edges, Dijkstra's Shortest Path Algorithm is used to calculate the shortest path from a starting node to all other nodes in a graph. The traditional 2D version of this algorithm animation struggles to show state information about the cost for vertices and weight of edges.

To explore the details of the X3D animation capabilities we have created a version of Brown and Najork's example. In this example and others that follow we hand crafted the X3D algorithm animations. Figure 4.7 shows our shortest path algorithm animation implemented in X3D. The shortest path are the green edges, while explored edges are purple and unexplored edges are white. Columns are used to represent the cost of getting to each vertex from the starting vertex A.

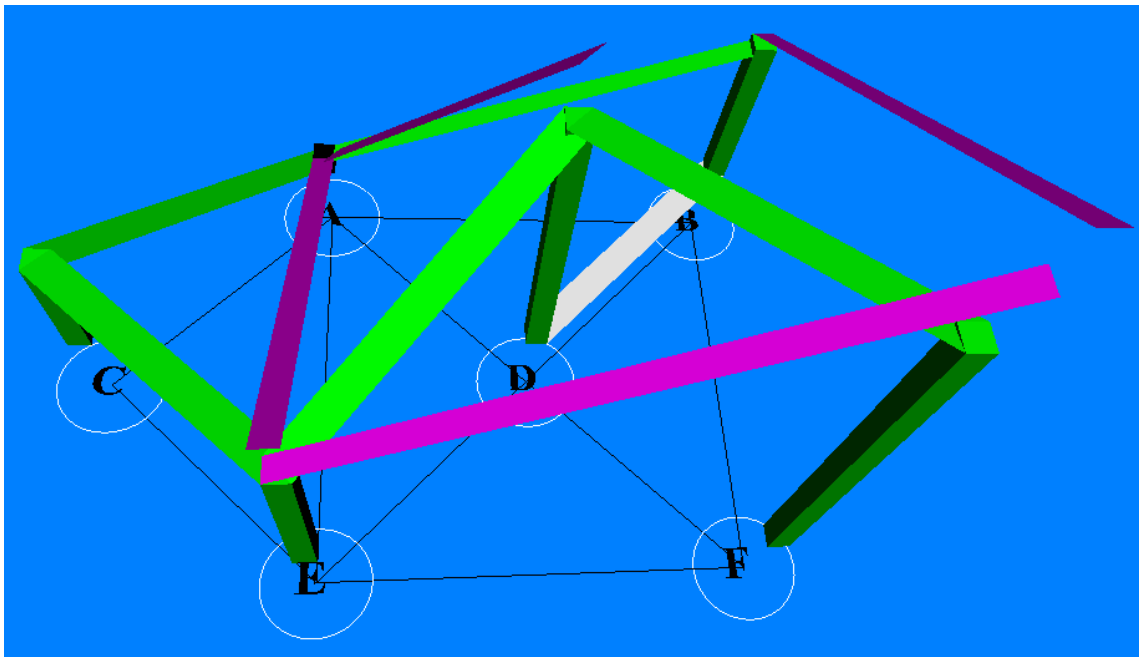


Figure 4.7: X3D Shortest Path Algorithm Animation — shortest path from vertex A to all other vertices. Hand crafted in X3D following the strategy of Brown and Najork [33, 168, 169].

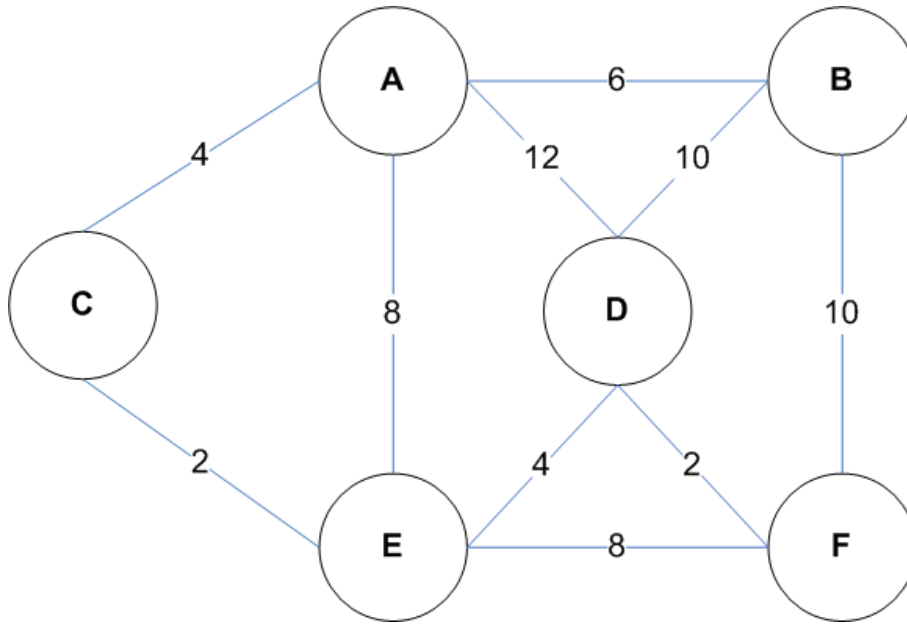


Figure 4.8: Shortest Path Algorithm — initial 2D state.

Brown and Najork [33, 168, 169] provide additional state information about the cost of vertices and weight of edges in 3D. Columns are used to represent the cost of getting to each vertex. An edge from  $u$  to  $v$  with weight  $w$  leaves the column above  $u$  at height 0, but the weight is added to the column above  $v$  at height  $w$ . Whenever an edge is examined from  $u$  to  $v$  a highlighted copy of the edge is lifted to the top of  $u$ 's column, hence its tip will hover over  $v$  at height proportional to  $\text{cost}(u) + \text{weight}(\text{edge})$ . If  $v$ 's column is taller, the edge can lower  $v$ 's cost, so  $v$ 's column is shortened. If the column is not shortened then the highlighted edge disappears.

The third dimension in this animation provides state information about the cost of vertices and weight of edges. Animation is used to show fundamental operations of the algorithm: lifting an edge represents addition, lowering a highlighted edge indicates the outcome of a comparison, and shortening a column shows assignment. Cox and Roman [63] produced a similar animation of the shortest path algorithm.

We extended this example by displaying the 2D graph from Figure 4.8 beneath the the columns and edges in the animation from Figure 4.7. We felt it was important to somehow use text for labeling the vertices to help a user understand the animation. When a user rotates the view or a much larger graph is used for the animation it is hard to determine which vertex a particular node or column is. The 2D graph is implemented using 2D circles and polylines. We found that not all of the X3D browsers we tested this example on have implemented the 2D geometry

component of the X3D specification. Hence, the 2D graph did not always display depending on the X3D browser.

Figure 4.9(a) shows the initial state of the animation where all the edges are drawn in white and vertex A represented as a black box above the 2D graph, and viewed looking down upon the animation. Figure 4.9(b) shows the algorithm starting where the cost of each node is first noted. The height of column C is four, E eight, D twelve, and B five. At this stage vertex F is unreachable from A. Next, the cheapest node from the starting node not yet visited is selected. If there is another node which can be reached via this selected node that is cheaper than before then the cost to reach that node is updated in the cost table. Figure 4.9(c) shows the cost from A to E has been reduced from eight to six by the path A-C-E. The algorithm then iterates to the next cheapest node and so on. Figure 4.9(d) shows an edge being examined (E-D) for the cost to get to D via E and C. Figure 4.9(e) shows the completed shortest path as green edges, and all the purple edges that are not part of the shortest path. Figure 4.9(f) shows a top down view once the shortest path algorithm has completed.

We decided to leave the purple and white edges in the animation as it helps show what paths existed before the shortest path and the edges that were not explored. Brown and Najork [33, 168, 169] chose to remove the redundant edges. If the example was a much larger graph then we may actually provide filter controls for a user to remove the purple and white edges to show the different paths. Removing the edges was not a critical step in this algorithm animation as we were focusing on the X3D implementation rather than the algorithm itself. Most 2D versions of this animation that we know of leave the redundant edges in.

We implemented the columns and edges as indexed face sets (§3.3.1) which is a node that represents a 3D shape formed by constructing faces (polygons) from vertices listed in the coordinate field (coordIndex) of the indexed face set node. The indexed face set contains a coordinate child node that defines the 3D vertices referenced by the coordinate index field. The indexed face set uses the indices in its coordinate index field to specify the polygonal faces by indexing into the coordinates in the coordinate node.

Manipulating and positioning the columns and edges as indexed face sets was easier than using boxes and cylinders. Calculating the length, position, and rotation of a cylinder when used as an edge and when it was either increased or decreased in size required too much effort. Changing the size of the box meant that the other elements linked to the box would also have to be adjusted. We could have used indexed line sets (§3.3.1) for the edges, but this would have only been one pixel in depth and would be hard for an user to see the changing of colour for an edge once it was part of the shortest path.

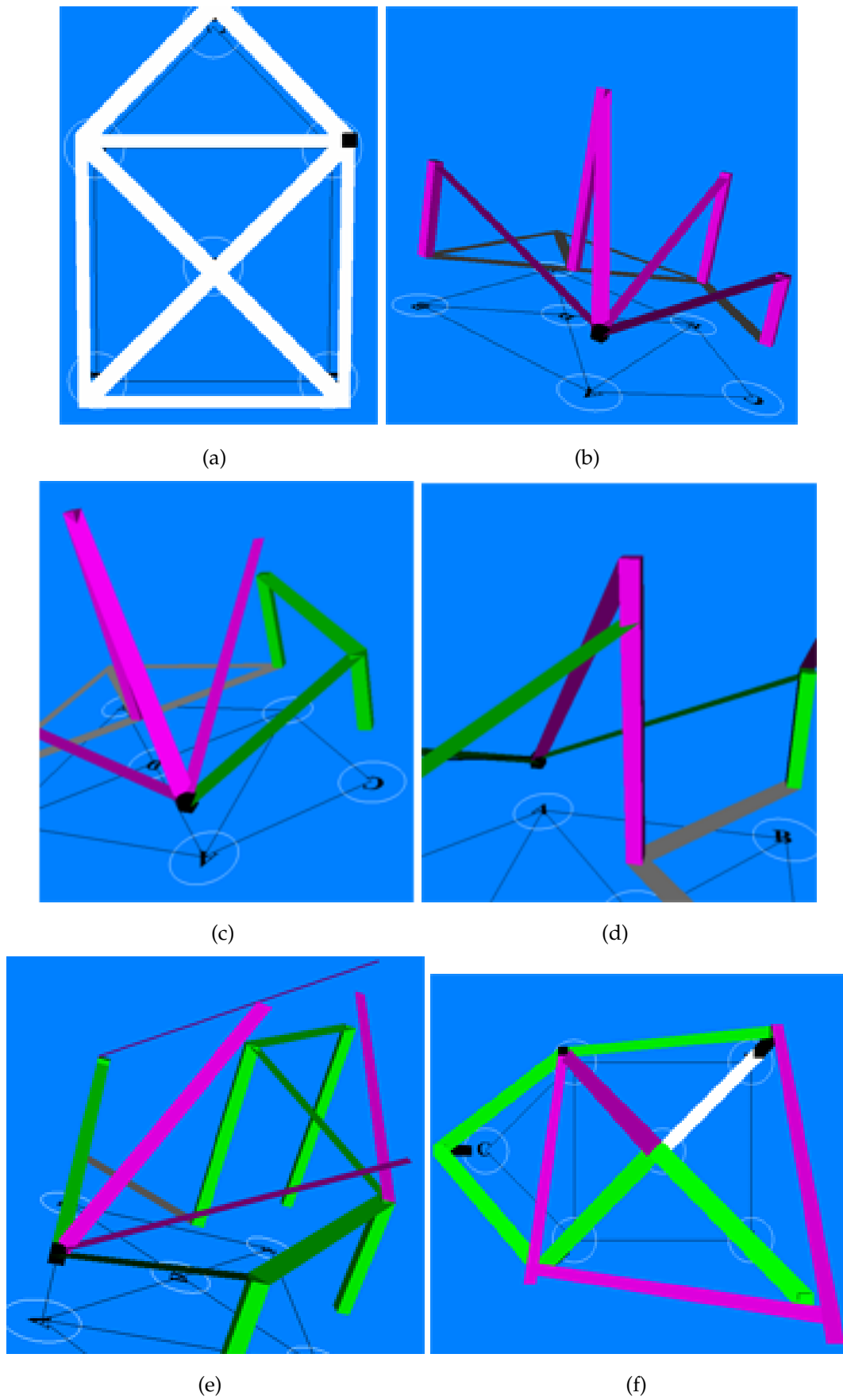


Figure 4.9: X3D Shortest Path Algorithm Animation — different stages of the algorithm.

Our indexed face set columns have eight points while edges have four points. When the cost to a vertex changed the top face of the column was either increased or decreased in the z dimension by modifying four coordinate points. When an edge was changed only two coordinate points were modified. Coordinate points were moved using coordinate interpolators, while the colour of a column or an edge was controlled by colour interpolators. A continuous timer was used to start and keep the animation going, which cycles every 20 seconds.

An alternative way to run this animation is to remove the purple edges once the shortest path to a vertex has been found. In order to accomplish this task we wanted objects to appear in the animation and then later disappear or be deleted altogether. We found no specific nodes for doing this task. Instead, we had to hide objects inside other objects when we did not require them to be visible. We accomplished the invisibility of objects by modifying some of the coordinate points of a node. Alternatively, we could have made a node visually transparent by adjusting the transparency attribute of the material node (§3.3.1). We could have also added another larger node on top of the node to be hidden, but the larger node would have to be transparent to begin with.

### 4.2.2 Heapsort

Finding a single view of an object during an algorithm animation that reveals all of its features can be difficult. Presenting multiple views of that object can be a helpful technique, but interpreting the multiple views may make it hard for users to understand.

The traditional way of displaying the heapsort algorithm animation is to have two views, one showing the sticks like array and the other the heap. Brown and Najork [33, 168, 169] united the multiple views of the heapsort algorithm in 3D to alleviate the problem where users must mentally integrate the different views in order to understand the overall algorithm.

Figure 4.10 shows our X3D animation of the completed heapsort algorithm with the united views of the heap and the array. A node in the tree is an element in the array being sorted, and has depth (Z axis) proportional to its value.

The algorithm works in two parts. The first part orders elements from the data set to be sorted into a heap. The second part picks the largest element in the data set and puts it in its final place in the heap. Then from the remaining unsorted data the next largest element is selected and put in its final place. This continues until the end of the data set.

In the heap, elements are represented as boxes and paths as cylinders. Position and colour interpolators are used for element and colour animations which are

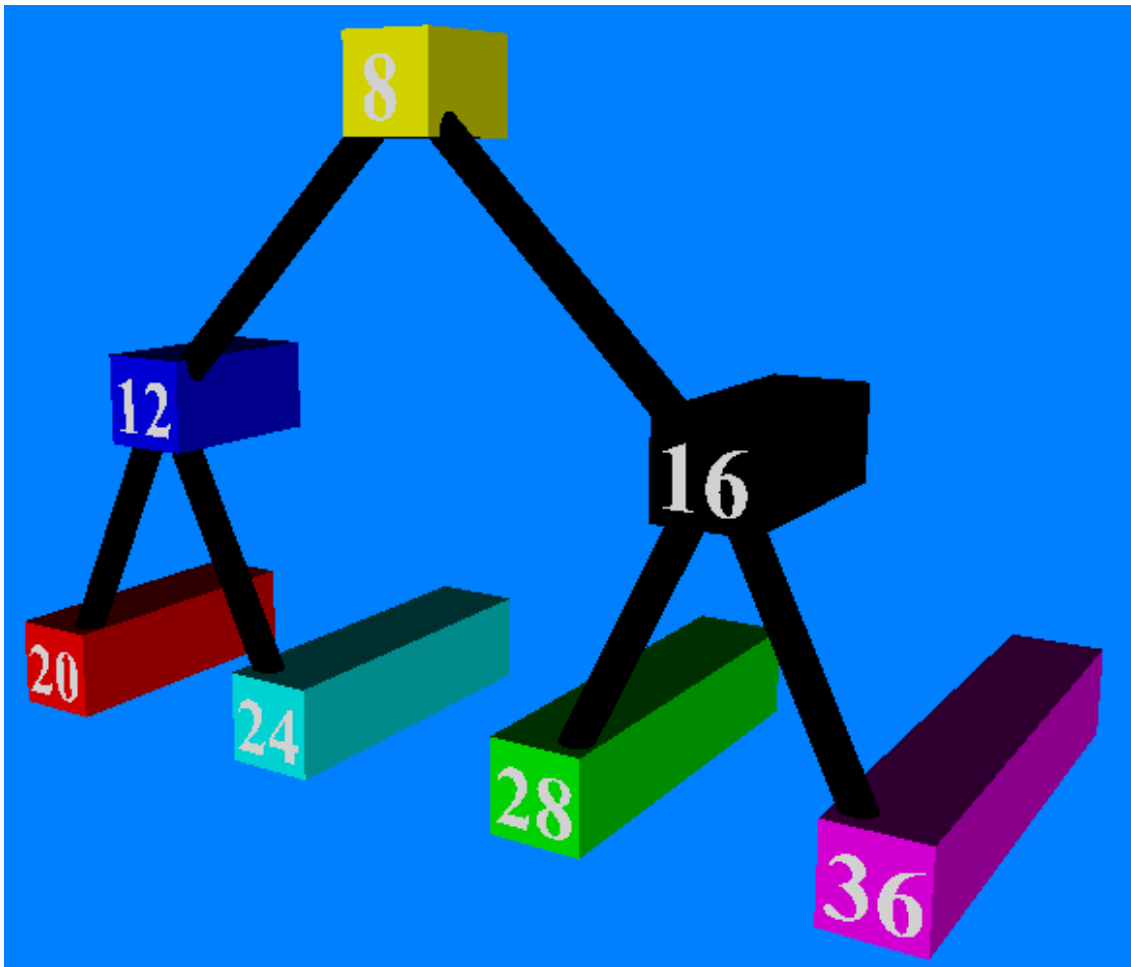


Figure 4.10: X3D Heapsort Algorithm Animation — completed algorithm animation with united sticks like array and heap views.



controlled by the timer (same kind of timer as in the shortest path animation §4.2.1) that cycles every 20 seconds. According to Brown and Najork [33, 168, 169], using colour in this animation is not crucial because the value of a stick is encoded by its length, but it is useful to distinguish the elements during the animation. We can use colour to support this statement by colouring the elements using the red, green, blue spectrum to signify that they have been ordered. Elements toward the red spectrum could represent larger elements, while elements toward the blue spectrum could represent smaller elements. Once an element arrives at its final destination its colour is flashed white to signify that it has been ordered. Adding start, stop, step, and speed buttons is also possible to give a user greater control over the animation.

We extended this example by including number text labels to the elements as it helps a user to understand the animation when viewing it from in front like in Figure 4.10. When viewing the animation from in front and without text labels it is hard to determine the value of each element as the length is obscured in the negative Z axis. The text labels could be embedded inside the boxes but would not be visible. So a separate text object was created which is offset from the boxes but nested inside a transform node (§3.3.2). The transformation grouping means the text will move when the box also moves. Alternatively, we could have used an image for the text labels and applied the images as texture to the nodes. We could then have also applied the text label images to all faces of the nodes so that if the viewpoint was rotated then the text label would always be visible. Using text labels as texture would have required creating separate text label images for each element in the animation and we deemed that as unnecessary because if we allowed a user to input their own data set then this would have been an extra process which we were not interested in evaluating. Since our heapsort animation was small this extra process would not have scaled very well if our sample data set was much larger.

Figure 4.11 shows the animation during different stages of the sorting algorithm. In this heapsort animation example there are 32 element transition moves. Figure 4.11(a) shows the start of the animation with the initial data set. The next three figures show elements that are in motion in the algorithm animation. Figure 4.11(b) shows elements 28 and 8 swapping positions, Figure 4.11(c) shows elements 24 and 8 being swapped and 28 reaching its final destination, and Figure 4.11(d) shows 12 being swapped with 8 as the last step in the algorithm.

Figures 4.11(e) and 4.11(f) show alternative views of the animation where the viewpoint is rotated by a user to show the boxes displayed horizontally, vertically, and the text labels not visible. Figure 4.11(e) shows the animation in progress while Figure 4.11(f) shows the animation once completed. These views can expose a

different way of viewing the animation as no text labels are represented so the user can only distinguish the elements by their size and position. We believe having number text labels for each element helps a user to understand the algorithm when viewing the animation from any view point, but only user studies will be able to prove this.

### 4.2.3 Elementary Sorting

Visualising a program's entire execution history can help in understanding the behaviour of a program, whereas most algorithm animation implementations only show the current state of an algorithm. Displaying the history of a small algorithm animation shows how the state of the algorithm has changed over time. When visualising larger algorithms the history can reveal features that are not commonly known.

Ronald Baecker's seminal 1981 film *Sorting Out Sorting* [17] introduced the sticks view for sorting algorithms. The sticks view shows the array of elements as a row of sticks, where the height of each stick is proportional to the element in the array. When the algorithm is completed the sticks are arranged in ascending order from left to right, but the sticks view does not provide any history of the execution of the algorithm. Brown and Najork [33, 168, 169] created the *chips view* which captures the history of execution of algorithms and is drawn in the X-Y plane at increasing values of the Z axis.

We extended this example by showing multiple algorithms all executing at the same time and implemented in X3D. Brown and Najork [33, 168, 169] in their examples only show one algorithm animating at once. We thought that it would be useful to show different algorithms animating concurrently as in *Sorting Out Sorting* but in X3D. Showing multiple algorithm animations concurrently will potentially help a user understand how the different algorithms work. With this extension we mainly wanted to determine how well X3D supports displaying more than one algorithm animating at once.

Figure 4.12 shows bubble, selection, and insertion sort algorithms all animating at once. The combined views allow a user to see both the current state of the array and the history of an algorithm's execution. Whenever an algorithm sorts an element the previous state of the array is drawn in the X-Y plane at increasing values of the Z axis in the chips view. The elements in the chips view are encoded by colour and when the algorithm is completed the elements in the sticks view will be ordered from blue to red. Once an element is sorted in the sticks view the element is flashed a white colour to signify that the element has been ordered in the data set.

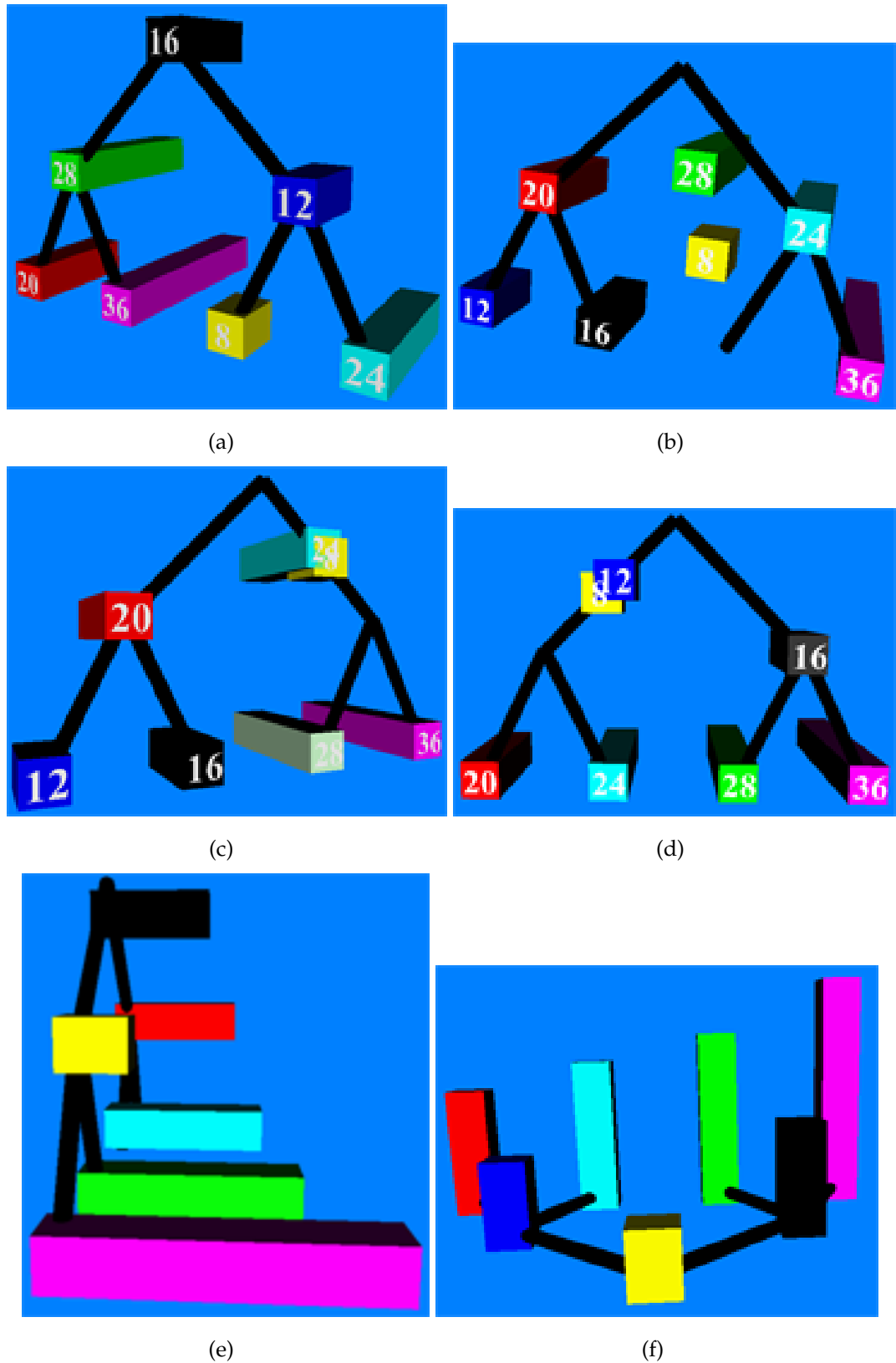


Figure 4.11: X3D Heapsort Algorithm Animation — different stages of the algorithm. The anomalous node placements in parts 4.11(b), 4.11(c), and 4.11(d), are the results of the elements being in motion.

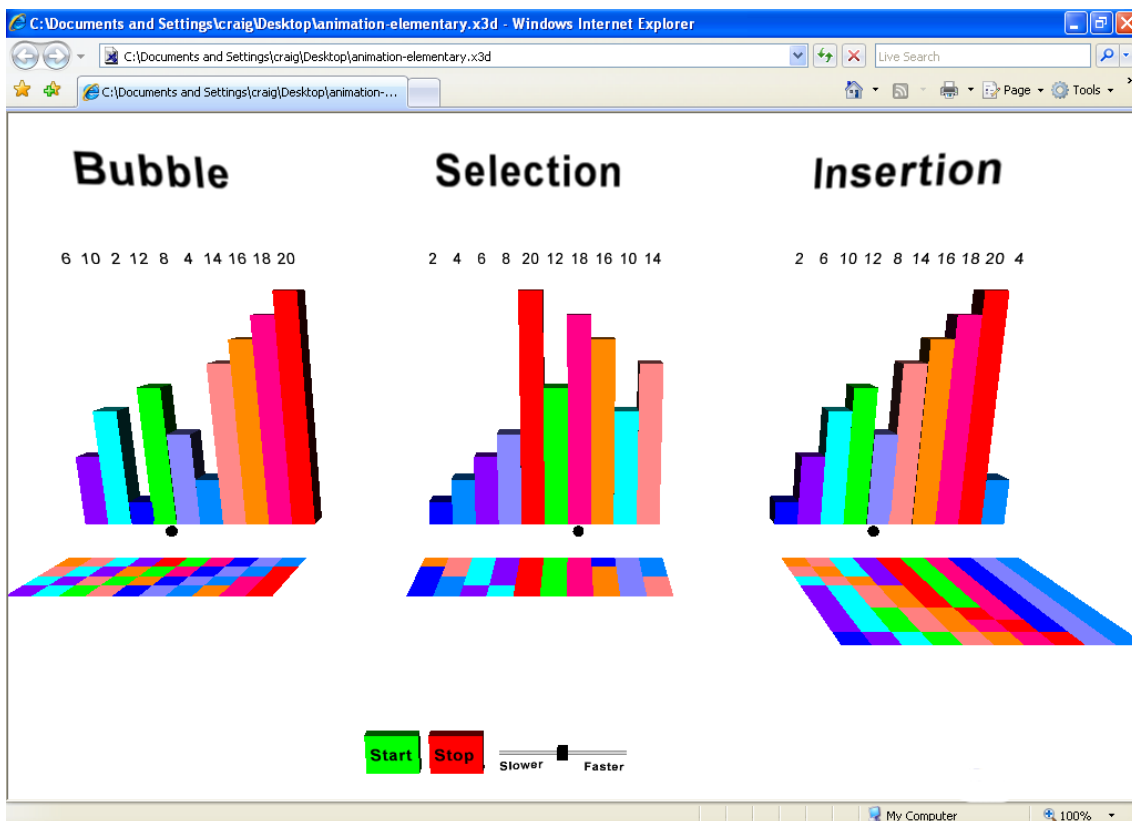


Figure 4.12: X3D Elementary Sorting Algorithm Animation — chips view.

Boxes were used to implement the sticks view. We found it best to implement the chips view as one node, an indexed face set. Our indexed face sets have separate colours for each position in the array. In our example from Figure 4.12 10 faces make up one line of the chips view, one face for each element in the array. If we had of used boxes instead of an indexed face set in the exact same position (X-Y-Z coordinates) then the boxes would obscure the sticks view and the algorithm animation steps when viewing from in front.

We also did another extension to Brown and Najork's [33, 168, 169] example by experimenting with the chips view. Figure 4.13 shows the *blocks view* instead of the chips view, and positioned below where the the chips view would normally be. Figure 4.14 shows the blocks view again, but positioned behind the animating sticks view. The chips view is also visible in Figure 4.14. We found that the block views in the different positions (below and behind) obscured parts of the execution history as some elements were not visible. Depending on the viewpoint some of the smaller elements were hidden by some of the larger elements which made it harder for a user to grasp the execution history. Our implementation of the blocks view required creating a box for each element in the previous execution state, which significantly increased the amount of X3D code compared with just using the indexed face set for the chips view. The increase in code was because an individual box was required instead of one attribute from an indexed face set.

In the animation if a user rotates the viewpoint the text labels above the sticks view would be hard to read, so a billboard node was used (§3.3.3). The billboard node is a grouping node which modifies its coordinate system so that the billboard node's local Z axis turns to point at the viewer. Unfortunately, the billboard node does not allow the text to be legible if the animation is turned upside down.

Rather than the traditional visualisation of iterating through the sticks view by flashing each element as in *Sorting Out Sorting*, we felt it was best to use a sphere object for the iteration steps as all the elements in the sticks view are different colours. We used the iterating sphere across all the algorithms to be consistent and to show how the actual algorithms work in terms of the inner details. We tried positioning the sphere in different places. We first positioned the sphere on top of each element but felt that it distracted the user when the animation was executing and it also meant we had to change the Y coordinate value each time the sphere moved. We found it best and easiest to position the sphere below the sticks view where it could be viewed from any viewpoint, which meant we only had to change the X coordinate value.

In the elementary sorting algorithm animation we used the inline node which can embed an X3D scene stored at a location on the web or local file system into the current scene. For each individual sorting algorithm in the greater elementary

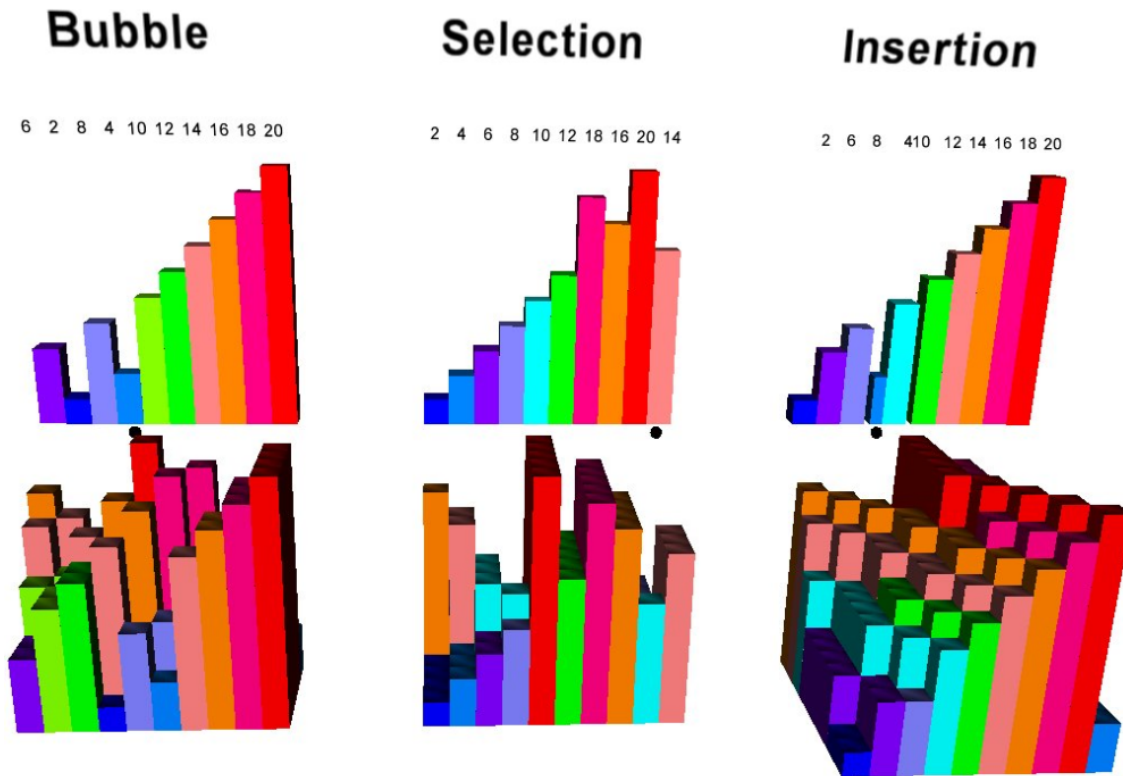


Figure 4.13: X3D Elementary Sorting Algorithm Animation — blocks view.

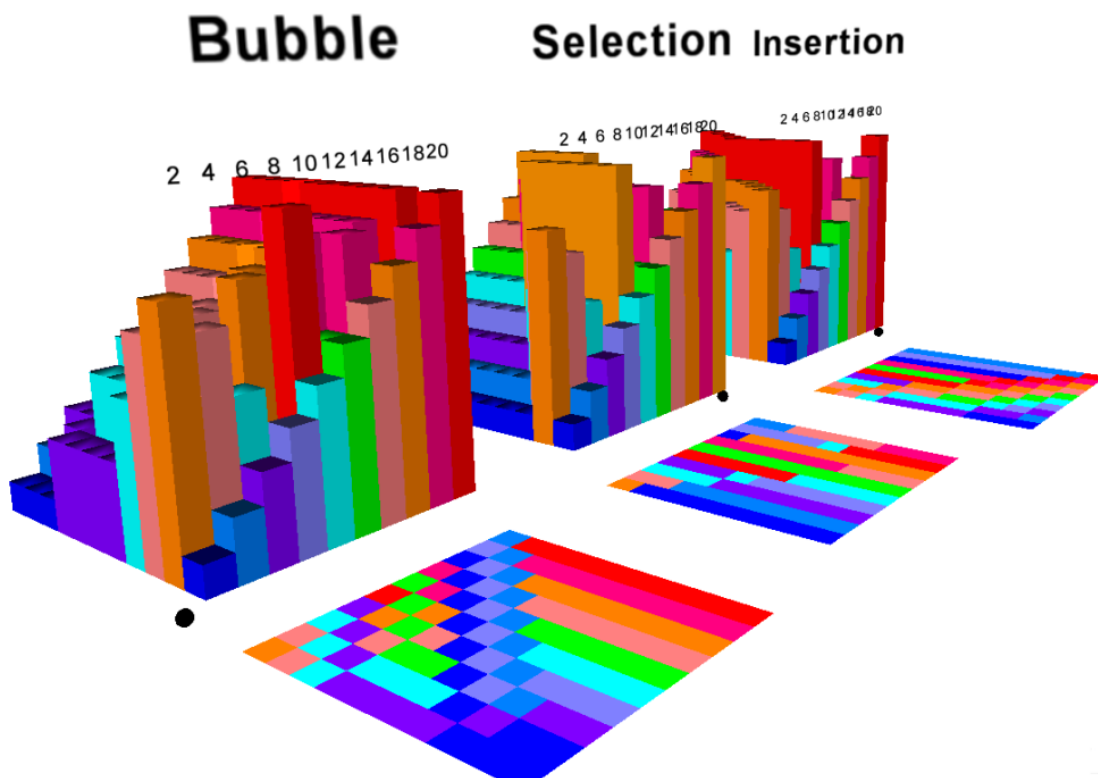


Figure 4.14: X3D Elementary Sorting Algorithm Animation — blocks behind view.

sorting animation we had to physically position the elements in a different place in the global 3D world so that they were not mapped onto each other. This is because we encoded the data into the presentation layer and each individual geometry node, position interpolator node, and ROUTE directive required a separate declaration. The only node we could generically use across all of the animations was the timer which was declared in the root file. We had to use the export semantics so that the timer could then be imported and used in the separate algorithm animation files.

#### 4.2.4 Discussion

**Routing Event Model:** One reason we were interested in using X3D for algorithm animation was the motion capabilities provided by the routing event model (§3.3.5). The X3D routing event model allows a user to click or drag a node such as start, stop, or change speed buttons (see bottom of Figure 4.12) and has event handlers that sense these touches or drags. The touch or drag sensors initiate a time sensor which then activates an interpolator that has frames where upon a variable in the target node is changed. The target node in our examples are 3D geometry (boxes, spheres), material (colour), and coordinate points (inside coordinate nodes).

The following code in Figure 4.15 is a sample from the insertion sort algorithm animation that demonstrates the routing event model. The code shows a time sensor (timer), interpolators (position and colour), and target nodes (material and transform), however, there are not touch sensors used in this example. Reading through the code a time sensor is defined to loop and last for 20 seconds (line 2). A position and colour interpolator are defined which states at what time in the animation an element moves or changes colour (lines 4–7). The box geometry (lines 9–14) and the appearance of the box are nested within a transform node (lines 8 and 15). Finally, the ROUTE directives (lines 16–23) are the glue which link the timer with the position and colour interpolators (lines 16–17) and then the transform (lines 18–19) and material (lines 20–23) nodes.

An issue with the X3D routing event model is that for each 3D geometry node that needs to be animated, a separate position interpolator and two ROUTE directives are required. This is the same for the material colour nodes. From our perspective the routing event model seems to be quite a cumbersome design. If the routing event model could be achieved by generic JavaScript functions, ROUTE directives, and interpolator nodes it would make it much easier to create complex animations. For our algorithm animations which use small data sets there was a lot of X3D code that could have been reduced if the design of the

```

1 <X3D>
2   <Scene>
3     <TimeSensor DEF="TIME" cycleInterval="20" loop="true"/>
4     <PositionInterpolator DEF="MOVE16" key="0.1 0.2
5       0.3 0.4" keyValue="2 8 0, 4 8 0, 4 8 0, 6 8 0"/>
6     <ColorInterpolator DEF="FLASH16" key="0.4 0.5 0.6"
7       keyValue="1 0.5 0, 1 1 0, 1 0.5 0"/>
8     <Transform DEF="ITEM16" translation="2 8 0">
9       <Shape>
10        <Box size="2 16 2"/>
11        <Appearance>
12          <Material DEF="COLOR16" diffuseColor="1 0.5 0"/>
13        </Appearance>
14      </Shape>
15    </Transform>
16    <ROUTE fromNode="TIME" fromField="fraction_changed"
17      toNode="MOVE16" toField="set_fraction"/>
18    <ROUTE fromNode="MOVE16" fromField="value_changed"
19      toNode="ITEM16" toField="set_translation"/>
20    <ROUTE fromNode="TIME" fromField="fraction_changed"
21      toNode="FLASH16" toField="set_fraction"/>
22    <ROUTE fromNode="FLASH16" fromField="value_changed"
23      toNode="COLOR16" toField="set_diffuseColor"/>
24  </Scene>
25 </X3D>

```

Figure 4.15: X3D routing event model from the insertion sort algorithm animation.



routing event model used a generic approach. Our most sophisticated example, the elementary sorting algorithm animation (§4.2.3), used 186 ROUTE directives and 93 interpolators which required a large overhead to manage. For example in Figure 4.15 it would be better to have one generic JavaScript function for each of the position and colour interpolators, and the ROUTE directives. If we were to reimplement these animations we would choose to encode the data separate from the presentation layer and use prototypes (§3.3.6).

**User Control:** A common issue across all the 3D algorithm animation examples was that X3D did not allow continuous smooth animations [236]. Instead, we had to create pauses for each element that moved otherwise elements that had a long distance to travel in the scene would be moving while other elements are animating hence causing confusion for the user. In the code from Figure 4.15, the pause can be seen in the keyValue of the position interpolator, where element 16 moves from position 2 8 0 (X,Y, and Z coordinate values) to 4 8 0 and then stays at 4 8 0 for another second before moving onto 6 8 0.

We created a dashboard which had start, stop, pause, and speed buttons for a user to control the animation (Figure 4.12). The dashboard was implemented using a proximity sensor node which generates events when the viewer enters, exits, and moves within a region in space (defined by a box). We found the dashboard to be very useful as it enables a user to rotate the viewpoint and always have the dashboard in front of them to control the animation.

The actual user control buttons in the dashboard were implemented using touch sensors, plane sensors, a time sensor, and JavaScript. Start and stop buttons were straightforward to implement and required a timer, touch sensor, position interpolator, and a ROUTE directive. For changing the speed and pause buttons, JavaScript functions were required. The speed control either increased or decreased the time sensor cycle interval depending on which way a user drags the box attached to the cylinder. The pause button required a function call and an additional ROUTE directive for each element in the animation.

**Auralisation:** “Software visualisation” through sound is known as program auralisation which is the process of forming mental images of the structure, and behaviour of software by listening to the execution of a program [70]. Brown and Hershberger [36] suggest using audio in algorithm animation to reinforce what is being displayed visually. In the elementary sorting algorithm animation we have added sound for when an element has been ordered in the data set. We implemented program auralisation by playing a sound when the element also flashes a white colour to signify that it has been ordered. Each element has

a different tone played whose pitch is linearly related to the element's value. Likewise we could do the same for the other animations and also play a sound for each comparison or movement of an element.

**Animation Quality:** We applied some of the 10 commandments of algorithm animation to our animations as suggested by Peter Gloor [91]. We tried to be consistent in all of our animations as we made each of them last the same amount of time per cycle, which was 20 seconds, and provided a dashboard for user controls. We allowed interactivity, where upon users can rotate the viewpoint of the animation, and stop, start, pause, and change the speed of an animation. We kept the user interested in our animations as we showed small data sets that made the animations complete quite quickly and provided some realistic examples that could be user in introductory computer science data structure courses.

We emphasised the visual component as much as possible and we feel our sorting animations are very self explanatory. The elements in the sorting animations use depth or height of columns to represent the value of an element. Colour is employed to show a red green blue spectrum once elements are ordered. Sliders are used to control the speed of an animation and a sphere is used to simulate iterating through a data set. We also provided the execution history in our elementary sorting animation which shows why the current action happened based on a snapshot of past actions.

We did not manage to apply all of the 10 commandments for various reasons. We could have been more clearer and concise if we had implemented step buttons throughout the animations. We believe we can implement this with a JavaScript function as shown by some of our other button implementations. We did not allow user input in our animations, but we would like to do so in the future. This would require accessing the run-time of the scene and modifying the input variables and could be implemented through the X3D Scene Access Interface (SAI). Therefore, we did not allow for the system to be generous and forgiving to erroneous user manipulations.

We adapted the animations to the knowledge level of a user by allowing them to be able to set the speed of animations, but we did not provide any step button or built-in online help. We did not incorporate both symbolic and iconic representations (e.g. pseudo code or source code). For this case study showing both representations was not the focus. Section 4.4 shows how we could integrate online help and symbolic and iconic representations.

Finally, for our animations we did not include any algorithm analysis but this would be a useful feature for a user. We could have listed what the worst case scenario is for each animation in text next to the title of the animation, however,

we were thinking more along the lines of how many comparisons and swaps were conducted during the animation. Since our algorithm animations encode the data in the visualisation we could either show this information statically or dynamically using a program counter that changes during the animation.

## 4.3 UML Diagrams

In this section we examine the text, layout, and extensibility features of X3D. A common approach for modeling software is to use the Unified Modeling Language (UML) [248] to show visual diagrams of how software is designed and operates. The nature of UML diagrams is for a 2D layout and most diagrams display a large amount of text. Laying out UML diagrams and displaying text in 3D is difficult. We have implemented class and package diagrams in X3D using XSLT to transform our RCD files. Our sequence diagrams in X3D are created from XTE execution traces but are manually hand crafted. We now elaborate on these diagrams.

### 4.3.1 Class Diagrams

A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them. The two main kinds of static relationships are associations and subtypes. Class diagrams also show the attributes and operations of a class and the constraints that apply to the way the objects are connected [83]. We have implemented some UML class diagrams in X3D using a similar style to McIntosh et al. [163] (§2.2.4 and Figure 2.11). Their layout of each class is manually calculated and they only show 700 classes, whereas our layout of classes is automatically calculated and generated from an execution trace.

Figure 4.16 shows a class diagram of the small CityScape Java program generated from an execution trace. The CityScape Java program creates a SimCity style simulation and is used in our undergraduate computer science courses. The diagram shows 100 classes. Each package is displayed at different depths along the Z axis while the classes in each package are displayed on the same vertical X-Y plane. Each class has the name of the class, its attributes, and operations. No relationships are represented. Figure 4.17 shows a much larger class diagram of the Eclipse integrated developers environment (IDE) application, also generated from an execution trace. The Eclipse class diagram shows 4536 classes.

The properties for each class in the class diagrams are declared as a prototype (ProtoDeclare) (§3.3.6). The prototype uses the associated UML fields for the classname, attributes, and operations. Then each class in the packages is described

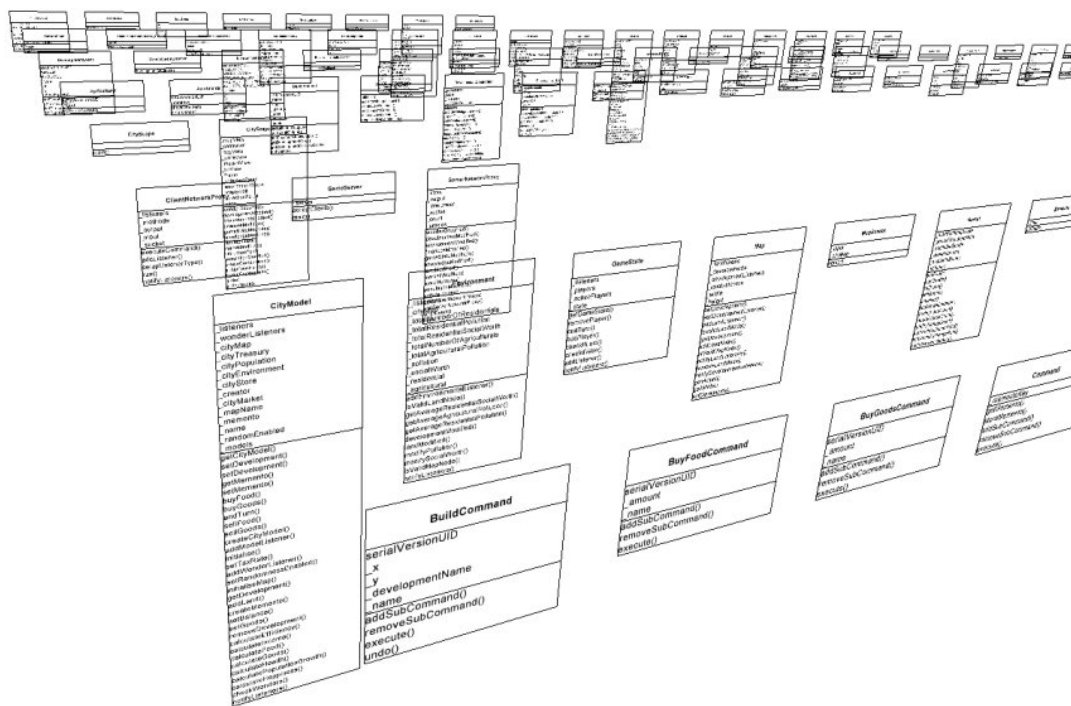


Figure 4.16: X3D Cityscape UML class diagram — 100 classes

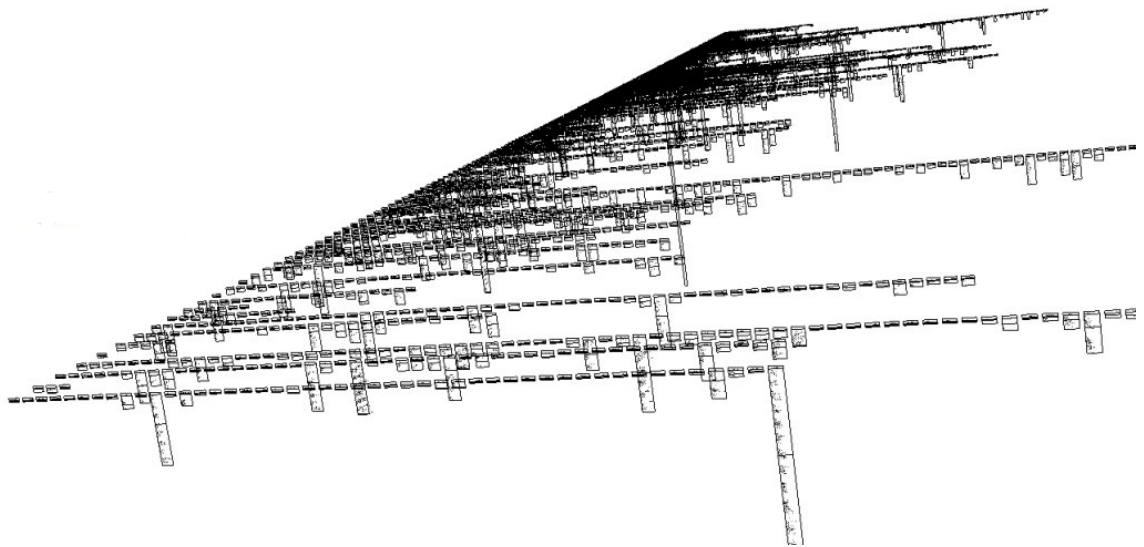


Figure 4.17: X3D Eclipse UML class diagram — 4536 classes, showing the flexible scale of the visualisation technique.

in a prototype instance (ProtoInstance), which also has the same fields as that of the prototype class.

Figure 4.18 is a snippet of X3D code from the CityScape UML class diagram displayed in Figure 4.16. In Figure 4.18 the class diagram prototype is declared (lines 1–12), the controller package (lines 13–28), and the prototype instance shows the BuildCommand class (lines 15–23), along with the class' attributes (serialVersionUID, \_x, \_y, \_developmentName, \_name) (lines 17–18) and operations (addSubCommand(), removeSubCommand(), execute(), undo()) (lines 19–20).

```

1 <ProtoDeclare name="ClassDiagram" url="x3d-classdiagram.x3d">
2   <field accessType="initializeOnly" name="classname"
3     type="MFString"/>
4   <field accessType="initializeOnly" name="attributes"
5     type="MFString"/>
6   <field accessType="initializeOnly" name="operations"
7     type="MFString"/>
8   <field accessType="initializeOnly" name="height"
9     type="SFFloat"/>
10  <field accessType="initializeOnly" name="width"
11    type="SFFloat"/>
12 </ProtoDeclare>
13 <Group DEF="nz.ac.vuw.mcs.comp301.controller">
14   <Transform translation="20 0 -120">
15     <ProtoInstance name="ClassDiagram">
16       <fieldValue name="classname" value="BuildCommand"/>
17       <fieldValue name="attributes" value="serialVersionUID _x _y
18         _developmentName _name"/>
19       <fieldValue name="operations" value="addSubCommand()
20         removeSubCommand() execute() undo()"/>
21       <fieldValue name="height" value="15"/>
22       <fieldValue name="width" value="15"/>
23     </ProtoInstance>
24   </Transform>
25
26 ...
27
28 </Group>

```

Figure 4.18: UML Class diagram prototype declaration and an example prototype instance from the CityScape Java program.

Line 1 of Figure 4.18 shows a link to the layout X3D file (`url="x3d-classdiagram.x3d"`). The class diagram layout file is used to do the layout for each of the prototype instances. The prototype instances do not produce any new XML elements instead they are mapped to existing X3D geometry. Inside the layout file a JavaScript function draws a border around the class, its attributes, and operations. The border around a class is implemented with an indexed face set, which calculates the height of the border, but has a fixed width. The layout file also defines the global position of the class in the world.

### 4.3.2 Package Diagrams

A package diagram shows packages, classes, and the dependencies among them. In theory a package diagram is just a class diagram that shows only packages and dependencies [83]. We have implemented some package diagrams of some Java programs in X3D.

Figure 4.19 shows a package diagram of the same CityScape Java program as in the previous subsection. The diagram shows eight packages at increasing Z and negative Y axes values. Classes are listed inside of each package. We experimented with changing the colours of the text in the package since black is very plain and wanted to see if another colour was more effective. The blue used for the class listings seems to stand out better than the black from the class diagrams.

The package diagrams were implemented in the same way as the previous class diagrams with prototypes. The only differences is that the boxes did not need to be as big and that the prototype fields are different. Since we were displaying less information the boxes did not need to cater for attributes and operations, instead we only had a package name and the names of classes.

### 4.3.3 Sequence Diagrams

Sequence diagrams are a type of interaction diagram. They describe how groups of objects collaborate in providing some behaviour [83]. We are interested in seeing if we can create a similar diagram to our previous work as listed in Figure 2.18(b). We have implemented a sequence diagram in X3D manually hand crafted rather than using XSLT as in our previous UML diagrams.

Figure 4.20 shows a manual drawing of a sequence diagram of the CityScape Java program of seven objects and the main method from an oblique angle. Each object activation is implemented using cylinders while an object's lifeline is implemented as an indexed line set (§3.3.1). Message calls and message returns are also implemented using indexed line sets.





Using cylinders was not the best choice of object to use, nor was a box when the diagram scales to be much larger. Once there are many more objects and messages then the global position of these shapes causes an issue. The issue is that a cylinder and a box are centred or drawn from the middle point of the shape. This means that the text labels of the objects at the top of the diagram would need to change position to stay in an even row every time a cylinder or box was increased in size. Instead, it would be better to use an indexed face set which can look like a box, but define eight coordinate points. The first four points determine where the object starts at the highest point in the Y axis (e.g. the top face of the box). The bottom face is four coordinate points at the lowest point in the Y axis. When using boxes for object activation the global position of object names, messages, and message returns are not reliant on the positioning of the boxes and can therefore be nicely aligned. For the object lifelines we wanted to draw dashed lines rather than solid lines, however not all of the X3D browsers implemented this feature or line property.

We used a billboard grouping node for the text labels (§3.3.3). The billboard node enables a user to rotate the view point and to always see the text lined up to the associated element it is matched to. This can be useful if a user rotates the view in order to read the diagram from right to left instead of the norm of left to right. Likewise if a user moves down, left, or right using the slide navigation type in the diagram the text will always face the viewer.

#### 4.3.4 Discussion

The prototypes used to create the UML class and package diagrams took some time to program, but are very useful for creating automatic software visualisations of UML diagrams in X3D. Building a library of reusable prototypes such as UML diagrams will enable future developers to use them in order to create their own software visualisations. Creating a software visualisation component in the X3D specification from a library of software visualisation prototypes such as UML diagrams will mean that X3D browser vendors can implement this kind of component therefore making it easier for developers to create software visualisations with X3D.

We could lay out all the packages in the package diagram on a single X-Y plane with the same Z axis value. The package layout for this kind of visualisation would not scale very well once there are many packages. Instead, we intentionally offset the packages in both the Y and Z axes as we would like to be able to combine both the package and class diagrams together. For example when a user is viewing the package diagram they can click on a package or a class and get further details.

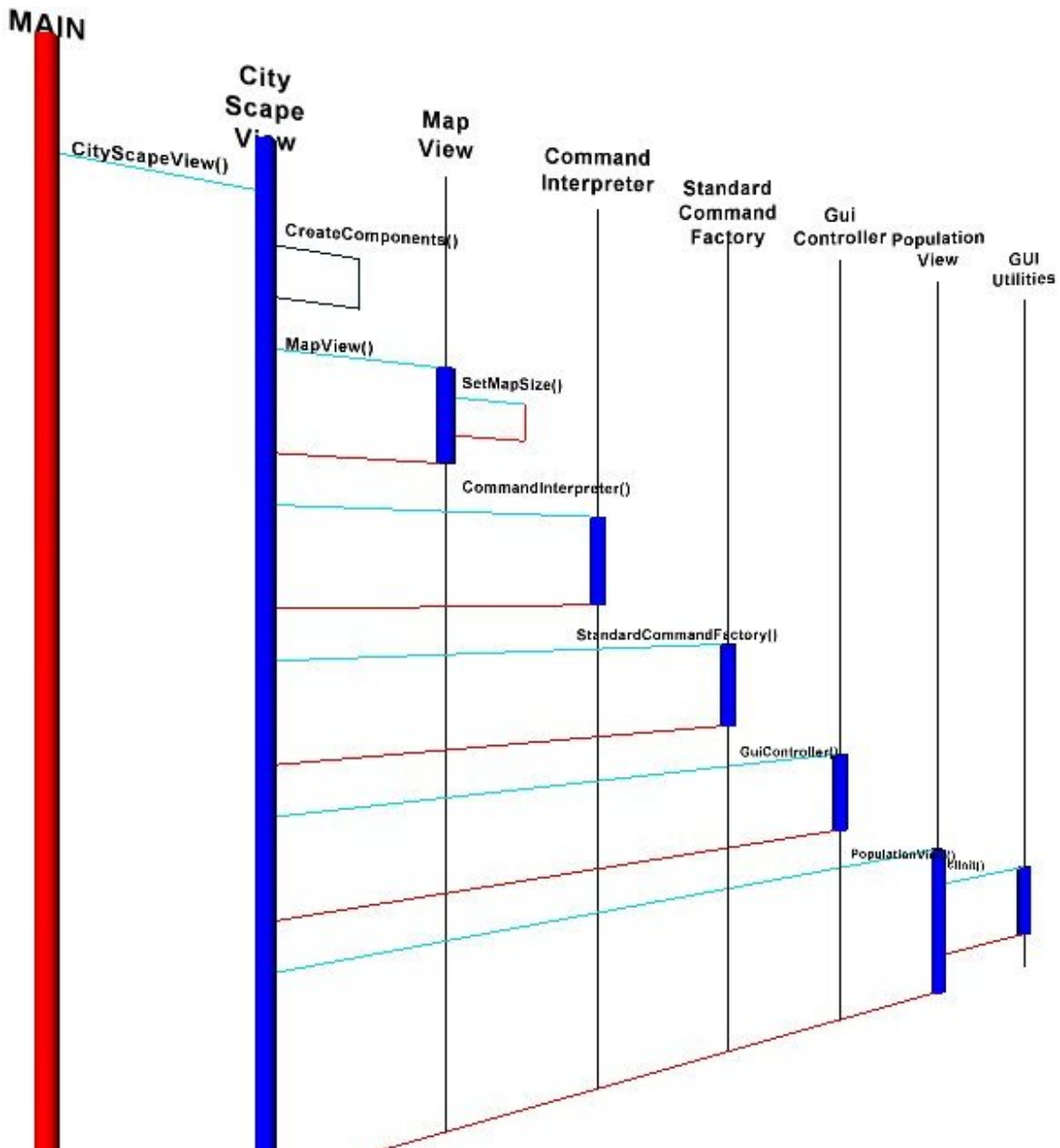


Figure 4.20: X3D Cityscape UML sequence diagram

Perhaps the classes in a package diagram could be expanded or unfolded to show what is visible in a class or all the classes within a package could be laid out on the X-Y plane with the same Z axis value. Likewise a user could then fold or retract the classes and turn them back into a single package entity.

We are currently investigating the automatic creation of sequence diagrams similar to our class diagrams and sequence diagrams from execution traces. Creating these kind of visualisations automatically requires more effort since X3D is purely a graphics file format and it is hard to encode the relationship data between the nodes within a X3D software visualisation. Using prototypes as in our class and package diagrams would be a good approach to solving this problem.

The 3D navigation capabilities provided by the X3D browsers proved to be very successful for navigating amongst 3D UML diagrams. These options included sliding up and down, flying, walking, and examining. The navigation options made it easy for a user to browse the packages, classes, methods, and fields in the Java software programs. Once the X3D content became very large (4000+ classes Figure 4.17) the rendering speed of the X3D scene was so slow that it compromised the performance of these navigation options so that the software visualisation was unusable.

We found the billboard node for changing the rotation of text to be very useful as it can help a user to read text labels from the distance or from an oblique angle. The sequence diagram was a good example use of the billboard node because the text was readable on the far right when exploring the diagram on the left. If the text were positioned far off in the distance then it would be hard to read. We also tried using the billboard node in our class diagrams. We found that it was not as effective, as the classes were positioned at decreasing values in the Z axis but with similar Y axis values so it actually made it hard to read the text.

## 4.4 Documentation-Related Visualisations

In this section we test the integration capabilities of X3D for documentation-related visualisations. When visualising software it can be useful to show the underlying data source of the visualisation or combine another view of the software which may help in understanding the overall piece of software. We have implemented some software visualisations in X3D that combine a visualisation with the original source code, Javadoc<sup>6</sup> API specification, and videos which describe components of the software.

---

<sup>6</sup>Javadoc is a tool for generating API documentation in HTML format from the original source and Java documentation comments in the source code.

### 4.4.1 Source Code Visualisation

We are interested in being able to show a visualisation or diagram with the associated source code of a program. This is useful to a developer as they can see the graphical representation of a program and the code that is associated with the diagram on the same screen. Previous work by colleagues in our research group showed how we did this using SVG (§2.3.4 and Figure 2.18(a)).

Figure 4.21 shows a visualisation which represents a UML like class diagram of a C++ program. The image has two displays, the left frame shows the X3D visualisation and the right frame shows the original source code of the program which was used as the information for the visualisation. The source code is rendered in HTML. In the visualisation classes are represented as red spheres and abstract classes as cones which animate to different colours. The white cylinders represent the inherited relationship amongst the classes.

A user can examine the visualisation by rotating the view which gives different views of the class diagram to gain a greater understanding. A user can manipulate nodes by dragging them around the visualisation to show different parts of the class diagram. A user can click on a class which changes to a white colour and the associated class declaration is highlighted in the source code of the right frame using JavaScript. In the visualisation the user has selected the class at the bottom right of the visualisation (changing to a white sphere) which has highlighted the associated class declaration (highlighted yellow) in the source code (the Fox class).

### 4.4.2 API Javadoc Visualisation

Following on from our source code visualisation we are interested in being able to provide non-source code textual descriptions of a piece of software alongside a visualisation. Figure 4.22 shows the class diagram from Figure 4.16 with the Javadoc for the CityScape Java Program.

When a user clicks on a class in the visualisation in the left frame the associated Javadoc class is displayed in the main class frame window on the right. We could quite easily also create a visualisation for the packages represented in the Javadoc. The advantage of this visualisation is that X3D allows a user to see the software visualisation as well as browse the Javadoc at the same time.

### 4.4.3 Structured Video Visualisation

Ron Baecker [19] suggests creating structured video systems for creating visual demonstrations and explanations of software to show users how to accomplish desired tasks. Users can access the videos by the web and delivered over the

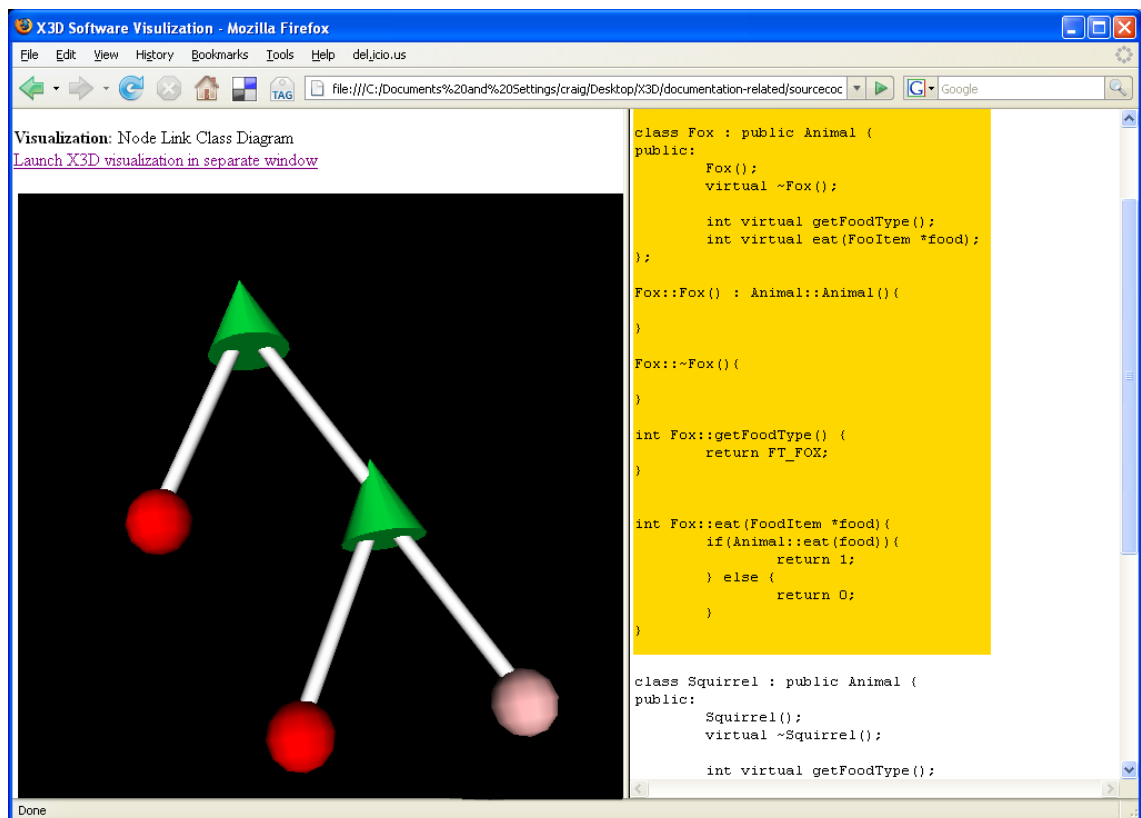


Figure 4.21: Source code visualisation with a UML like class diagram and C++ source code.

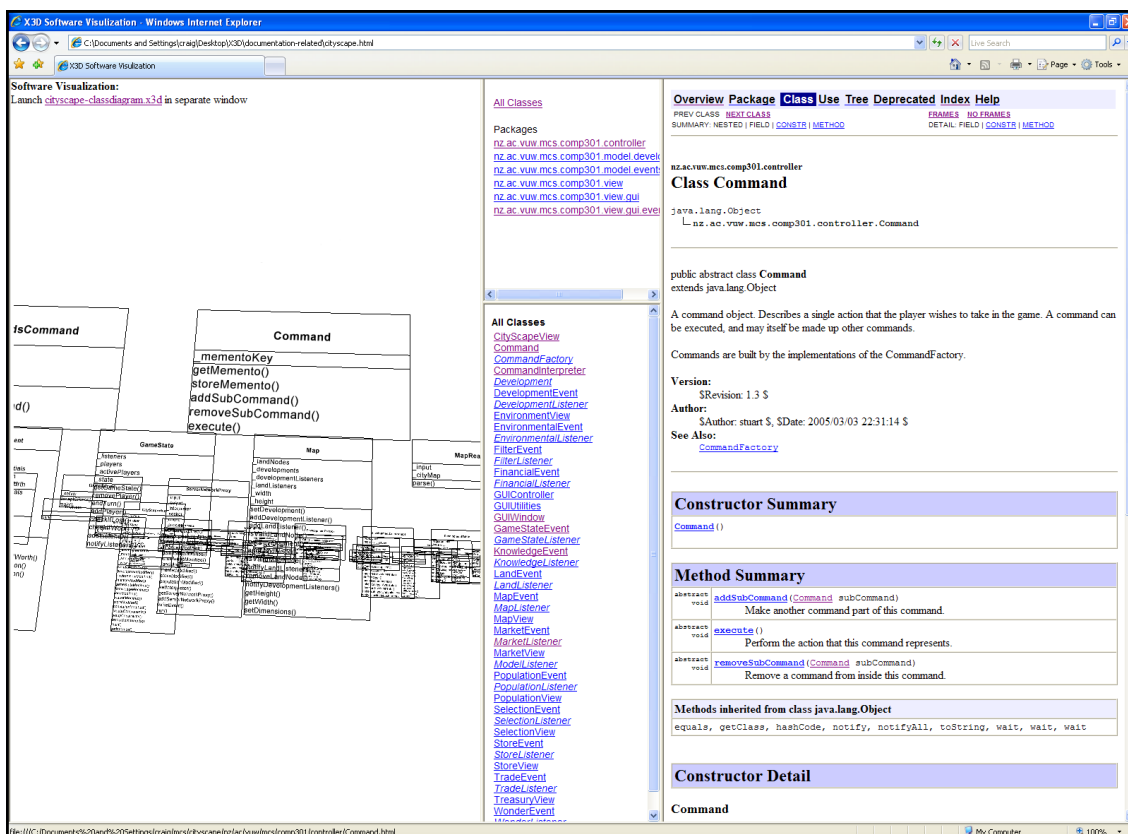


Figure 4.22: API Javadoc Visualisation with the Cityscape UML class diagram.

Internet. None of his examples show actual source code or visualisations of software.

Figure 4.23 shows a video of a developer discussing the design of one of the components from the same software visualisation as in Figure 4.21, once a user has clicked on one of the nodes in the class diagram. The advantage of using a video approach is that a developer will be able to go into more detail as to why a particular component was designed in a certain way or what the software actually does or intends to accomplish. The videos may even open up issues or discuss ideas which are just too hard to describe within written or diagram documentation.

Each of the components within the visualisation have a video associated with them. Once a user clicks on a node it triggers a touch sensor to play a movie in the left part of the visualisation. The video is mapped as texture using the movie texture node to an indexed face set (§3.3.1).

We could also display the video on the specific geometry node of a component within the visualisation that was selected, but it would be harder for a user to interpret the video since the shapes in the visualisation are all different. If we have of used UML diagrams and display the video on the classes or packages then the text would have been hidden. We felt it was better to separate the additional video from the main part of the visualisation, so that a user can navigate around the visualisation and play a video once they come across an interesting component.

Mapping the video as texture to geometry within the visualisation was easier than using another frame as in the previous documentation visualisations (§4.4.1 and 4.4.2). We could have also used the YouTube<sup>7</sup> approach by embedding a video in a web page. The advantage of using the X3D movie texture node is that it only requires a couple of extra lines of X3D code rather than a totally separate HTML file. The actual movie is also embedded in the software visualisation rather than a separate frame.

The video part of the visualisation could also be displayed within a dashboard implemented with a proximity sensor so that no matter what angle the visualisation is displayed at the video will always be directly straight in front of the viewer. Finally, the location, direction, and relative intensity of the sound of the video can also be controlled using the X3D sound node.

#### 4.4.4 Discussion

We found the linking capabilities of X3D to be very useful and easy to create documentation-related visualisations. The linking between frames is controlled by

---

<sup>7</sup><http://youtube.com>

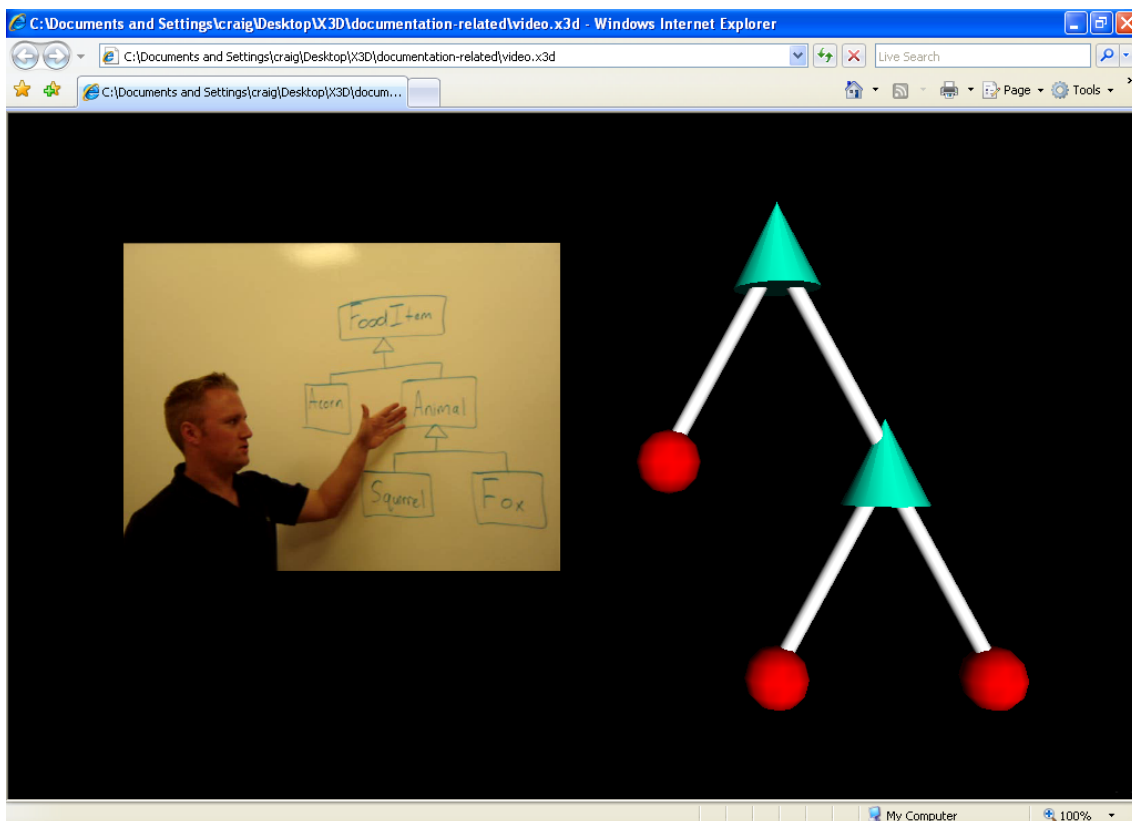


Figure 4.23: Structured Video Visualisation with a UML like class diagram.



the anchor node which is similar to the HTML `<a href='url'>` anchor element. The only difference is that the parameter field needs to be set to inform the browser where to open the linked HTML file (e.g. `parameter="target=frame_name"`). The anchor node also supports both the `http://` and `file://` protocols.

The biggest concern with the X3D integration capabilities is in terms of scaling the documentation for the visualisation. For each piece of geometry in the visualisation there needs to be an associated link to either an anchor embedded in an HTML web page, HTML web page, or a video. Annotating this information manually into a software visualisation can be quite time consuming. There needs to be a way to easily make the connection between the external documentation and the components within a software visualisation, presumably automatically generated by our software visualisation tools.

## 4.5 Execution Trace Visualisations

In this section we analyse the performance display capabilities of X3D for displaying large execution trace visualisations. Our ongoing visualisation project VARE produces XML execution traces (§2.3). We are interested in producing visualisations from the execution traces. We have described earlier in this chapter how we can produce execution trace visualisations over the web (§4.1) and UML diagrams from our execution traces (§4.3). Drawing diagrams other than UML in 3D is hard and it is unclear what technique or metaphor to use in order to create a 3D visualisation.

Wiss et al. [270] found if it were possible to predict what structure the data will have for a visualisation then this will help determine what metaphor or technique is most applicable for a visualisation. We now show some of our visualisations based on 3D shapes and 3D information visualisation metaphors transformed from our execution traces.

### 4.5.1 All Elements From an Execution Trace

We want to create the same type of visualisation from different execution traces using the same template. Figure 4.24 demonstrates a simple X3D software visualisation using XSLT to transform all elements from an execution trace into a sequence of black spheres. The example used is the CityScape Java program as described earlier in the chapter.

The XSLT template code parses an execution trace to find an event, draw a sphere, increment the X axis variable, and then continue to find the next event and

so on. In Figure 4.24 a file called axes.x3d is included to display the different axes where the Y axis is green, the X axis red, and the Z axis blue.

Figure 4.25 extends the previous figure by giving each node a different shape and colour depending upon the event in the execution trace and displayed on separate lines. Blue spheres are object creations, method calls are green boxes and the main class is a method call represented as a red box. White cones are method returns and also used to represent the end of an object.

Figure 4.26 shows the same information but displayed in a single line which shows the exact sequence of events that occurred in the execution of the CityScape program. The CityScape program is not multi-threaded.

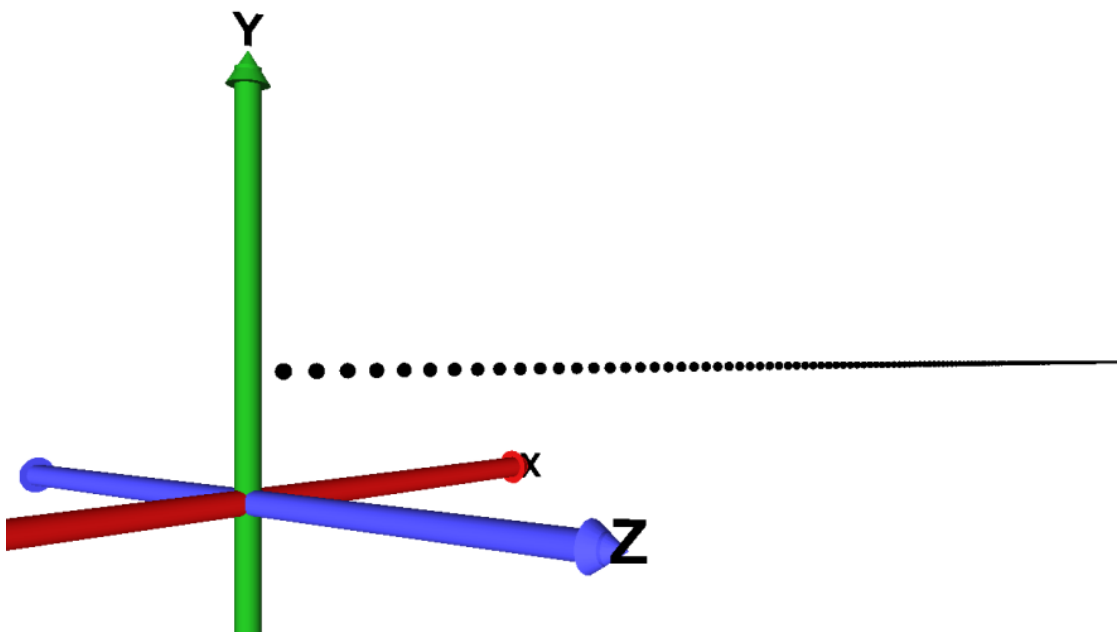


Figure 4.24: All elements from an execution trace represented as a sequence of spheres.

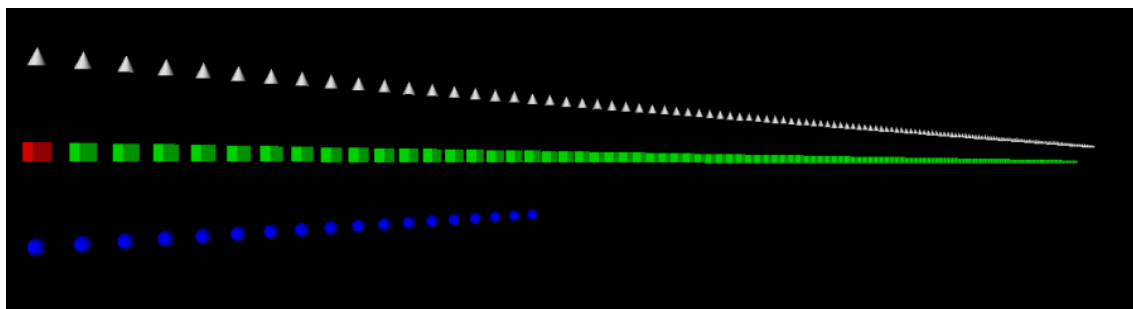


Figure 4.25: All elements from an execution trace represented as different shapes on different lines.

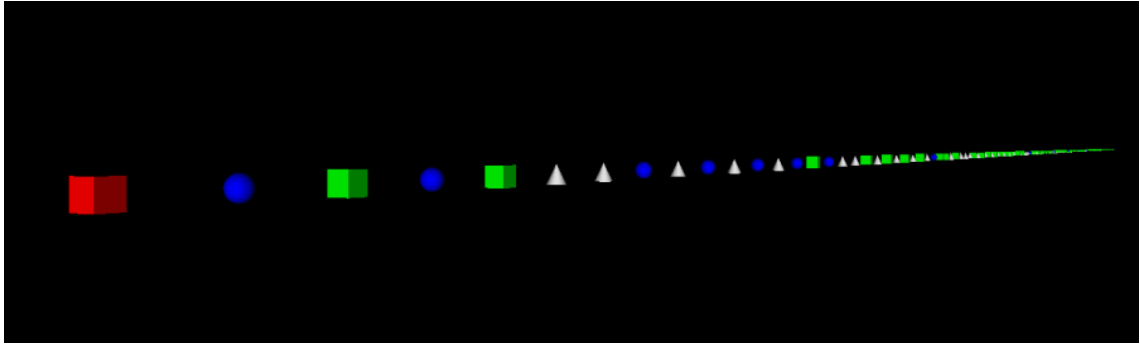


Figure 4.26: All elements from an execution trace represented as a different shapes and on the same line.

A sequence of shapes does not give that much insight into the execution of software and it is hard to display all the information on the screen at one time. Exploring more powerful visualisation techniques for laying out the information about an execution trace may provide more insight about a program. We now explore 3D compound shapes and 3D information visualisation metaphors.

### 4.5.2 3D Compound Shapes

There are many kinds of 3D shapes that exist which we could implement to represent our execution traces. We have decided to implement a representative sample of shapes including a cuboid and a 3D spiral cylinder. These kind of visualisations could make it easier for the user to analyse and navigate repeated patterns of execution of a program as opposed to a long sequence of shapes in a line. We now elaborate on both of these 3D compound shapes showing visualisations using the CityScape and Eclipse Java programs.

Figure 4.27 shows 275 events from the CityScape program laid out as a cuboid. We use the same shapes and colours for representing the different events in an execution trace as the visualisations from the previous subsection. The visualisation starts at the main method represented as the red box at the bottom in the middle of the figure. The next event is displayed along the X axis. After the first 7 elements the position of the next element is displayed with a decremented Z axis value and so on for the first 49 elements. Each layer of the cuboid has 49 events. The next layer then has the Y axis value increased and so on until all the events of the execution trace are displayed. The eventual shape may not end up being a cube due to the number of events in the execution trace.

Each event in the execution trace is parsed and then a JavaScript layout function is called to position the node in the world. Figure 4.28 shows the XSLT template

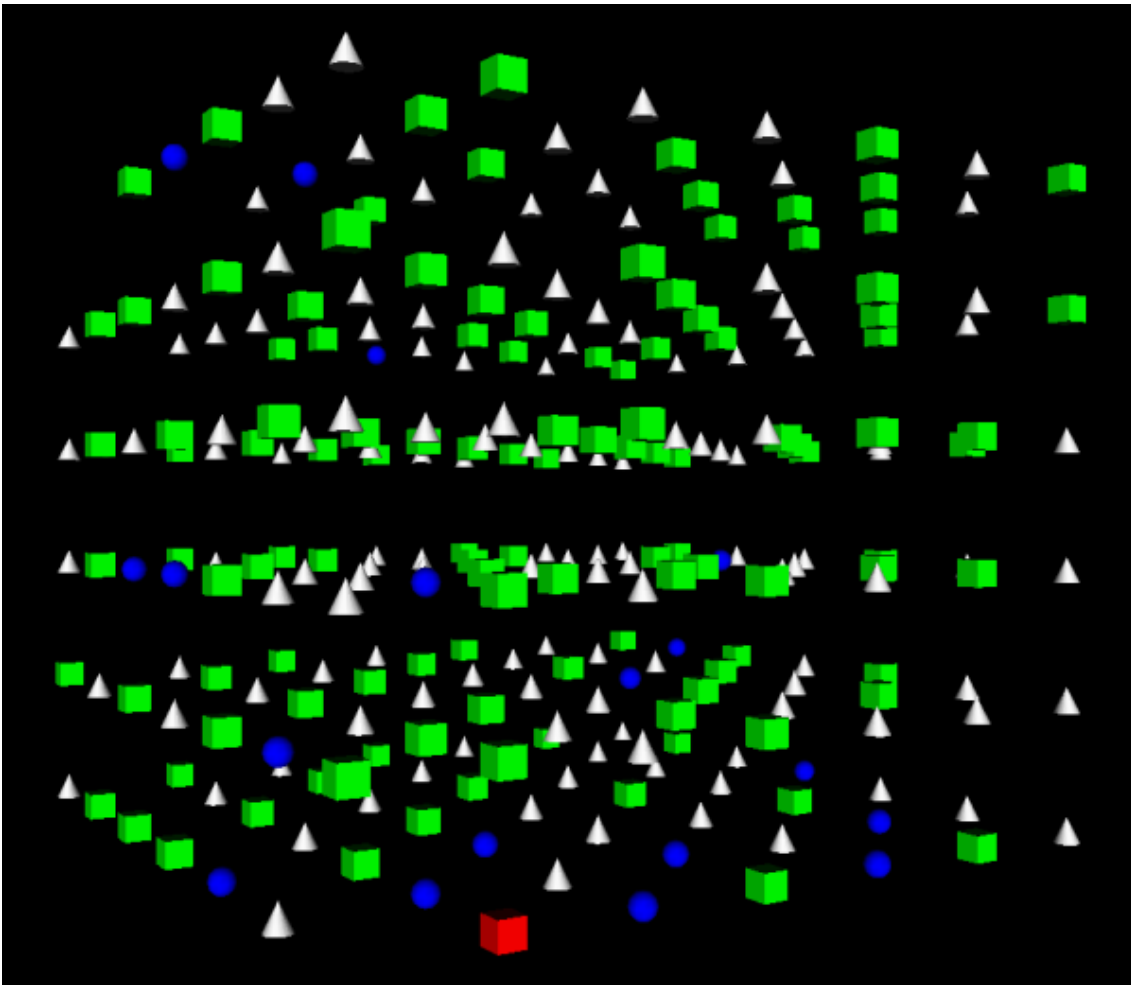


Figure 4.27: All elements from the CityScape execution trace represented as a cube like shape.

code for parsing object creation events. The code searches for an event that is of type `xte:objectcreation` (line 2), then gets the associated `eventid` (line 3) and passes it to the layout function (line 6).

Figure 4.29 shows the JavaScript layout function, `getTranslation()` (lines 4–15) which takes an `eventid` as a parameter and returns X, Y, and Z translation coordinates for a transform node. Once the layout function returns with the translation values a blue sphere is drawn (lines 7–17 of Figure 4.28). This is the same for other events such as method calls (green box) and method returns (white cone), but different shapes and colours are used. The `length` variable (line 6 of Figure 4.29) determines what size the cube is going to be.

```

1 <!-- OBJECT CREATION -->
2 <xsl:template match="xte:objectcreation">
3   <xsl:variable name="eventid" select="@xte:eventid"/>
4   <xsl:text disable-output-escaping="yes"><![CDATA[
5   <Transform translation="]]></xsl:text>
6   <xsl:value-of select="result:getTranslation(string(eventid))"/>
7   <xsl:text disable-output-escaping="yes"><![CDATA[">
8   <Shape>
9     <Appearance>
10      <Material diffuseColor="0 0 1"/>
11    </Appearance>
12    <Sphere radius="1"/>
13  </Shape>
14 </Transform>
15 ]]>
16 </xsl:text>
17 </xsl:template>

```

Figure 4.28: XSLT code to parse an object creation event from our execution traces

Figure 4.30 shows the end results after applying the cube XSLT transformation code and the JavaScript layout function (Figure 4.29) to the CityScape execution trace. The CityScape cube visualisation code shows an object creation (blue sphere lines 1–8), method call (green box lines 9–16), and method return (white cone lines 17–24) events.

The visualisations in Figures 4.31, and 4.32 show the Eclipse execution trace after 10,000, and 100,000 events. Despite the large scale of these examples the XSLT transformation step took less than two minutes and the rendering process 10 seconds for 10,000 events and up to 10 minutes for 100,000 events. The rendering and run-time performance of the 100,000 events visualisation was severely comprised

```
1 <xalan:component prefix="result" functions="getTranslation">
2   <xalan:script lang="javascript">
3
4     function getTranslation (id){
5
6       var length = 10;
7       var plane = length * length;
8
9       var x = ((id%length) * 10);
10      var y = (((id - (id%plane)) / plane) * 10);
11      var z = (((id%plane) - ((id%plane)%length)) / length) * -10;
12
13      var result = x + " " + y + " " + z;
14      return result;
15    }
16
17 </xalan:script>
18 </xalan:component>
```

Figure 4.29: JavaScript layout function that calculates x, y, and z coordinates for a node.

```
1 <Transform translation="12 0 0">
2 <Shape>
3   <Appearance>
4     <Material diffuseColor="0 0 1"/>
5   </Appearance>
6   <Sphere radius="1"/>
7 </Shape>
8 </Transform>
9 <Transform translation="15 0 0">
10 <Shape>
11   <Appearance>
12     <Material diffuseColor="0 1 0"/>
13   </Appearance>
14   <Box size="2 2 2"/>
15 </Shape>
16 </Transform>
17 <Transform translation="18 0 0">
18 <Shape>
19   <Appearance>
20     <Material diffuseColor="1 1 1"/>
21   </Appearance>
22   <Cone height="2"/>
23 </Shape>
24 </Transform>
```

Figure 4.30: Elements from the CityScape cube visualisation.

and the navigation options were unusable.

The aim of these visualisations was to test the performance capabilities of the X3D browser implementations rather than produce a stunning visualisation. A different stylesheet is used to create the Eclipse cube visualisations where only the length field is modified in the layout function and the number of events in the execution trace. Ideally we would like an end user to be able to have control over both input variables, the size of the cube and the number of events. A user can also turn collision on or off to fly through the visualisation or to hit nodes.

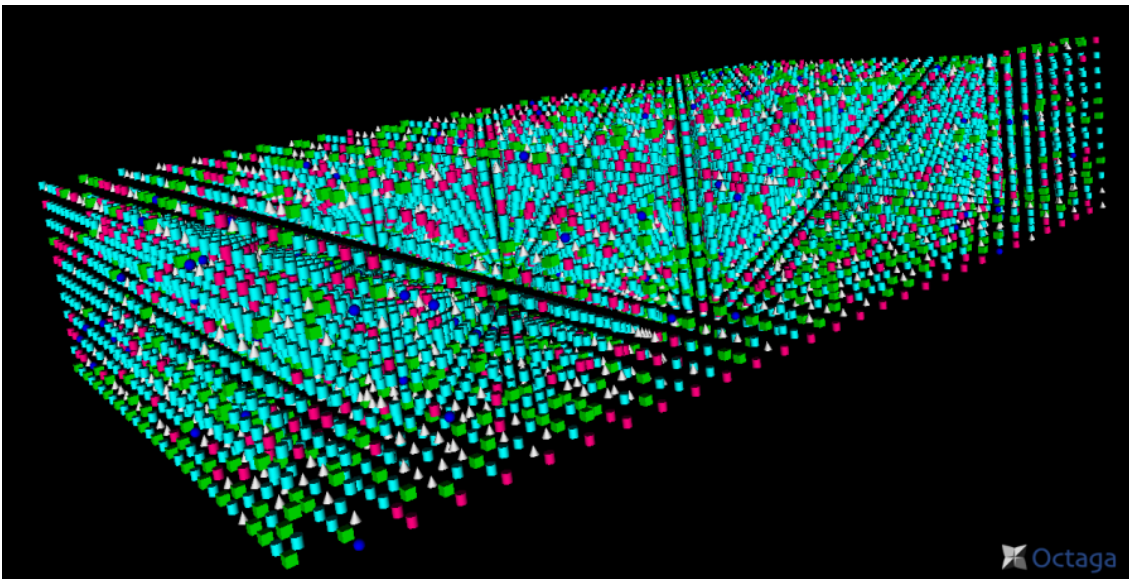


Figure 4.31: 10,000 elements from the Eclipse execution trace.

Figure 4.33 shows the CityScape events laid out as a 3D spiral cylinder. Figure 4.34 shows 10,000 events from the Eclipse program also laid out as a 3D spiral cylinder. The purpose of these software visualisations is to show how we can use the same information but represented in a different manner, by only changing our layout function. These software visualisations also use the `getTranslation()` function (Figure 4.29) for the layout of the events from the execution trace, but the implementation is slightly different. The spiral starts at the bottom and goes in the positive  $y$  direction. The figures show the visualisations rotated on their side with the  $Y$  axis being in the distance. Each element has a different  $Y$  coordinate value and increasing in value.

### 4.5.3 3D Metaphors

We want to be able to create more complex visualisations other than a sequence of shapes or 3D compound shapes. Our XML execution traces are essentially



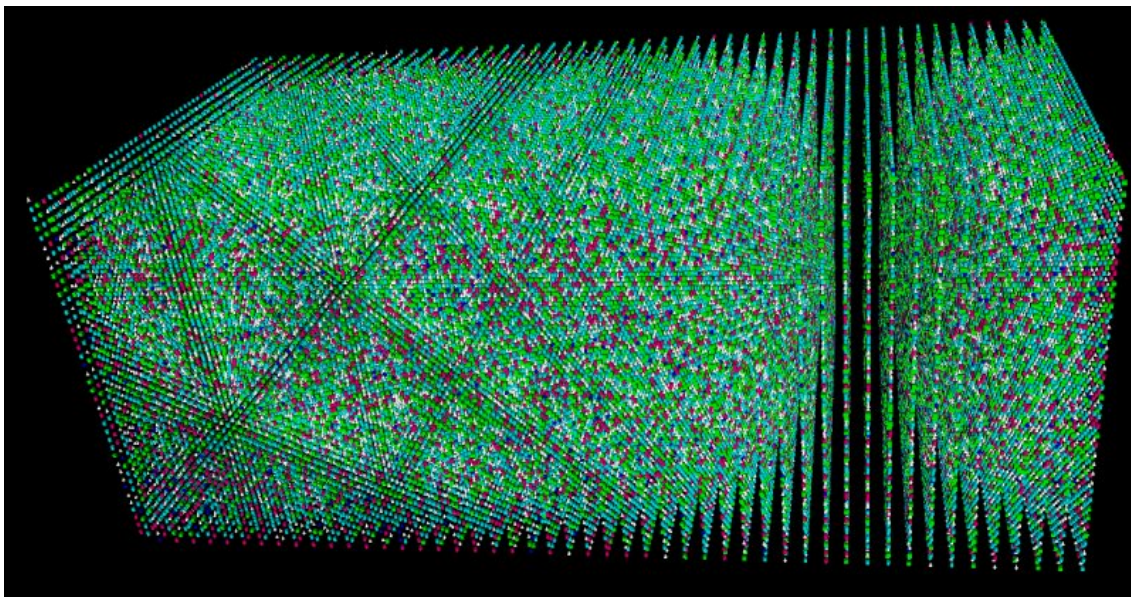


Figure 4.32: 100,000 elements from the Eclipse execution trace.

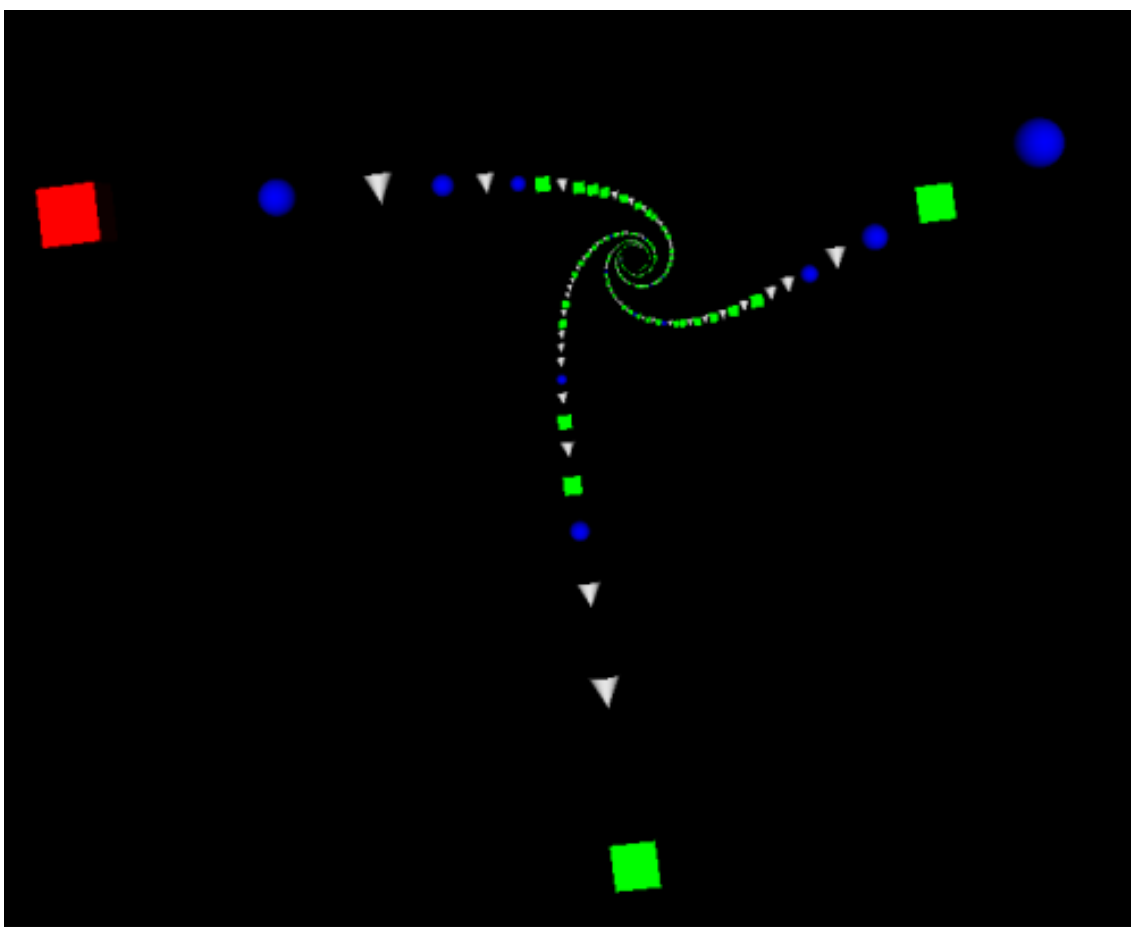


Figure 4.33: CityScape 3D Spiral.

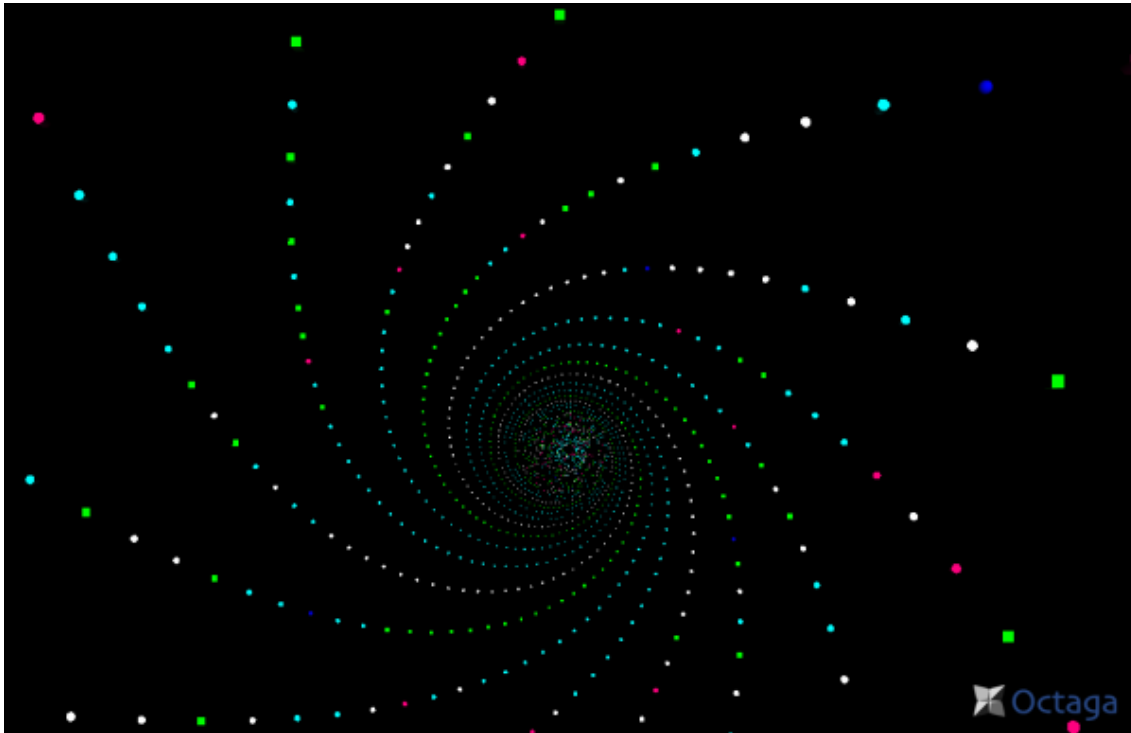


Figure 4.34: Eclipse 3D Spiral.

tree structures. We would like to explore visualising the execution traces as tree structures by replicating some 3D information visualisation metaphors (§2.1.2). Wiss et al. [271] performed an empirical study on three 3D information visualisation designs. The results indicated that subjects were significantly faster with the information landscape followed by the cam tree and then the information cube. We have implemented an information landscape and an information cube to show the flexibility of our visualisation generation approach.

Figure 4.35 shows the layout of all the events from an execution trace of the CityScape Java program as an information landscape [7, 247]. The information landscape is essentially 2 1/2 D rather than 3D. The same shapes are used to represent the same kind of events as in previous sub-sections. The image starts at the red box (the highest node in the image) and is animated from right to left.

Figure 4.36 shows the same information as Figure 4.35, but displayed as an information cube [213]. The information starts with the main method outer red box, followed by the first object creation blue sphere at the top right of the cube then continuing along the links to each of the object creation blue spheres. The spheres are transparent and the events that an object executes or sub-events are enclosed within the sphere.

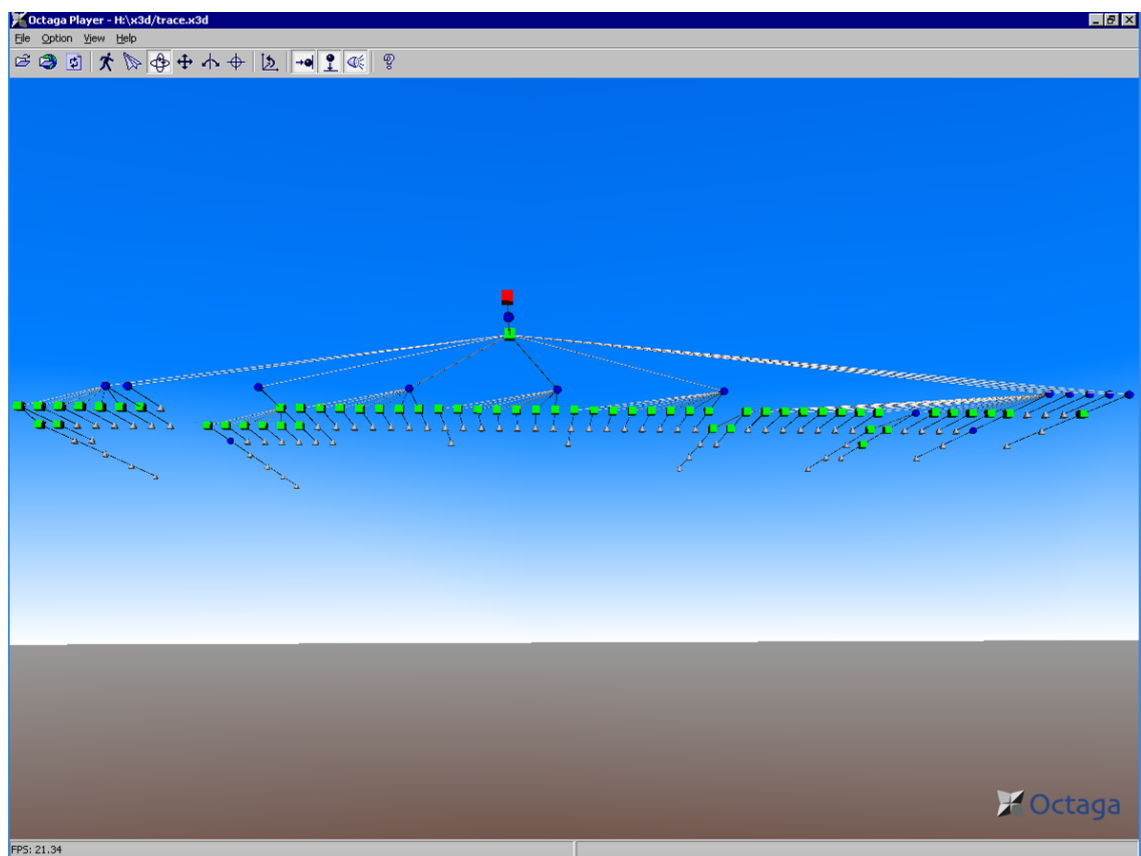


Figure 4.35: All elements from an execution trace represented as an information landscape.

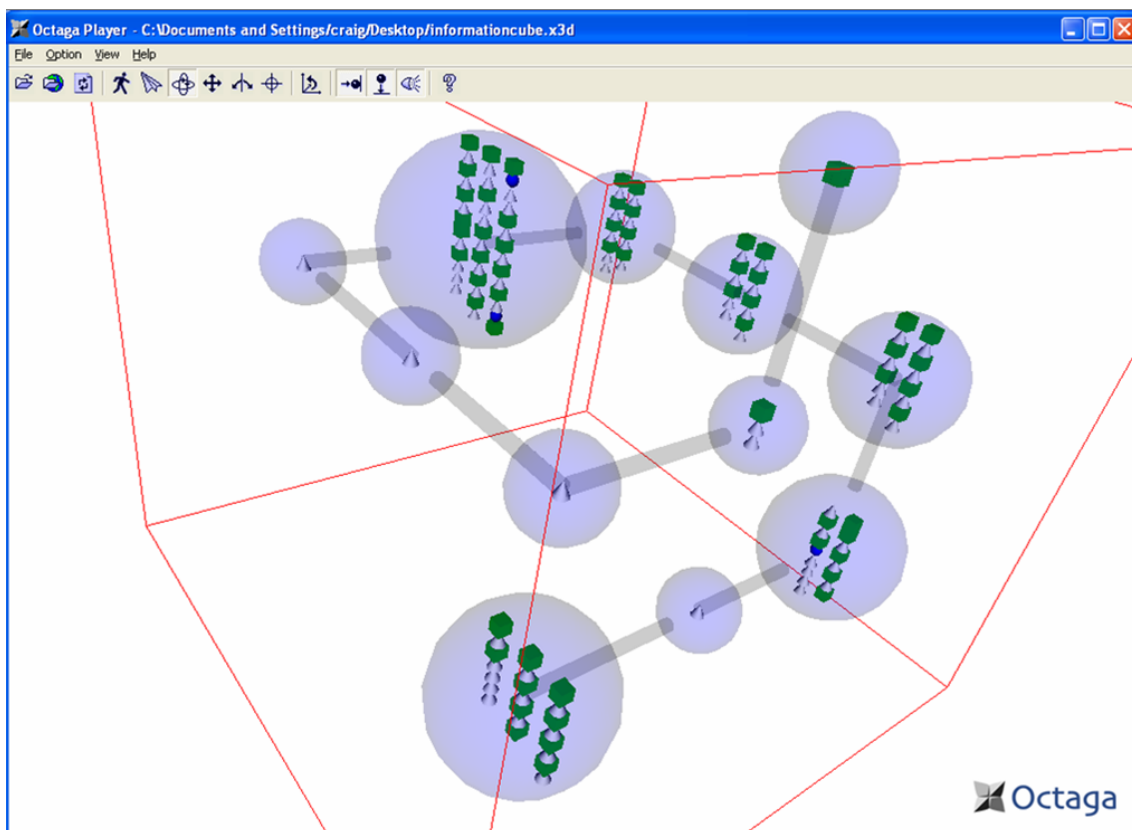


Figure 4.36: All elements from an execution trace represented as an information cube.

#### 4.5.4 Discussion

The main difference between the information visualisation metaphors and the 3D compound shapes is that they show the relationships between the events better since there are linked edges between the nodes in the visualisations. Both types of visualisations encode the information in the visualisation but could be easily modified to use prototypes to separate the information similar to our UML diagrams (§4.3).

Implementing the information landscape was more difficult than the information cube as it required more complex calculations for the links between the nodes and the number of lines of code was greater. A linear layout approach was used. Using graph drawing algorithms [250] as used in graph drawing software systems (§2.3.5) should improve the creation of these 3D information visualisation metaphors.

Our information landscape does not actually reflect a true information landscape. We could achieve a true reflection by changing the dimensions of each objectcreation and method call event to a size that is proportional to the number of sub-events. Hence some events would be proportionally larger than other events, which would make them more prominent in the visualisation. We limited the transparency level in the information cube to only one level to make the visualisation more readable. X3D allows many levels of transparency so long as the nested levels are at a lower level of transparency otherwise the inner objects will not be visible.

A useful feature to add to the execution trace visualisations would be animation and filtering capabilities. Animating the events from an execution trace as a 3D compound shape or an information visualisation metaphor could be useful to see the way the events happen during execution. If this task were implemented it would not scale well for the larger visualisations since the routing event model has a very cumbersome design; see the discussion earlier (§4.2.4).

Filtering could be applied to the events in a visualisation such as remove all method returns or just show the objects that get created, by setting or changing the transparency level to one for a node. For the 3D information visualisation metaphors this would also have to apply to the relationship edges as well. Adding filters would again be quite cumbersome since the routing event model would have to be used, once a user selects or clicks a button. There maybe a smarter way to achieve filtering if the data was encoded in a prototype and there was a JavaScript function that was generic enough to simulate the routing event model.

The size of the Eclipse visualisations after 10,000 events was approximately 2MB, 50,000 events 10MB, and 100,000 events 18MB. The CityScape visualisations

were approximately 50KB. It takes less than 10 seconds to render about 10,000 events, three minutes for 50,000 events, but up to 10 minutes to render 100,000 events. We used the Xj3D converter – a Java application for Windows – to compress the large Eclipse visualisations into the X3D binary format (.x3db). The result was the size of the X3D files were reduced by about 75%. The visualisations once converted, however, were only viewable in the Xj3D browser (§3.4.1) since it is the only X3D browser that currently supports the X3D binary format.

## 4.6 Summary

In this chapter we have described our tool for automatically creating software visualisations from execution traces over the web. We then described our case studies which replicated various well known software visualisation techniques in X3D. The case studies evaluated X3D's animation and interactivity aspects, examined the text, layout, and extensibility features, tested the integration capabilities of X3D, and analysed the performance display capabilities.

The case studies included algorithm animations (shortest path, heapsort, and elementary sorting), UML diagrams (package, class, and sequence diagrams), documentation-related visualisations (source code, API, and structured video visualisations), and execution trace visualisations (3D compound shapes and 3D information visualisation metaphors). For each case study we described the visualisation, how we implemented the visualisation in X3D, discussed the issues that arose during the implementation, and how we could improve each of the visualisations.

The next chapter describes a framework for evaluating X3D as a software visualisation medium that makes use of some well known information and software visualisation taxonomies and frameworks. The following chapter then applies the framework to our experience of creating X3D software visualisations as we have described in this chapter.

# Chapter 5

## Software Visualisation Media Evaluation Framework

### Contents

---

<b>5.1</b>	<b>Differences With Earlier Evaluation Model . . . . .</b>	<b>118</b>
<b>5.2</b>	<b>Scope* . . . . .</b>	<b>121</b>
5.2.1	Requirements* . . . . .	121
5.2.2	Design* . . . . .	122
5.2.3	Method* . . . . .	123
5.2.4	Performance . . . . .	123
<b>5.3</b>	<b>Form* . . . . .</b>	<b>123</b>
5.3.1	Graphical Capability . . . . .	123
5.3.2	Presentation* . . . . .	125
5.3.3	Visualisation Techniques* . . . . .	126
<b>5.4</b>	<b>Interaction* . . . . .</b>	<b>127</b>
5.4.1	User Controls* . . . . .	127
5.4.2	User Navigation* . . . . .	127
5.4.3	User Tasks* . . . . .	128
<b>5.5</b>	<b>Discussion . . . . .</b>	<b>128</b>

---

As described earlier (§2) there have been many software visualisation tools and systems that have been developed over many years, but there has been little effort to evaluate the effectiveness and utility of these tools and systems. There is also no silver bullet for evaluating software visualisations [104] nor any benchmarks [152] to determine how effective a software visualisation is.

There are various information visualisation and software visualisation design guidelines [170, 228], taxonomies [177, 194, 195, 196, 221], models [74, 75], and frameworks [153, 243]. None of this previous research focuses on the actual graphics technology per se. The primary goal of our work is to evaluate how good X3D is for use in software visualisation.

In this chapter we present a framework for evaluating software visualisation media. The framework significantly extends some previous work by fellow colleagues in our research group that created a model for the evaluation of software visualisation media [74, 75]. “Software visualisation media” are defined as technologies used in the creation, deployment and display of graphical images to an end-user via a computer display [74, 75]. The definition does not refer to software visualisation media as the tools nor the visualisations, but to the language used to express the visualisations.

The previous model was based on the framework by Card et al. [45] whose work was inspired by the seminal work of Jacques Bertin entitled “Semiology of Graphics” [24]. The previous model evaluated Scalable Vector Graphics (SVG) for use in software visualisation. Having applied the model to SVG, this chapter now extends the previous model to become a framework, incorporates new features, and integrates some other researchers’ work. In particular our extensions address the needs of 3D software visualisation and other ideas that improve generality. The differences between the new framework in this chapter and the previous model [74, 75] are discussed next.

## 5.1 Differences With Earlier Evaluation Model

We have significantly extended the previous evaluation model [74, 75] which is now a subset of the framework we present in this chapter. The first major difference is that we created three top-level categories, **scope** (§5.2), **form** (§5.3), and **interaction** (§5.4), instead of two distinct categories for information and software visualisation. The scope category defines how the visualisation medium has been designed, how a user can create visualisations using the medium, and how well the medium performs. The form category defines what the characteristics of the output of the software visualisation medium is. Finally, the interaction



category defines how a user can interact with the visualisation medium.

In the **scope** category we include a new area based on the requirements (§5.2.1) for software visualisation [153], as this will help evaluate the capabilities of the medium in context. We next included a new area which looks at why a medium was designed, what platforms the medium operates on, current medium implementations, and if the medium has been used or evaluated for software visualisation before (§5.2.2). We then change our previous model's *integration* area to serve in a more general role and call it *method* (§5.2.3). This modified area keeps two ideas from the previous integration area, how to create visualisations using the medium, and how to view and deploy visualisations implemented in the medium. We retained the term *integration* to describe only how well the medium integrates with other technologies. Finally, we extend the performance (§5.2.4) area by including what kind of data sets can be used, what is the type of visualisations that can be implemented, and what are the size of the visualisation files. Table 5.1 summarises the differences for the scope category with the areas of the previous evaluation model.

Scope	New in Framework	From Previous Model
Requirements	Task, Audience, Target, Representation, Medium	
Design	Purpose, Environment, Hardware, Operating System Production Use, Empirical Evaluation	Implementations, Learning
Method	Integration	Creation, Viewing and Deployment
Performance	Data Sets, File Size, Types of Visualisations	Scalability

Table 5.1: Scope - Differences between new framework and previous model.

In the **form** category we also use aspects like our previous model from Bertin [24], Card et al. [45], and Ware [258] for the graphical capability (§5.3.1) of a medium. We first add new elements to the spatial substrate and marks and properties subsections. We extend the previous models higher level capabilities area and now call it presentation (§5.3.2). We add new elements (Animation, other modalities, programming languages, program synchronisation, multiple views) in the presentation area focusing on specific software visualisation information from the work of Price et al. [194, 195, 196] and Roman and Cox [221]. We introduce a new area, visualisation techniques (§5.3.3), which looks at what kind of techniques

can be represented using the visualisation medium. The techniques include compound shapes, information visualisation metaphors [45], software visualisation techniques [237], pictorial representations [177], and model frameworks [118]. The previous model only used pictorial representations. Table 5.2 summarises the differences for the form category with the areas of the previous evaluation model.

Form	New in Framework	From Previous Model
Graphical Capability		
<i>(Spatial Substrate)</i>	Composition, Alignment Overloading	Dimension Support, Folding Recursion, Axis Distortion
<i>(Marks and Properties)</i>	Value, Texture Connection, Enclosure Text, Advanced Graphics	Marks, Size, Colour, Orientation, Transparency
<i>(Temporal Encoding)</i>		Encoding Time, Encoding Identity, Variation of Retinal Encoding
Presentation	Animation Other Modalities Programming Languages Program Synchronisation Multiple Views	Data Display independence Referencable Objects Layout Constraints
Visualisation Techniques	Compound Shapes, IV Metaphors, SV Techniques, Model Frameworks	Pictorial Representations

Table 5.2: Form - Differences between new framework and previous model

In the **interaction** category we extend our previous models interaction category and call it user controls (§5.4.1), and then add a new element, elision control [194] to user controls. We then add a new area in the interaction category, user navigation (§5.4.2), which looks at how a user can navigate within a software visualisation implemented using the medium. The user tasks (§5.4.3) are also a new area in the interaction category based on Shneiderman’s [228] seven tasks (overview, zoom, filter, details-on-demand, relate, history, and extract) from the visual information seeking mantra (§2.1). Table 5.3 summarises the differences for the interaction category with the areas of the previous evaluation model.

In the next chapter we will use the framework presented in this chapter to evaluate X3D for use in software visualisation (§6). We now elaborate on the details of each of the three categories: scope, form, and interaction.

Interaction	New in Framework	From Previous Model
User Controls	Elision Control	Graphic Changeability, Input Events, Computation User Notation, Temporal Control
User Navigation	Navigation Types, Navigation Control, View Refinement, Speed	
User Tasks	Overview, Zoom, Filter, Details-on-demand, Relate History, Extract	

Table 5.3: Interaction - Differences between new framework and previous model

*Items marked with an asterisk in the framework indicate extensions to the previous evaluation model.*

## 5.2 Scope\*

We first need to determine what the requirements are for software visualisations, as this will help determine the suitability of the medium. We then look at how the software visualisation medium is designed, how the visualisations are specified using the medium, and how well the medium performs for displaying visualisations, in terms of scalability and range of visualisations.

### 5.2.1 Requirements\*

Maletic et al. [153] present a framework for the general tasks of understanding and analysis during the development and maintenance of large-scale software systems. They define five dimensions to reflect the why, who, what, where, and how of software visualisation. We see these dimensions as the requirements for software visualisations and feel that these are important questions to ask when conducting an evaluation for a software visualisation medium. Answering these requirements questions will help determine whether or not the medium being evaluated is suitable for the software visualisations a developer wants to create. The dimensions are as follows:

- Task\*: Why is the visualisation needed?
- Audience\*: Who will use the visualisation?

- Target\*: What is the data source to represent?
- Representation\*: How to represent a visualisation?
- Medium\*: Where to represent the visualisation?

### 5.2.2 Design\*

We propose a new area in the framework, based on how the visualisation medium has been designed. How is the software visualisation medium designed in terms of general characteristics. What is the current status of the medium? Have there been any evaluations conducted on the medium for software visualisation or other domains? The last two points (production use and empirical evaluation) are similar to ideas by Stasko et al. [194], but we apply them to the software visualisation medium rather than the overall software visualisation system.

- Purpose\*: What is the purpose of the design of the medium?
- Environment\*: What is the primary target environment for the visualisation medium?
- Implementations: What kind of implementations and how many exist for the medium?
- Operating System\*: What kind of operating systems is the medium designed for?
- Hardware\*: What kind of hardware does the medium operate on?
- Learning: How much time is required to learn the medium in order to become competent in using the medium to create a substantial software visualisation?
- Production Use\*: Has the medium been used in the production for any software visualisations either in industry or in academia? If so over what time period?
- Empirical Evaluation\*: Has the medium been subjected to good experimental software visualisation evaluation?

### 5.2.3 Method\*

What kind of tools are used to create and view a software visualisation developed in the medium and how well does the medium integrate with other languages? The ideas in this section are similar to that of the previous evaluation model [74, 75].

- Creation: What tools or technologies exist to create software visualisations using the medium?
- Viewing and Deployment: What tools or applications are required to view the software visualisations created using the medium?
- Integration\*: Does the medium integrate or have language bindings to other media or languages (e.g. scripting languages)?

### 5.2.4 Performance

To what degrees does the medium scale up to handle large examples? The first two points and the last point are new ideas while the scalability point is a common general issue in software visualisation.

- Data Sets\*: What kind and size of data sets can the medium handle as input?
- Types of Visualisations\*: What are the types of software visualisations the medium can handle? Which type of visualisation is the medium particularly good at visualising?
- Scalability: How well do current implementations of the medium scale with large software visualisations [74, 75]?
- File Size\*: What are the sizes of the files and how do they scale?

## 5.3 Form\*

What are the characteristics of the output of the software visualisation medium? In particular we want to know what are the graphical capabilities of the medium, how can the visualisations be presented using the medium, and what visualisation techniques are supported by the medium.

### 5.3.1 Graphical Capability

We define the graphical capability of a software visualisation medium as the visual encoding, which is how data is mapped to visual structures that can be seen in a visualisation.

## Spatial Substrate

The spatial substrate defines the space where graphical marks are visible entities that are arranged and positioned in space [45]. The space where the graphical marks are located are defined in terms of axes [45]. The following properties listed define several techniques to increase the amount of information that can be encoded in the spatial substrate and were first defined by Bertin [24] and then extended by Card et al. [45].

- Dimension Support: How many dimensions can the medium support?
- Composition\*: Can axes be orthogonally placed to create a metric space?
- Alignment\*: Can an axis be repeated in a different position in space?
- Folding: Is it possible to fold dimensions so that they are a continuation of an axis in an orthogonal dimension?
- Recursion: Can space be repeatedly subdivided?
- Overloading\*: Can the same space be reused for the same data set?
- Axis Distortion: Can an axis be squashed or stretched so that information can fit in a certain space (e.g. perspective wall [149] or fish-eye views [87])?

## Graphical Marks and Properties

Graphical marks are the objects a user sees when viewing a software visualisation. Whereas the graphical properties also called retinal properties by Bertin [24], are properties processed by the retina of the eye which is sensitive to the mark independent of position. We have added two new points, text and advanced graphics.

- Marks: Can the software visualisation medium support the four elementary types of marks: points, lines, areas, and volumes? What kind of shapes can marks be specified as?
- Size: Can marks change size or scale?
- Value\*: Do marks support a gray scale (series of grays ranging from black to white)?
- Texture\*: Can different textures be applied to marks?
- Colour: Can a mark have colour and can it be changed, either value, hue or saturation? How else can colour be used in a visualisation?

- Orientation: Can marks be positioned or rotated in space?
- Transparency: Is it possible to make marks partially or fully transparent?
- Connection\*: Can marks be used to signify topological structures like graphs and trees?
- Enclosure\*: Can marks be used to encode hierarchies?
- Text\*: Can text be supported in the medium? What kind of fonts and properties can be applied to the text?
- Advanced Graphics\*: Does the medium support advanced graphic techniques (e.g. Non-Uniform, Rational B-Spline (NURBS))?

### **Temporal Encoding**

Does the medium support changing the visualisation over time to communicate additional information [45, 74, 75]?

- Encoding Time: Can the medium easily change graphics to show the passage of time?
- Encoding Identity: Can animations be specified on marks to show identity?
- Variation of Retinal Encoding: Can the value of all retinal encodings change over time?

### **5.3.2 Presentation\***

We are interested in how the software visualisations can be presented and what kind of software visualisation specific information can be displayed. The first three points are from the previous evaluation model [74, 75]. The last five points are based on aspects from some software visualisation taxonomies [194], but aimed at if the medium can support these kind of visualisation aspects, rather than can the actual visualisation system do these kind of visualisations.

- Data Display Independence: Is the underlying data structures or information separate from the actual display of the data [74, 75]?
- Referencable Objects: Does the medium support being able to reference objects in a software visualisation [74, 75]?
- Layout Constraints: Does the medium support basic or advanced layout constraints [74, 75]?

- **Animation\***: Does the medium support animations in the software visualisations?
- **Other Modalities\***: Does the medium support other modalities such as the ability to play sound [36] and video to convey information and if so how does the medium accomplish these tasks?
- **Programming Languages\***: What kinds of programming languages and paradigms can the medium support visualisations of?
- **Program Synchronisation\***: Can the medium support synchronised visualisations of multiple programs simultaneously?
- **Multiple Views\***: Can the medium support multiple views of different parts of the software being visualised?

### 5.3.3 Visualisation Techniques\*

We introduce a new area in the framework which looks at what kind of visualisation techniques does the software visualisation medium support. The previous evaluation model [74, 75] was only concerned with pictorial representations [177] which we also include.

- **Compound Shapes\***: Is it possible to create compound shapes using a combination of primitive graphical marks?
- **Information Visualisation Metaphors\***: What kind of information visualisation metaphors [45] does the medium support (e.g. Tree Maps [116], Cone Trees [220], Information Cubes [213], Information Landscapes [7, 247])?
- **Software Visualisation Techniques\***: What kind of software visualisation metaphors or displays does the medium support (e.g. UML diagrams [83], algorithm animations [33])?
- **Pictorial Representations**: Does the medium support the three well-known types of pictorial representations (graph-based displays or node-link diagrams, statistics-based displays, and source-code-related displays) [177]?
- **Model Frameworks\***: Does the medium support the model frameworks for representing object-oriented systems as described by Keown [118]?



## 5.4 Interaction\*

What capabilities of the software visualisation medium are supported for a user to interact with the visualisation and control it? The user controls are based on previous ideas [74, 75], while we add two new areas user navigation and Shneiderman's information visualisation user tasks [228] (§2.1).

### 5.4.1 User Controls\*

What methods does the software visualisation medium allow a user to employ to interact or customise a visualisation? The ideas in this section are based on ideas from the previous evaluation model [74, 75] and Stasko et al. [194].

- **Graphic Changeability:** Can a user change the graphic representation of graphical marks at run-time?
- **Input Events:** What kind of input events can the visualisation medium recognise (mouse or keyboard)?
- **Computation:** How does the visualisation medium respond to input events?
- **User Notation:** Can a user draw or annotate their own notations using the visualisation medium on screen?
- **Elision Control\*:** Can a user control the amount of information in a visualisation by using controls to elide information?
- **Temporal Control:** Does the medium allow a user to control the temporal aspects of a visualisation such as the current position in time of a visualisation by using start, stop, pause, fast-forward, or rewind buttons?

### 5.4.2 User Navigation\*

We have added a new area for user navigation which looks at to what degree does the medium support navigation through a software visualisation and how can a user control the navigation options.

- **Navigation types\*:** What types of navigation does the medium support?
- **Navigation control\*:** How can a user control the navigation in a software visualisation?
- **View refinement\*:** Does the medium support viewpoints and how can a user control them in a software visualisation?

- **Speed\***: Can the user control the speed of navigation in a software visualisation?

### 5.4.3 User Tasks\*

We have added a new area for user tasks which looks at how a software visualisation medium can implement the user tasks from Shneiderman's [228] visual information seeking mantra (§2.1).

- **Overview\***: Does the medium allow a user to gain an overview of the entire software visualisation?
- **Zoom\***: Does the medium support a user to be able to zoom into items of interest?
- **Filter\***: Can the medium support filtering out uninteresting items?
- **Details-on-demand\***: Does the medium allow a user to select an item or a group of items in the software visualisation and get details when needed?
- **Relate\***: Does the medium allow a user to view the relationships among items in a software visualisation?
- **History\***: Can a history of actions to support undo, replay, and progressive refinement be supported by the medium?
- **Extract\***: Does the medium allow a user to extract sub-collections and any query parameters in a software visualisation?

## 5.5 Discussion

In this chapter we proposed a framework for evaluating graphics media to see how suitable they are for software visualisation. The framework follows in tradition of a similar structure to the software visualisation taxonomies by Price et al. [196, 195, 194] and Roman and Cox [221]. There is some overlap between the categories used in our framework and the top-level categories of their frameworks. The differences between the taxonomies and our framework is mainly our focus on the graphics media rather than the software visualisation systems or the software visualisations themselves.

There are a number of differences between this framework and the earlier evaluation model [74, 75] created by some fellow colleagues in our research group.

We have grouped our ideas in the new framework into three main top-level categories rather than just information and software visualisation sections.

These categories are aimed at the scope of the software visualisation media (Scope §5.2), what are the characteristics of the output of the software visualisation media (Form §5.3), and what kind of user tasks the software visualisation media can support (Interaction §5.4). We have added in the following areas to the scope category: requirements, design, and method, and added in new elements to the performance area. We have added in new points to the spatial substrate and marks and properties areas within the form category. We relabeled the presentation area and added some new elements. We created a new visualisation techniques area in the form category. We comprehensively extended the previous interaction category by including two new areas user navigation and user tasks.

This framework represents an iterative improvement over the earlier model for evaluating graphics technologies. We have used the experience gained at applying the first model to SVG [74, 75] in designing this new framework. We are not aware of any other work in the area of evaluating graphics technologies for information visualisation or software visualisation to date. Most evaluations of information visualisation work focuses on user tasks, the tools to build the visualisations, or the actual visualisations themselves. We propose that our framework be continually improved and refined as necessary. In the future we would like to see our framework applied to other graphics technologies for software visualisation to further help validate our research methodology.

There are some limitations to our framework. Our framework does not look at empirical studies, user evaluation, nor usability testing [171]. Conducting empirical studies, user evaluation, or usability testing are out of scope for this thesis, as we are limiting ourselves to focus only on the graphics technology output of a visualisation.

A recent survey [81] reviewed 65 papers describing new information visualisation applications or techniques. 11 papers stated that a user evaluation was part of future work, while 12 papers did some form of evaluation. Of the 12 papers only two user studies were considered successful. A number of papers highlight some of the problems of evaluating information visualisations [8, 132, 190, 230, 251]. From these papers it is not clear what the best way forward or which approach is best for software visualisation evaluation and it is something we would like to explore in the future as a possible extension to our new framework.

We do not consider dealing with senses such as touch or smell as we are primarily concerned with the vision and sound senses. The aim of our work is targeted toward standard computers a software developer would use on a daily basis. So we only consider 2D computer display technologies that represent 3D by means

of perspective and animation. Other technologies like augmented reality [99] and virtual reality [214, 244, 259] could be integrated into the framework at a later date. Although we are evaluating a new technology we are not predicting where software visualisation is heading in the future. Instead, we are trying to replicate standard software visualisation techniques from the literature with a new graphics technology.

Holmberg et al. [107] have created a framework for evaluating interactive web-based visualisation technologies and briefly compare SVG, DHTML, VRML, X3D, and Java3D. Their framework considers only generic web-based visualisations, while our framework can cater for any graphics technologies but is aimed at the domain of software visualisation. Our framework is also not bound to any particular platform such as the web. All of their categories and most of their measures are subsets of our framework.

The next chapter applies the evaluation framework presented in this chapter to the X3D software visualisation case studies (§4) and to our use of the X3D specification [265].

# Chapter 6

## Evaluation of X3D

### Contents

---

<b>6.1</b>	<b>Scope . . . . .</b>	<b>132</b>
6.1.1	Requirements . . . . .	132
6.1.2	Design . . . . .	133
6.1.3	Method . . . . .	134
6.1.4	Performance . . . . .	137
<b>6.2</b>	<b>Form . . . . .</b>	<b>138</b>
6.2.1	Graphical Capability . . . . .	138
6.2.2	Presentation . . . . .	141
6.2.3	Visualisation Techniques . . . . .	143
<b>6.3</b>	<b>Interaction . . . . .</b>	<b>146</b>
6.3.1	User Controls . . . . .	146
6.3.2	User Navigation . . . . .	148
6.3.3	User Tasks . . . . .	151
<b>6.4</b>	<b>Discussion . . . . .</b>	<b>152</b>
6.4.1	Advantages of X3D . . . . .	153
6.4.2	Disadvantages of X3D . . . . .	154
6.4.3	Potential Improvements for X3D . . . . .	156

---

We now apply our software visualisation media evaluation framework (§5) to evaluate X3D for software visualisation based on our analysis (§3) and experience with X3D (§4). We apply the three categories of the framework: Scope, Form, and Interaction and then summarise our findings.

## 6.1 Scope

### 6.1.1 Requirements

We would like to evaluate X3D for use in software visualisation and in particular with respect to our ongoing software visualisation project (§2.3). We want to use X3D software visualisations over the web and to be able to represent the visualisations in 3D.

- **Task:** In this thesis we want to find out if X3D is a suitable medium for representing software as visualisations. Our ultimate goal is to create visualisations to understand software for the purposes of software reuse, maintenance, re-engineering, and reverse engineering (§2.3). The aim of visualising existing software components is to understand if and how a given piece of code can be reused in a new software program or modified to change the behaviour [160].
- **Audience:** The target audience for our software visualisation media evaluation research will be developers who want to create software visualisations and software visualisation tools. The users of our software visualisations are developers that come from the areas of software reuse, software maintenance, re-engineering, and reverse engineering.
- **Target:** The data source for our software visualisations are XML execution traces that can gather static or capture dynamic information about a software component (§2.3.3). Tracing the execution of a component involves catching information such as method calls, method returns, field accesses, field modifications, object creations, and object deletions. This information can show potential consequences and alternative executions that can be created by overloading or replacing certain parts of a component.
- **Representation:** We would like to be able to implement an exemplary range of software visualisation techniques in X3D that can do most things a developer would like to see about a software component. Some example software visualisations that we have implemented include algorithm animations (§4.2),

UML diagrams (§4.3), documentation-related visualisations (§4.4), and execution trace visualisations (§4.5) represented as 3D compound shapes and 3D information visualisation metaphors.

- **Medium:** The aim of our visualisation architecture is to be able to create and view visualisations of remotely executing software over the web (§4.1). For the purposes of the architecture we would like to display software visualisations in widely used web browsers such as Microsoft Internet Explorer and Mozilla Firefox, and executing on a standard computer a software developer would use on a daily basis. Colleagues of our research group have previously looked at 2D technologies for desktop (Tcl/Tk) and web based applications (SVG) (§2.3.4).

### 6.1.2 Design

X3D has been designed as a lightweight file format for the web and has an XML encoding. There are a number of X3D implementations that operate on a variety of operating systems.

- **Purpose:** The aim of X3D is to provide a royalty-free open standards file format (§3.1) and run-time architecture to represent and communicate 3D scenes and objects using XML (§3.2.2).
- **Environment:** The target environment of X3D is the web and the Internet (§3.1).
- **Implementations:** We looked at three of the main X3D browser implementations which were either commercial (BS Contact VRML/X3D Player, Flux Player) or free-ware (Octaga Player) (§3.4.1). We also looked at the Xj3D browser which is the open source implementation to experiment with the X3D specification. Finally, there are about 10 other implementations that exist and some of them are either one person development teams, beta-versions, designed predominantly for VRML, or do not have plug-ins to widely used browsers.
- **Operating System:** BS Contact VRML/X3D Player, Flux Player, Octaga Player, and Xj3D Browser all operate on Windows. Octaga, Xj3D, and FreeWRL operate on Linux (§3.4.1). FreeWRL is the only other browser we looked at that operates on MacOSX.
- **Hardware:** X3D was designed to operate in web browsers on personal computers. Our testing was primarily conducted on a Dell Latitude D610

laptop running Windows XP with a Intel Pentium 1.86GHz processor and 512MB memory. Our aim is to test the media on computers that the average developer would use on a daily basis when developing software. We also used a Dell PowerEdge 1850 Windows server with two Pentium IV Xeon 2.8GHz processors and 2GB of memory. When using the server for testing we did not have any hardware acceleration. We would recommend using a desktop or laptop that has slightly more processing power and at least 1GB of memory as some of our more sophisticated visualisations struggled when displayed on our test laptop (§4.3 and 4.5.2).

- **Learning:** Drawing simple X3D scenes took about a week to accomplish and about a month to produce a reasonable X3D software visualisation by hand. We were finally very competent at using the X3D language after a few months. The hardest part in the learning process for us was knowing how to apply 3D mathematics to create complex visualisations since that is not our background, which is independent of learning X3D.
- **Production Use:** Our software visualisation prototype tools have only been used in our research environment. We are yet to use the prototype tools within a teaching or an industrial setting. Our aim would be to make our tools open source to allow other developers to make use of them. We are aware of one other academic research project using X3D for UML class diagrams (§2.2.4 and Figure 2.11) and their tools are not publicly accessible either.
- **Empirical Evaluation:** Our X3D software visualisation prototype tools and visualisations have had no empirical evaluation and we wish to consider this as part of future work. We would like to do some evaluation on whether 2D or 3D web based software visualisations is best (§2.1.1). Another area to explore is the cognitive issues [188] for representing software structures in 3D. We are aware of no other empirical evaluations when using X3D for software or information visualisation.

### 6.1.3 Method

X3D can be created by hand but for complex software visualisations it is best to automate the creation process by using XML based languages like XSLT. Some of the X3D browsers can be plugged into commercial wide web browsers such as Mozilla Firefox and Microsoft Internet Explorer.

- **Creation:** We found the best way to create basic X3D software visualisations



was to use text editors. Figure 6.1 shows editing the insertion sort algorithm animation (§4.2.3) in a simple text editor. Creating complex software visualisations by hand, however, was incredibly arduous. Instead, we found it best to use XSLT transformations and wrote the XSLT code in a text editor. We then processed the XSLT stylesheets with the execution traces using an XSLT processor either on the command line or from our web application, VARE-3D (§4.1).

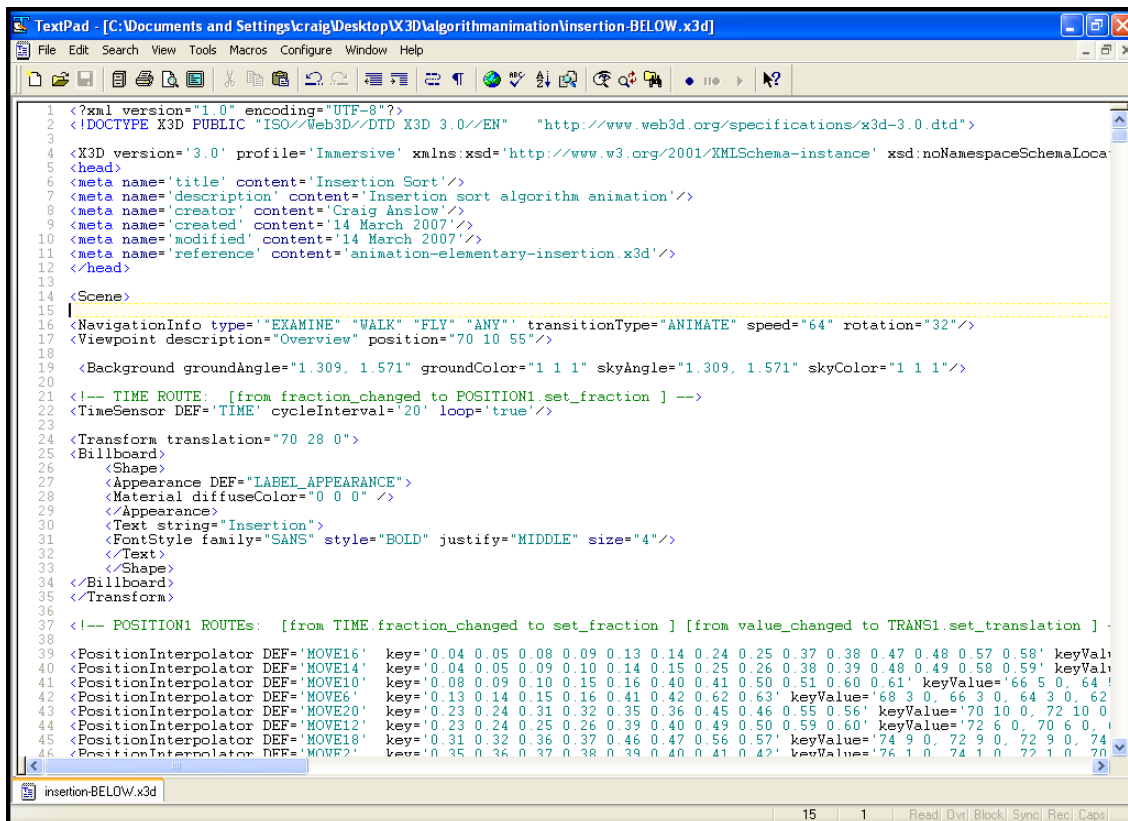
We also experimented with two digital content creation tools X3D-Edit [40] and Flux Studio [180] for editing X3D content but they were more designed for non-programmers (§3.4.2). The first version of X3D-Edit displays the XML in a graphical tree which required a lot of mouse manipulation by the user, and was very computer resource hungry possibly due to it being built on top of IBM's Xena tool. The second version is a plug-in to the Netbeans IDE and incorporates the Xj3D viewer for previewing X3D content. There is no Eclipse plug-in yet. Flux Studio has a WYSIWYG (What You See Is What You Get) style, designed for non-programmers, and it is hard to get at the raw X3D content. Flux Studio has an option of being able to quickly view an X3D file via a preview button which opens the file in the Flux Player. Since our software visualisations are mainly constructed from XML execution traces there was no real need to use these kind of digital content creation tools.

Writing X3D by hand was best suited for writing components which could be used in applications, XSLT, or adjusting the output from the export of another tool.

- **Viewing and Deployment:** A widely used web browser such as Internet Explorer or Mozilla Firefox can be used to view the X3D software visualisations with an X3D browser plug-in (§3.4.1). Alternatively a user can use a stand-alone X3D web browser. X3D files can be located on the local file system or on the web. The National Institute of Standards and Technology (NIST) have a web page<sup>1</sup> that detects if any X3D and VRML plug-ins are installed in a user's web browser. In general we found that when browsing to an example X3D file on the web using Internet Explorer or Mozilla Firefox, if we did not already have an X3D browser plug-in installed, that the commercial browsers did not understand the X3D file type and did not recommend any X3D browsers to install.
- **Integration:** X3D has an XML encoding (§3.2.2) so it can quite easily integrate with other XML languages and XML tools such as XSLT. Figure 6.2

---

<sup>1</sup><http://cic.nist.gov/vrml/vbdetect.html>



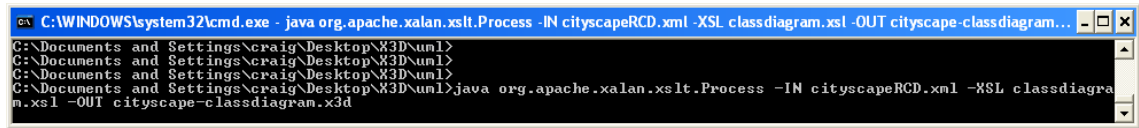
```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d-3.0.dtd">
3
4 <X3D version="3.0" profile="Immersive" xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance" xsd:noNamespaceSchemaLoca
5 <head>
6 <meta name="title" content="Insertion Sort"/>
7 <meta name="description" content="Insertion sort algorithm animation"/>
8 <meta name="creator" content="Craig Anslow"/>
9 <meta name="created" content="14 March 2007"/>
10 <meta name="modified" content="14 March 2007"/>
11 <meta name="reference" content="animation-elementary-insertion.x3d"/>
12 </head>
13
14 <Scene>
15
16 <NavigationInfo type="EXAMINE" "WALK" "FLY" "ANY" transitionType="ANIMATE" speed="64" rotation="32"/>
17 <Viewpoint description="Overview" position="70 10 55"/>
18
19 <Background groundAngle="1.309, 1.571" groundColor="1 1 1" skyAngle="1.309, 1.571" skyColor="1 1 1"/>
20
21 <!-- TIME ROUTE: [from fraction_changed to POSITION1.set_fraction ] -->
22 <TimeSensor DEF="TIME" cycleInterval="20" loop="true"/>
23
24 <Transform translation="70 28 0">
25 <Billboard>
26 <Shape>
27 <Appearance DEF="LABEL_APPEARANCE">
28 <Material diffuseColor="0 0 0" />
29 </Appearance>
30 <Text string="Insertion">
31 <FontStyle family="SANS" style="BOLD" justify="MIDDLE" size="4"/>
32 </Text>
33 </Shape>
34 </Billboard>
35 </Transform>
36
37 <!-- POSITION1 ROUTES: [from TIME.fraction_changed to set_fraction ] [from value_changed to TRANS1.set_translation ] -
38
39 <PositionInterpolator DEF="MOVE16" key="0 04 0 05 0 08 0 09 0 13 0 14 0 24 0 25 0 37 0 38 0 47 0 48 0 57 0 58" keyVal
40 <PositionInterpolator DEF="MOVE14" key="0 04 0 05 0 09 0 10 0 14 0 15 0 25 0 26 0 38 0 39 0 48 0 49 0 58 0 59" keyVal
41 <PositionInterpolator DEF="MOVE10" key="0 09 0 09 0 10 0 15 0 16 0 40 0 41 0 50 0 51 0 60 0 61" keyValue="66 5 0, 64
42 <PositionInterpolator DEF="MOVE5" key="0 13 0 14 0 15 0 16 0 41 0 42 0 62 0 63" keyValue="68 3 0, 66 3 0, 64 3 0, 62
43 <PositionInterpolator DEF="MOVE20" key="0 23 0 24 0 31 0 32 0 35 0 36 0 45 0 46 0 55 0 56" keyValue="70 10 0, 72 10 0
44 <PositionInterpolator DEF="MOVE12" key="0 23 0 24 0 25 0 26 0 39 0 40 0 49 0 50 0 59 0 60" keyValue="72 6 0, 70 6 0,
45 <PositionInterpolator DEF="MOVE18" key="0 31 0 32 0 36 0 37 0 46 0 47 0 56 0 57" keyValue="74 9 0, 72 9 0, 72 9 0, 74
46 <PositionInterpolator DEF="MOVE2" key="0 35 0 36 0 37 0 38 0 39 0 40 0 41 0 42" keyValue="76 1 0, 74 1 0, 72 1 0, 70

```

Figure 6.1: Editing X3D file in a text editor.

shows creating an X3D file from an XSL stylesheet using the Apache Xalan XSLT engine. X3D has specific language bindings for ECMAScript (implemented as JavaScript) and Java which allowed us to create complex software visualisations (§4.1, 4.3 and 4.5).



```

C:\WINDOWS\system32\cmd.exe - java org.apache.xalan.xslt.Process -IN cityscapeRCD.xml -XSL classdiagram.xsl -OUT cityscape-classdiagram...
C:\Documents and Settings\craig\Desktop\X3D\uml>
C:\Documents and Settings\craig\Desktop\X3D\uml>
C:\Documents and Settings\craig\Desktop\X3D\uml>java org.apache.xalan.xslt.Process -IN cityscapeRCD.xml -XSL classdiagram.xsl -OUT cityscape-classdiagram.x3d

```

Figure 6.2: Creating an X3D software visualisation from the command line using the Apache Xalan XSLT processor.

### 6.1.4 Performance

We use XML execution traces for our data sets. We can produce a large range of software visualisations implemented in X3D which can scale to a large number of nodes and classes.

- **Data Sets:** Our input data was XML execution traces (§2.3.3). Converting other text based formats into X3D is also possible, but domain specific applications or converters would need to be created. X3D also allows other formats to be converted or exported into X3D including: VRML, Keyhole Markup Language (KML) (used in Google Earth), COLLADA (§2.3.5), and from proprietary applications like 3D Studio Max and Maya (§3.4.3).
- **Types of Visualisations:** We produced a range of visualisations including algorithm animations, UML diagrams, source-code related, and large node-link diagrams from execution traces (§4). A disappointing factor is that when rendering the same visualisation some of the X3D browsers display a slightly different output as they don't implement all of the specification. As an example we found only one browser (BS Contact) that actually implements the 2D geometry component of the specification. X3D was best at producing visualisations represented as compound shapes to show either the structure or the behaviour of software (§4.5).
- **Scalability:** X3D can safely render 10,000 - 30,000 nodes within 10 seconds, but 100,000 nodes (Figure 4.32) took approximately 10 minutes to render and the software visualisation was basically unusable in terms of navigation and discovery (§4.5.2). Perhaps this was due to the specific hardware and

graphics card we were using or the X3D browser implementation. More testing is required on machines with a higher specification to give a more conclusive result. X3D did not scale well when there was large amounts of text as the rendering speed was severely affected, again navigation and discovery were the issues (§4.3 and Figure 4.17).

- **File Size:** The size of our algorithm animations (§4.2) files were less than 100KB. The size of our UML diagrams varied (§4.3). Our large UML class diagram of Eclipse was 2MB and the other diagrams were less than 100KB. The size of our execution trace visualisations (§4.5) were 2MB (10,000 nodes), 10MB (50,000 nodes), and 18MB (100,000 nodes). Our small execution traces took less than a few seconds to convert into X3D software visualisations using XSLT. Our much larger execution traces took no longer than a couple of minutes when executed by our tool (§4.1) or from the command line (Figure 6.2). We also converted our execution trace visualisation files into the X3D binary format using the Xj3D converter (§3.4.3), which reduced the size of the files by about 75%. Currently only the Xj3D browser (§3.4.1) supports the binary encoding. The binary encoding has only just recently become an ISO standard and we would expect other X3D browser implementations to adopt this encoding in the future.

## 6.2 Form

### 6.2.1 Graphical Capability

X3D performs very well for the graphical capabilities section. X3D has a vast range of features to support the different facets of the graphical capabilities section. Where X3D does not have built-in support for a certain capability in this section, the capability can often be accomplished using scripting languages.

#### Spatial Substrate

- **Dimension Support:** X3D can support both 2D and 3D dimensions for displaying software visualisations. We are yet to explore if X3D can support dimensions greater than 3D, but we have no need for greater than 3D at the moment in our software visualisations.
- **Composition:** Axes can be placed orthogonally to create a metric space similar to Film Finder [45] and can make use of the third dimension.

- **Alignment:** Axes can be repeated at different positions in space. We used the inline node (§3.3.2), which embeds an X3D scene stored at a location on the web or local file system into the current scene, for repeated axes to display three different algorithms all animating at once (§4.2.3).
- **Folding:** Information can be folded along the X, Y or Z axes since X3D has three dimensions, similar to the visualisations produced by SeeSoft [79] for two dimensions.
- **Recursion:** Space can be repeatedly divided like a Tree Map [227]. To implement a Tree Map would require using functions in scripting languages since X3D is primarily a representation file format. There are no X3D built-in features for repeatedly subdividing space.
- **Overloading:** The same space can be reused for the same data set by using a switch node (§3.3.2). The switch node can be used in conjunction with a touch sensor or a proximity sensor to show different aspects of an element in the same space within a visualisation. For example we can change our execution trace visualisations (§4.5) to show the details of the event (e.g. the name of the method call or object that was created) once a user either touches the node or navigates within a certain proximity of the node.
- **Axis Distortion:** Axes can be distorted. Munzner et al. [166, 167] (see Figure 2.4) showed how to visualise the structure of the web in 3D hyperbolic space using VRML and C++. This can now be implemented using X3D and Java.

### Graphical Marks and Properties

- **Marks:** X3D supports: points, lines, areas, and volumes. Points and lines can be implemented in a software visualisation using 2D geometry, while areas and volumes can be implemented using 3D geometry (§3.3.1).
- **Size:** Shapes that are not defined by coordinates have size, height, or radius fields (§3.3.1) which can be set during implementation or animated to change in a software visualisation using the routing event model (§3.3.5).
- **Value:** The material node supports the gray scale colour range which can adjust the value of a node (§3.3.1).
- **Texture:** Textures can be applied to nodes and links by either using colour, images, sounds, or video (§3.3.1). We have used all of these kinds of textures in every one of our software visualisations (§4).

- **Colour:** Colour can be used to reveal the state of an animation, highlight areas of interest, and emphasise visual patterns, as well as colouring of nodes and links. The colour of a node can be specified using the `diffuseColor` field inside the material node or specified as its own node (§3.3.1). Colours are defined as RGB colours.
- **Orientation:** Shapes can be positioned and rotated in a visualisation by the transform node (§3.3.2) and can change position or rotate by the routing event model (§3.3.5). Shapes can be orientated in any order (e.g. translated then rotated and vice versa).
- **Transparency:** The transparency field in the material node specifies how clear an object is, with 1.0 being completely transparent, and 0.0 completely opaque (§3.3.1). We used transparency in some of our execution trace visualisations (§4.5.3 and Figure 4.36). We found that when nesting transparent objects the outer levels need to have a higher level of transparency otherwise inner level objects will not be visible. So multiple levels of transparency are allowed.
- **Connection:** X3D can represent graph and tree like structures (§4.5.3). X3D can only show the graphical representation using geometry primitives. X3D can't show the underlying data relationships very well since there is no support for modeling relationship information. We can encode the relationship data in the X3D code but it is hard and time consuming. It is best to represent relationship information using domain specific prototypes (§3.3.6).
- **Enclosure:** Hierarchies can be encoded such as UML class diagrams or graph like hierarchies, but again like the connection item above X3D can only show representation. Actual data is hard to encode into the X3D code and it is best done with domain specific prototypes (§3.3.6).
- **Text:** X3D has support for displaying text using standard fonts and the various combinations and effects that can be applied, such as font faces, styling, and size. Since text is mainly 2D and read from left to right, we found the billboard node (§3.3.3) to be very useful when the user rotated the Z axis of the viewer which allowed the text to rotate and be read clearly (§4.3.4). Unfortunately, the billboard node does not allow the text to be legible if the animation is turned upside down.
- **Advanced Graphics:** X3D has special profiles for Non-Uniform, Rational B-Spline (NURBS), representing humans (H-ANIM), and geography specific

detail (GeoSpatial). These kind of special profiles might be useful if we were to implement a 3D city metaphor (§2.2.2).

There is no specific software visualisation component or profile. We have implemented some domain specific prototypes (§4.3 and 4.5) that could be turned into a software visualisation component in the X3D specification. Having a software visualisation or information visualisation component in the X3D specification would allow X3D browser vendors to decide if they would like to support this component. If the X3D browser vendors implemented the visualisation component then this would make it much easier for developers to create information or software visualisations using X3D as they would not have to create their own domain specific prototypes.

### Temporal Encoding

- Encoding Time: Graphics can be changed to show the passage of time by using the X3D routing event model (§3.3.5).
- Encoding Identity: Different properties of a mark can be used to encode identity. In our algorithm animations (§4.2) we have used colour and size to distinguish between elements, and shapes and colour to distinguish between event types in our execution trace visualisations (§4.5).
- Variation of Retinal Encoding: Almost all graphical attributes of X3D nodes can be changed using the routing event mode (§3.3.5). In our algorithm animations and software visualisations examples the target nodes that we changed were 3D geometry (boxes, spheres), material (colour), and coordinate points (inside coordinate nodes).

#### 6.2.2 Presentation

Data can be encoded in X3D software visualisations using three different approaches. X3D does not support layout constraints of the data in a software visualisation, but the data can be animated and supplemented with sound and video. X3D can represent software visualisations of various programming languages and can even show program synchronisation and multiple views of the data in a visualisation.

- Data Display Independence: Data can be encoded into a visualisation like we have done for our algorithm animations (§4.2), but it would be better to separate the data from the graphical presentation. Since X3D has an XML encoding it is possible to encode the data in XML and use XSLT to

transform the data into X3D nodes before displaying the visualisation. The transformation could also happen at run-time by encoding a style-sheet into the X3D file and using a scripting language function. For our execution trace visualisations (§4.5) we encoded the data as X3D geometry and used JavaScript functions for the layout and control of the data. For our UML diagrams (§4.3) we encoded the data we wanted to display into domain specific prototype instances (§3.3.6) and then used an JavaScript function to change the prototype instances into geometry primitives.

- **Referencable Objects:** Nodes can have a label using the DEF attribute, which allows a node to be referenced by other elements in a visualisation by using the USE attribute (§3.3.1). The DEF attribute is only a textual description so it is important when implementing a visualisation that all nodes that use the DEF attribute are unique.
- **Layout Constraints:** Layout constraints and layout algorithms are not supported in X3D. X3D is primarily a graphics representation file format, which could be used as the output for the implementation of a layout algorithm in a software visualisation. JavaScript functions can be embedded inside an X3D file to implement a lightweight layout algorithm or constraint system (§4.5.2 and Figure 4.29).
- **Animation:** Mouse sensors (drag or click) and interpolators (position, coordinates, colour) can be used to create motion or animation using the routing event model. A time sensor can be used to control time cycle intervals that can be looped, paused, and restarted (§3.3.5).
- **Other Modalities:** Sound in X3D allows WAV, MIDI or MP3 files to be played (§3.3.4) and could be used to demonstrate break points, error detection, or sorting in an algorithm animation (§4.2.4). Video is provided through the MPEG-4 profile and we have used video to show people describing software alongside a software visualisation (§4.4.3 and Figure 4.23).
- **Programming Languages:** We can show visualisations of Java and C++ programs (§4), but there should be no reason why X3D could not be used to represent software visualisations of other programming languages and paradigms.
- **Program Synchronisation:** We have shown multiple simultaneous algorithm animations (§4.2.3). In the algorithm animation examples the data itself was encoded into the actual X3D software visualisations. We have not shown any multi-threaded visualisations or multiple trace visualisations. Our execution



traces can capture multi-threaded information (§2.3.3). If we transformed a multi-threaded execution trace into a visualisation we could use different colours for the different traces and position the information in a different position in the scene. Likewise we could show multiple traces of the same program in different colours. Both visualisations would have to be located at different places in the X3D scene.

- **Multiple Views:** We have accomplished multiple views by having multiple frames within the web interface and loading two different visualisations of the same data source (§4.4). Alternatively multiple views could also be accomplished within the same X3D software visualisation, but if one of the views (e.g. source code, API documentation) is inherently designed for 2D viewing then it maybe be best to show multiple web frames.

### 6.2.3 Visualisation Techniques

X3D can support a range of visualisation techniques from compound shapes to more complex visualisation metaphors. There are unfortunately no domain specific components or libraries developers can use to create software visualisations, but it is possible to implement existing software visualisation techniques.

- **Compound shapes:** We implemented 3D cubes and 3D spirals (§4.5.2). Other 3D shapes are definitely possible. We used a JavaScript function to achieve our goals since X3D does not actually support this kind of metaphor, because it is primarily a presentation format. Figure 6.3 shows 50,000 elements from the Eclipse execution trace represented as a 3D cube.
- **Information Visualisation Metaphors:** We implemented an Information Landscape [7, 247], and an Information Cube [213] (§4.5.3). We used some JavaScript function to layout the nodes and edges in the visualisations. Other metaphors [45] are also possible but we have not explored them yet as we only had time to implement a sample of information visualisation metaphors.
- **Software Visualisation Techniques:** X3D can support a range of visualisations including algorithm animations (§4.2), UML diagrams (§4.3), documentation-related visualisations (§4.4), and execution trace visualisations (§4.5). Figure 6.4 shows a UML package diagram of the Eclipse Java program which shows 257 packages.
- **Pictorial Representations:** X3D can support all three types of pictorial representations [177]. We found X3D was excellent for graph-based displays but

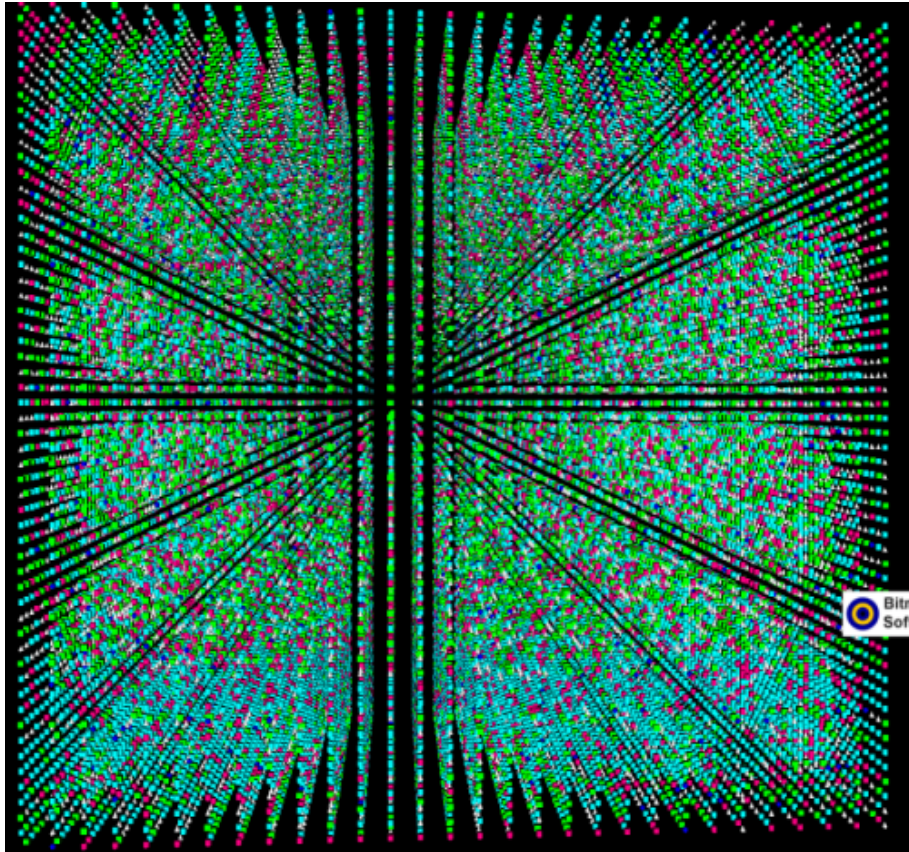


Figure 6.3: Eclipse execution trace — 50,000 events.

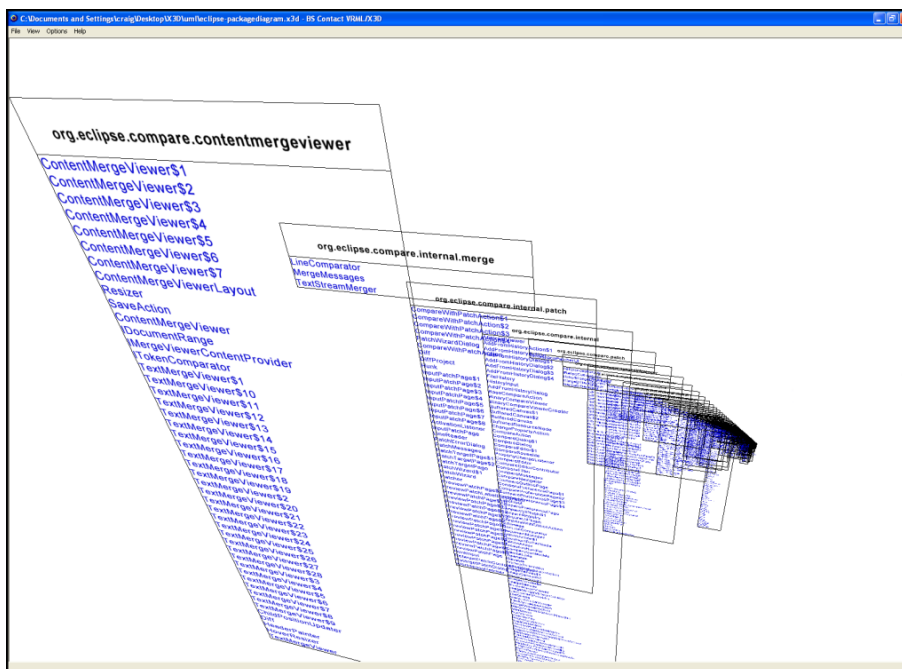


Figure 6.4: Eclipse UML package diagram — 257 packages

not so good for rendering text for source-code-related displays. We are yet to experiment with statistical based displays for software visualisation.

- Graph Based: The main elements of X3D are nodes and the links between the nodes. Attributes can be added to nodes and links to encode information such as shape, appearance, and behaviour. Edges can be created between node objects (§4.5.3).
- Statistical Based: We are yet to implement any software visualisations that display statistical information. We could create statistics based software visualisations similar to that of some other work that shows the citations of String Theory from 1981-2001 [89] and show over 3000 nodes. We should note that many statistical graphs require complex layouts and there is no high level support for these features in X3D.
- Source-Code-Related: X3D has an XML encoding and integrates easily with HTML. We created a software visualisation which showed the visualisation in one display and another display displaying source code in HTML (§4.4.1). Showing the actual source code in X3D can be done, but reading the source code when displayed in 3D is a hard task when the view rotates. We can also display the source code text in a dashboard so that it would be visible when the viewer rotates.
- Model Frameworks: We can implement all the software visualisation frameworks by Keown [118] in X3D, except the metric centric framework as we do not focus on metrics. The hierarchy centric framework focuses on the class hierarchy and uses the majority of available 3D properties to represent properties of the hierarchy. The inheritance centric framework allows a class to be viewed in the context of all classes which it is derived. The method centric framework shows a class representation together with representations of its methods. The property centric framework is where a central class is visualised together with representations of the properties that compose the class. Finally, the single class centric framework visualises a single class, from the system showing as much detail as required.

## 6.3 Interaction

### 6.3.1 User Controls

X3D supports basic user controls, but for more sophisticated software visualisation controls scripting is required.

- **Graphic Changeability:** Graphical elements such as position, coordinates, colour, size, scale, transparency, textures, and visibility can change at run-time via the X3D routing event model with user mouse or keyboard controls (§3.3.5).
- **Input Events:** Users can interact with a software visualisation by keyboard or mouse input (§3.3.5). Navigation is controlled by mouse input but it is possible to use game like keyboard controls, ADWS keys for left, right, up, and down. Mouse interaction is usually mouse selecting, mouse overs, or dragging nodes. Users can also type messages to control objects in a scene as well.
- **Computation:** X3D detects input events by touch, drag, key, and time sensors (§3.3.5). Graphics can then be changed by interpolators which determine what colour, position, orientation, or coordinates of a node changes and at what time during the visualisation defined by frames. A time sensor is used to link the input events and the interpolators. Multiple ROUTE directives are used to link all the different parts of the computation. One thing to note is that nodes don't have to be controlled by user input, they can be controlled by just a time sensor, which is what we have implemented for some of our algorithm animations.
- **Elision Control:** X3D supports eliding information or suppressing detail from the display by controlling the rendering technique to view vertices, wire-frame, flat, or smooth nodes. Figure 6.5 shows the information cube visualisation from Figure 4.36 with the graphics rendered as a wireframe. The lighting component allows objects in the world to be illuminated in various ways (§3.3.4). Some browsers have implemented a headlight control that allows a user to make all objects in a world be their defined colour when the headlight is turned on and black when the headlight is turned off. We discuss more elision type techniques for filtering, in Section 6.3.3.
- **Temporal Control:** X3D supports start and stop buttons. More complex buttons such as pause, fast-forward, rewind, step, and speed control require use of scripting. Some of these buttons were implemented in our algorithm

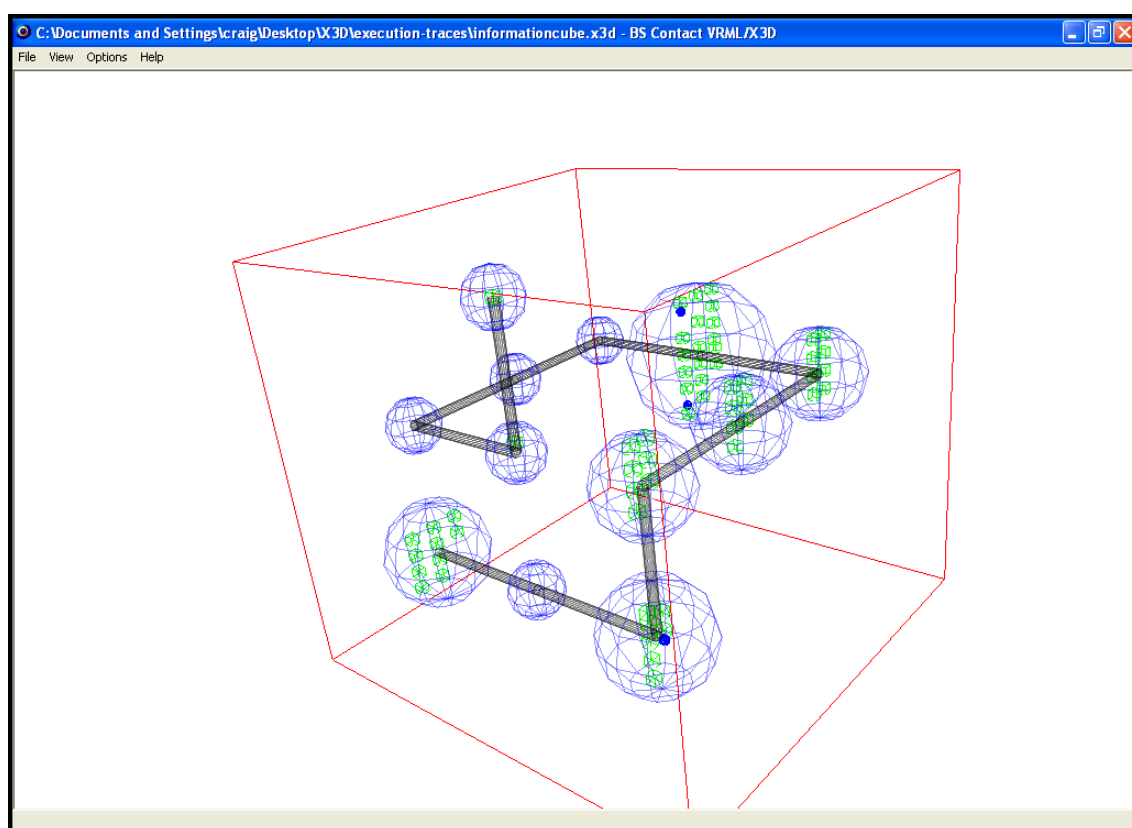


Figure 6.5: Information Cube — wireframe rendering.

animations (§4.2 and Figure 6.6). We also had to control the timer by a script as once a button was selected by a user the timer kept on going within the visualisation. When a user presses the stop button the objects stopped animating, but the timer kept on going. When a user then pressed the start button the objects would then restart in a different position from where they had originally stopped. In order to solve this problem we had to scale the timer using a script function so that objects did stop, pause, and start as a user would expect when using buttons of this nature.



Figure 6.6: Algorithm animation user controls.

- **User Notation:** We haven't created any user notation visualisations, tools, or controls to create software notations in X3D online. User notation has not been the focus of our work as we are interested in encoding information we already have from our XML execution traces (§2.3.3) into a software visualisation. We would like to explore user notations implemented in X3D in the future and that it would require heavy use of scripting languages and Java.

### 6.3.2 User Navigation

X3D has very good support for user navigation and can support a range of techniques.

- **Navigation Types:** X3D supports a range of 3D navigation techniques (§3.3.3) for a user to navigate in a scene. We found that not all of the browsers implemented all of the navigation techniques, and some have different names for the same navigation technique.
- **Navigation Control:** A user can change the navigation type at run-time by either right clicking in a browser, see Figure 6.7, or using one of the buttons in the tool bar of a stand-alone X3D browser (§3.4.1). The Flux X3D browser plug-in actually makes their navigation control buttons visible inside widely used browsers. The navigation info node (§3.3.3) allows a developer to set

the kinds of navigation which can be applied to a software visualisation, likewise set the default navigation option when the X3D browser loads the visualisation.

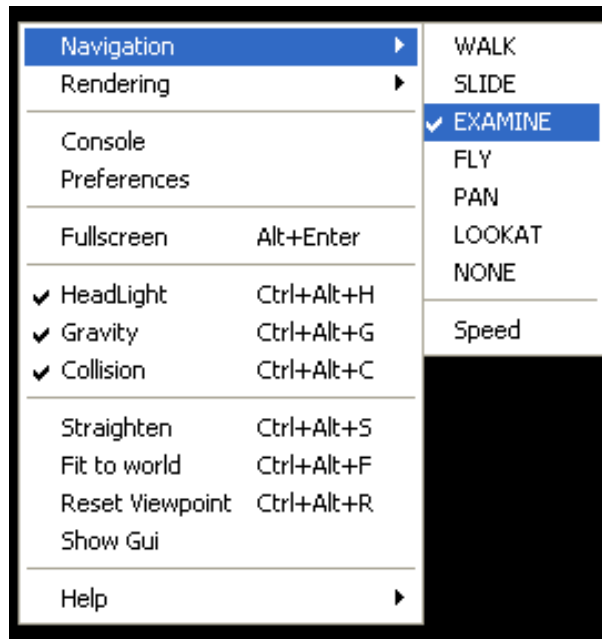


Figure 6.7: X3D Navigation — navigation options by right clicking in the Octaga Player.

- **View Refinement:** Users can change the point of view in a software visualisation using a viewpoint (§3.3.3) by selecting the viewpoint from the viewpoints menu. Figure 6.8 shows the available viewpoints (Overview, Main Method, Right, Left) from the information landscape visualisation from Figure 4.35. There is no way to create or save a viewpoint at run-time. Viewpoints can be distinguished by textual descriptions, but there is no way to search for a specific viewpoint, instead they are just listed in order when the X3D browser renders the scene. Navigating to a particular viewpoint in a large software visualisation can be quite cumbersome when there are many viewpoints defined in a scene. To increase view refinement efficiency it would be great if viewpoints could be mapped to keyboard shortcuts.
- **Speed:** A user can control the navigation speed by right clicking to show the properties menu in a web browser and then selecting the navigation option followed by the speed option, similar to Figure 6.7. The rotation speed of nodes can also be controlled at run-time. The ranges for navigation and rotation speed are from zero to infinite and all browsers implement them differently, some use base two increments, while others use more descriptive

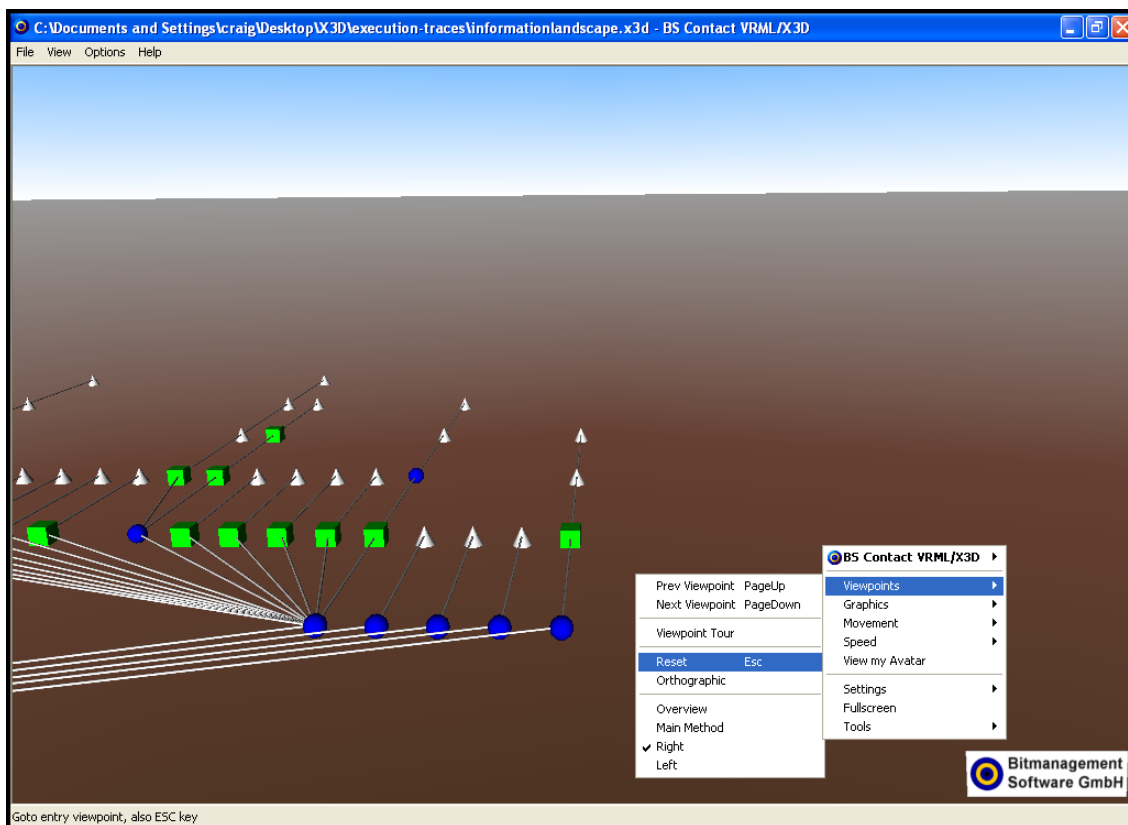


Figure 6.8: Information Landscape — showing available viewpoints.



tags such as slow, fast, fastest, or very fast. The navigation and rotation speed can also be initially set in the navigation info node (§3.3.3).

### 6.3.3 User Tasks

X3D supports providing an overview and zooming. X3D has features for filtering and showing details on demand, but both require combining various X3D nodes to produce a good visual effect. Relationships, a history of actions, and extraction of data are not supported, but are possible through the X3D browser run-time API.

- **Overview:** A user can gain an overview of the entire software visualisation, only if a specific viewpoint is defined which a user can select and that the viewpoint encompasses the whole data set (§3.3.3). Alternatively a user can zoom out to get an overview as well. When creating a software visualisation it is best to make the default viewpoint to be an overview of the complete visualisation.
- **Zoom:** A user can zoom into items of interest using the built-in X3D browser navigation controls, or by selecting a pre-defined viewpoint which is located at a zoomed in location of a particular item (§6.3.2). The look-at navigation type allows a user to select a node, then the viewer will zoom immediately to some convenient viewing distance from the selected node (§3.3.3).
- **Filter:** The boolean filter node exists for selective routing of true or false values and negation. For this node to be useful for a software visualisation it needs to be used with a script to determine what nodes require filtering out. Some of the browser built-in functions, rendering techniques, and lighting are discussed earlier in elision control (§6.3.1). We also found a number of ways to apply filtering to an X3D scene. Transparency can be used to filter out certain nodes, likewise scaling or changing the size of nodes, once a user clicks or moves a user control (§3.3.1). Geometry can also be moved to different parts of a software visualisation which a user can't see in the current view to give a filtering effect. Alternatively a visibility limit can be set which does not render nodes once they are beyond a specific position in a software visualisation (§3.3.4).
- **Details-on-demand:** When a user navigates to a certain distance from a node or selects a viewpoint the browser can change the physical appearance of a node using the level of detail or switch nodes (§3.3.2). We can show details-on-demand in our execution trace visualisations using these approaches (§4.5). Another approach is to have user controls where a user selects or

moves a control and certain properties are shown about a node somewhere else in the software visualisation or next to the node that requires details. We have done this second approach for our documentation-related software visualisations (§4.4) which uses frames to show the source code, API, or a movie about a particular software component.

- **Relate:** Viewing the relationships among items could mean many things and depending on what the relationships were they could all be implemented in different ways. An example from an execution trace visualisation could be “show all the classes that inherit from an abstract class or show all the objects that call another object”. This could be implemented by modifying the colour or size of nodes or edges, but there maybe a need to have text input. Since X3D is primarily a presentation format there is no method to encode the relationship information about classes or objects in the software visualisations, except if node prototypes (§3.3.6) were used.
- **History:** There is no built-in support for creating a history of user actions. This can be achieved by exploring the SAI run-time API of an X3D browser (§3.4.1) and capturing the events a user performs and then providing options for a user to go back and repeat those actions. Unfortunately not all X3D browser implementations make the SAI available for X3D content developers to use. There is also no support for creating a bookmark or saving a specific viewpoint once a user finds an interesting view in a software visualisation, but this is possible using the SAI.
- **Extract:** X3D is predominantly a presentation format and does not provide features for extracting sub-collections or query parameters of a visualisation. Extracting information might be able to be accomplished using a much larger Java application and looking at the run-time information or the SAI of the browser (§3.4.1).

## 6.4 Discussion

To summarise the previous evaluation sections of the framework we have captured the main points and present them as tables for each of the three categories in Appendix A. We now discuss the advantages and disadvantages of using X3D for software visualisation, and then potential improvements.

### 6.4.1 Advantages of X3D

X3D has a number of features which make it a good medium to use in software visualisation. The advantages are as follows:

**Graphics** X3D has a number of different graphical elements including 2D geometry, 3D geometry, text, textures, lighting, geometry, NURBs, and environment effects which provide an excellent set of features to get high quality visual pictures required in software visualisations (§3.3).

**Extensibility** X3D is designed as a set of components, which are groups of functionality such as geometry or interaction features (§3.2.3). The specification has been designed so that it can be extended in the future by adding in new components when necessary and not affecting already designed components such as NURBs or representing humans with such schemes as H-Anim. Node prototyping can be used to extend X3D features without relying on components to be created or added to the specification (§3.3.6). Developers specify prototypes and then use instances of them which are then mapped to existing geometry primitives or X3D features. Profiles are built from components and are a standardised sets of extensions to meet specific application needs such as immersive virtual reality (§3.2.3).

**XML Integration** X3D has an XML encoding (§3.2.2) which means it can take advantage of a wide variety of tools such as XSLT and Java XML libraries to create and manipulate X3D content. X3D has an accepted DTD and XML Schema. X3D can be embedded in HTML documents. Since XML is very ubiquitous and distributed as ASCII text, learning the syntax and structure of X3D documents is relatively straight forward.

**Multimedia** Sound (§3.3.4) and video (§3.3.1) can be quite easily included in X3D documents by mapping sound and movie textures to geometry. Using sound and video can enhance a software visualisation by providing further information. We used sound to signify ordering of elements in algorithm animations (§4.2) and video for providing information about classes in UML diagrams (§4.4.3).

**Deployment** X3D is a text file format that can be deployed easily over the web so it makes it accessible to anyone with an Internet connection. Users don't have to install large libraries or implement their own viewer in order to view X3D content. Instead one can just install an X3D browser on their computer or plug-in a number of existing X3D browsers to their web browser (§3.4.1).

**Portability** Since X3D is a file format it can be rendered in either OpenGL or DirectX so X3D is not reliant on any particular underlying API platform. There

are a number of X3D browser implementations that work on the following different operating systems Windows, Linux, and MacOS X (§3.4.1). The majority of the X3D browsers are mainly designed for Windows.

**Licensing** X3D is a royalty free open standard, so it is available for public use (§3.2).

## 6.4.2 Disadvantages of X3D

X3D does have some disadvantages which make it hard to use for software visualisation. The most important of these disadvantages are as follows:

**User Controls** No user controls for software visualisation are included. Geometry can be used to create very basic user controls (§3.3.5). More complicated software visualisation controls such as filtering, or algorithm animation controls require heavy use of scripting languages (§4.2.4).

**Animation** X3D relies too heavily on the routing event model (§3.3.5) for animation and user interactions in X3D software visualisations. The design of the routing event model is very cumbersome when there are lots of objects that require animation or manipulation in a visualisation (§4.2.4, 4.3.4, 4.5.4).

**Creation** Creating large software visualisations by hand is very time consuming. Since X3D is predominantly a file format for representing graphics it is better to use other tools and languages like Java and XSLT to create larger X3D software visualisations (§4.1, 4.3, 4.5).

**Filtering** X3D lacks filtering, search, and query capabilities. Since X3D is declarative in nature we found no specific nodes to delete or hide unwanted objects in a software visualisation. Instead we had to hide objects inside other objects, change the scale or size of an object, or use transparency effects (§4.2). We could have also used the switch node (§3.3.2) to achieve the same hidden effect. Nodes can be labeled with a descriptive text name using the non-unique DEF keyword, however, this is the only feature supported for script functions to search or query for types of objects (§3.3.1).

**Layout** Positioning objects in X3D requires using the transform node which has fields for translating an object in the X-Y-Z axes and rotation about an arbitrary point and axis (§3.3.2). With our UML diagrams (§4.3) and execution trace visualisations (§4.5) we have nodes positioned in graph like structures that have node-link relationships. X3D does not support the layout or adjusting of connected nodes very well. When a user attempts to move a node around

a screen or implement more complex layout algorithms such as spring embedder [250], there are no X3D features to preserve the node-link relationship between pairs of nodes. This is due to the XML encoding being declarative and not allowing nodes to have nested grouping relationships overlap. In order to preserve the node-link relationships, Java or scripting languages have to be used, and involves substantial programming.

**Conformance** No tools or processes exist to determine if an X3D application or software visualisation conforms to an X3D implementation, similar to validating HTML web pages with the W3C HTML Validation Service<sup>2</sup>. One could, however, validate X3D content against the X3D XML schema or DTD, but that would require downloading the X3D content and X3D schema to a users machine then opening a separate XML application to validate the file. We are more concerned that no specific tools or web based services exist for this process. There exists an X3D conformance testing program (§3.5.1) but it only intends to promote consistent and reliable tool implementations of the X3D specification by many vendors across multiple platforms. There has been no empirical or user evaluation studies on the ease of use of programming in X3D, nor for any systems or visualisations built using X3D.

**Installation** To view X3D software visualisations in widely used web browsers such as Microsoft Internet Explorer or Mozilla Firefox an X3D plug-in is required (§3.4.1). We found that neither Microsoft Internet Explorer nor Mozilla Firefox detected what kind of plug-in was required when browsing to an X3D software visualisation if no X3D browser plug-in was already installed. Only some of the X3D browser implementations can be plugged into widely used web browsers. The downloads for the X3D browser plug-ins ranged from 1.5MB to 6MB.

**Tool Support** We found that some of the X3D browsers did not implement all of the X3D specification nor do they make the SAI run-time API available. This makes it hard for developers to create X3D software visualisations that are consistent for all X3D browsers (§4.2, 4.3). The size of most of the development teams of the X3D browsers range from one person to teams smaller than 10 people. The problem with the development of the X3D browsers and tools is that they simply do not have the people to develop tools to meet all user requirements of functionality. Since 3D graphics is by nature a specialised field, the existence of more markets for X3D on the web may prompt more tool

---

<sup>2</sup><http://validator.w3.org>

development and hence attract more people to using X3D for 3D graphics on the web.

### 6.4.3 Potential Improvements for X3D

There are a number of potential improvements that are desirable if X3D is to be used in software visualisation.

A significant improvement in X3D would be to have a component dedicated for domain specific software visualisation features (§4.1.3, 6.2.1). Determining what those features are would be the first step. Some work has been done in our case studies using node prototyping for UML diagrams (§4.3) and the implementation of 3D compound shapes from XML execution traces (§4.5).

There are a number of upcoming community improvements to the X3D specification that look like quite promising features for software visualisation. These include the layering, layout, followers, texturing 3D, and picking sensor components. The layering component will allow organising information in a visualisation into independent, overlapping layers. The layout component will allow content to be arranged to appear in specific regions of the display surface. The layering and layout component may help solve some of the current X3D layout issues (§6.4.2). The followers component will allow smooth animations of objects, which may solve the pausing issue in our algorithm animations for us to create continuous smooth animations (§4.2.4). 3D textures will be able to be applied to geometry as well environmental shading texturing, which may be useful to show different aspects of a software visualisation. Finally, picking sensors will allow users to select specific items in a software visualisation.

A vital step would be to minimise the installation overhead as much as possible. In order to overcome this overhead, widely used web browsers such as Microsoft Internet Explorer and Mozilla Firefox should implement the X3D specification into their browsers rather than relying on end users to install a plug-in or stand-alone X3D browser. Implementing X3D in the the widely used web browsers will also facilitate integration of X3D into web software and services.

After minimising the installation overhead the next step would be to provide consistency across the main X3D browsers and plug-ins. Even though some X3D browsers only aim to implement certain aspects of the specification it is important that the main X3D browsers are consistent so that there is a similar user experience.

The next chapter presents our conclusions, contributions, and future work.

# Chapter 7

## Conclusions

In this chapter we offer our conclusions. We summarise the contents of each chapter, outline our contributions, and look at potential work for the future.

In Chapter 2 we gave an introduction to information visualisation, looked at various ways to display information using different visualisation techniques, described what software visualisation is, followed by a close look at existing 3D software visualisation systems. Finally, we gave an overview of our ongoing software visualisation architecture project and described our motivation for this thesis.

In Chapter 3 we gave a brief overview of X3D, described the contents of the specification, commented on a number of X3D browsers and tools, discussed work in progress to improve the X3D specification, and finally gave our assessment of the suitability of X3D for web based software visualisation.

In Chapter 4 we outlined the contribution of a transformer component from our visualisation architecture project to produce X3D software visualisations from XML execution traces. We then presented a representative sample of software visualisation techniques which are common throughout the software visualisation literature. The case studies included algorithm animations, UML diagrams, documentation-related visualisations, and software visualisations from XML execution traces.

In Chapter 5 we presented a framework for evaluating software visualisation media. The key contribution of this chapter was a major extension to a previous model created by colleagues in our research group, by adding in new components, addressing the needs of 3D, and offering a better grouping of key ideas.

In Chapter 6 we applied the software visualisation media evaluation framework (§5) to evaluate X3D for software visualisation based on our analysis (§3) and experience with X3D (§4). The results of the evaluation showed X3D to be a useful medium for implementing a large range of 3D software visualisation

techniques. Finally, we discussed the advantages and disadvantages for X3D in software visualisation.

The goal of this thesis was to evaluate X3D for use in software visualisation. Our detailed evaluation was presented in Chapter 6. The major advantages of X3D are rich graphics, extensibility, and XML integration. The major disadvantages of X3D are lack of software visualisation user controls, a primitive animation model, and weak support for filtering and layout. Nonetheless we encourage software visualisation developers to adopt X3D if they need 3D for the web.

## 7.1 Contributions

The contributions of this thesis are as follows:

- **X3D Software Visualisation Case Studies (§4)** - we have replicated and discussed the implementation of a representative range of software visualisations in X3D including algorithm animations, UML diagrams, documentation-related visualisations, and execution trace visualisations.
- **VARE-3D (§4.1)** - we have produced a prototype tool which can produce X3D software visualisations from XML execution traces over the web.
- **Software Visualisation Media Evaluation Framework (§5)** - we have create a framework for evaluating software visualisation media or graphics technologies for use in software visualisation based on a previous evaluation model.
- **Evaluation of X3D For Use in Software Visualisation (§6)** - we have applied our software visualisation media evaluation framework to evaluate how good X3D is for software visualisation.

## 7.2 Future Work

This thesis suggests a number of areas for future work.

**Layout information:** An area that was exposed during our case studies for producing visualisations from execution traces (§4.5) is that X3D lacks support for the layout of information about a program. Exploring the use of graph layout languages such as the Graph eXchange Language (GXL) [269] with our X3D software visualisations would be worthwhile to see if the layout features could be improved. GXL is an XML graph exchange format for representing relationships



between nodes and edges. Ideally we would see GXL being an intermediary step between the transformation of our execution traces into X3D software visualisations. Another approach is to create a grammar for representing objects in 3D software visualisation diagrams, similar to the grammar of node-link diagrams produced by Ware [258].

**Data independence:** The data for our algorithm animations (§4.2) was encoded inside the X3D code and the visualisations do not allow a user to change the data or select the algorithm they want to animate. Making the data independent of the presentation of the visualisation (§6.2.2) will allow us to create an interactive system where upon users can select the kind of algorithm animation they want to display and input the data set to use. Another possible avenue for data input would be to extract the most appropriate information from our execution traces (§2.3.3). Adding information about the algorithm such as how it works and performance, as well as stepping through pseudo code as the algorithm animates would be useful features.

**Application of evaluation framework to other media:** The software visualisation media evaluation framework (§5) could be expanded by going into more detail for other modalities such as sound, video, haptics, or virtual reality. Applying the framework to other media such as Flash, Tcl/Tk, Java3D, COLLADA, other 3D languages (§2.3.5), and augmented reality could increase the usefulness of the evaluation conducted for X3D and the previous evaluation model [74, 75].

**VARE integration** There are not many available free software visualisation systems. Most of our previous tools (§2.3.4) have been implementations of different aspects of our VARE architecture project (§2.3). A good next step would be to integrate some of our test driving, repository, and visualisation tools together with VARE-3D. Integrating these tools would create an end to end complete visualisation system for users and would test the validity of VARE. Eventually we could make the complete system publicly available as a web application.



# **Appendix A**

## **Evaluation of X3D Summary**

The following tables Scope (Table A.1), Form (Tables A.2 and A.3), and Interaction (Table A.4) summarise our evaluation for each of the three categories in Chapter 6.

<b>Requirements</b>	
Task	understand software for reuse, maintenance, re-engineering, and reverse engineering
Audience	software visualisations for developers in the areas of reuse, maintenance, re-engineering and reverse engineering
Target	static and dynamic information from software XML execution traces
Representation	3D software visualisations
Medium	over the web
<b>Design</b>	
Purpose	3D graphics open standard for the web that has an XML encoding
Environment	over the web
Implementations	four main implementations, about 10 secondary ones
Operating System	mainly Windows, but some Linux and MacOSX
Hardware	recommend x86 architecture, dedicated video card greater than 1GB of RAM
Learning	basic visualisations a few weeks, very competent after a few months
Production Use	none
Empirical Evaluation	none
<b>Method</b>	
Creation	by hand, XSLT, X3D editor, Java application
Viewing and Deployment	commercial web browser over the web with X3D plug-in or stand-alone X3D browser
Integration	XML oriented, JavaScript, Java
<b>Performance</b>	
Data Sets	XML execution traces
Types of Visualisations	algorithm animation, 3D shapes, 3D information visualisation metaphors, UML diagrams, documentation-related visualisations
Scalability	10K - 30K nodes comfortable and not so good for large amounts of text
File Size	less than 100KB for small files and 10-18MB for large files

Table A.1: Evaluating X3D — Scope.

<b>Graphical Capability</b>	
<i>Spatial Substrate</i>	
Dimensions	2D and 3D
Composition	axes can be placed orthogonally to create a metric
Alignment	axes can be repeated at different positions in space
Folding	along X, Y, or Z axis
Recursion	requires scripting
Overloading	switch node
Axis Distortion	requires scripting
<i>Marks and Properties</i>	
Marks	2D and 3D geometry
Size	size, height, or radius fields
Value	supports gray scale
Texture	images (.png, .gif, and .jpg), sound (.wav, .mp3), and video (.mpg)
Colour	colour node or fields of the material node
Orientation	transform node, translation, and rotation attributes
Transparency	material node field. 1.0 transparent, 0.0 opaque
Connection	can represent graph tree like structures
Enclosure	can represent hierarchies
Text	standard fonts and effects
Advanced Graphics	NURBS, H-Anim, GeoSpatial, no software visualisation component
<i>Temporal Encoding</i>	
Encoding Time	routing event model
Encoding Identity	colour, size, shape, and position
Variation of Retinal Encoding	geometry, material, and coordinate points using routing event model

Table A.2: Evaluating X3D — Form (Part A).

<b>Presentation</b>	
Data Display independence	data can be encoded in X3D visualisations, but is time consuming to create and causes redundant code, better to separate the data from presentation
Referencable Objects	text based DEF and USE fields
Layout Constraints	neither layout constraints or algorithms are supported
Animation	routing event model
Other Modalities	sound (.wav, .mp3) and video (.mpg)
Programming Languages	Java and C++
Program Synchronisation	algorithm animation, haven't done any execution traces
Multiple Views	HTML web frames or within the X3D visualisation
<b>Visualisation Techniques</b>	
Compound Shapes	3D cubes, 3D spirals
IV Metaphors	Information Landscape, Information Cube
SV Techniques	Algorithm animations, UML Diagrams, documentation-related and execution trace visualisations
Pictorial Representations	mainly graph based displays
Model Frameworks	all except metrics centric

Table A.3: Evaluating X3D — Form (Part B).

<b>User Controls</b>	
Graphic Changeability	routing event model and users controls
Input Events	mouse and keyboard
Computation	detects input events by touch, keyboard, and drag sensors and initiates a timer, linked by ROUTE directives
Elision Control	rendering technique, lighting, transparency changing geometry properties
Temporal Control	start, stop exist. Pause, fast-forward, rewind, step, and change of speed require scripting
User Notation	not supported
<b>User Navigation</b>	
Navigation Types	any, walk, examine, fly, look-at, none, slide, and pan
Navigation Control	right click or X3D browser tool bar option
View Refinement	descriptive viewpoints
Speed	navigation and rotation speed by X3D browser option
<b>User Tasks</b>	
Overview	user navigation or specific overview viewpoint
Zoom	user navigation and look-at navigation type
Filter	Boolean filter field, user controls but requires heavy use of scripting
Details-on-demand	level of detail (LOD) or switch nodes, user selection
Relate	not supported, requires scripting and node prototyping
History	not supported, requires scripting and using the SAI
Extract	not supported, requires scripting and using the SAI

Table A.4: Evaluating X3D — Interaction.





# Bibliography

- [1] ALFERT, K., AND ENGELEN, F. Three-dimensional visualization of java class relations. In *Proceedings of the World Conference on Integrated Design & Process Technology (IDPT)* (2000).
- [2] ALFERT, K., AND ENGELEN, F. Experiences in 3-Dimensional visualization of java class relations. *Journal of Design & Process Science* 5, 3 (2001), 91–106.
- [3] ALFERT, K., AND FRONK, A. Manipulation of three-dimensional visualization of java class relations. In *Proceedings of the World Conference on Integrated Design & Process Technology (IDPT)* (2002).
- [4] AMAR, R., EAGAN, J., AND STASKO, J. Low-level components of analytic activity in information visualization. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)* (2005), IEEE Computer Society Press, pp. 111–117.
- [5] AMAR, R., AND STASKO, J. A knowledge task-based framework for design and evaluation of information visualizations. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)* (2004), IEEE Computer Society Press, pp. 143–150.
- [6] AMES, A., NADEAU, D., AND MORELAND, J. *VRML Sourcebook*. John Wiley & Sons, 1996.
- [7] ANDREWS, K. Visualising cyberspace: information visualisation in the harmony internet browser. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)* (1995), IEEE Computer Society Press, pp. 97–104.
- [8] ANDREWS, K. Evaluating information visualisations. In *Proceedings of the AVI Workshop on BEyond time and errors: novel evaLuation methods for Information Visualization (BELIV)* (2006), ACM Press, pp. 1–5.
- [9] ANSLOW, C. XML database support for program trace visualisation. Honours Report, Victoria University of Wellington, March 2003.

- [10] ANSLOW, C., MARSHALL, S., BIDDLE, R., JACKSON, K., AND NOBLE, J. XML database support for program trace visualisation. In *Proceedings of the Australasian Symposium on Information Visualisation (INVIS)* (2004), Australian Computer Society, Inc, pp. 25–34.
- [11] ANSLOW, C., MARSHALL, S., NOBLE, J., AND BIDDLE, R. VET3D: a tool for execution trace web 3D visualization. In *Companion to the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)* (2006), ACM Press, pp. 655–656.
- [12] ANSLOW, C., MARSHALL, S., NOBLE, J., JACKSON, K., MCGAVIN, M., AND BIDDLE, R. Program trace formats for software visualisation. Tech. Rep. CS-TR-06-1, School of Mathematics, Statistics and Computer Science, Computer Science. Victoria University of Wellington, 2006.
- [13] ARENDASH, D. The unreal editor as a Web 3D authoring environment. In *Proceedings of the ACM International Conference on 3D Web Technology (Web3D)* (2004), ACM Press, pp. 119–126.
- [14] ARNAUD, R., AND BARNES, M. *COLLADA: Sailing the Gulf of 3D Digital Content Creation*. AK Peters, Ltd, 2006.
- [15] ARNAUD, R., AND PARISI, T. Developing web applications with COLLADA and X3D. Whitepaper, March 2007. [http://www.khronos.org/collada/presentations/Developing\\_Web\\_Application%27s\\_with\\_COLLADA\\_and\\_X3D.pdf](http://www.khronos.org/collada/presentations/Developing_Web_Application%27s_with_COLLADA_and_X3D.pdf).
- [16] ARTHUR, K. W., BOOTH, K. S., AND WARE, C. Evaluating 3D task performance for fish tank virtual worlds. *ACM Transactions on Information Systems (TOIS)* 11, 3 (1993), 239–265.
- [17] BAECKER, R. Sorting out sorting. 30 minute colour sound videotape, 1981. Presented at ACM SIGGRAPH '81 and excerpted and reprinted in ACM SIGGRAPH Video Review 7, 1983.
- [18] BAECKER, R. *Software Visualization*. MIT Press, 1998, ch. Sorting Out Sorting: A Case Study of Software Visualisation for Teaching Computer Science, pp. 369–381.
- [19] BAECKER, R. Showing instead of telling. In *Proceedings of the International Conference on Computer Documentation (SIGDOC)* (2002), ACM Press, pp. 10–16.

- [20] BALZER, M., AND DEUSSEN, O. Hierarchy based 3D visualization of large software structures. In *Proceedings of the IEEE Conference on Visualization (VIS)* (2004), IEEE Computer Society Press, p. 598.4.
- [21] BALZER, M., NOACK, A., DEUSSEN, O., AND LEWERENTZ, C. Software landscapes: Visualizing the structure of large software systems. In *VisSym 2004, Symposium on Visualization* (2004), Eurographics Association, pp. 261–266.
- [22] BAXTER, G., FREAN, M., NOBLE, J., RICKERBY, M., SMITH, H., VISSER, M., MELTON, H., AND TEMPERO, E. Understanding the shape of java software. In *Proceedings of ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)* (2006), ACM Press, pp. 397–412.
- [23] BEDERSON, B. B., AND SHNEIDERMAN, B., Eds. *The Craft of Information Visualization: Readings and Reflections*. Morgan Kaufmann, 2003.
- [24] BERTIN, J. *Semiology of Graphics*. University of Wisconsin Press, 1983.
- [25] BEYER, D. Co-change visualization. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)* (2005), IEEE Computer Society Press, pp. 89–92.
- [26] BEYER, D. Co-change visualization applied to postgresql and argouml: (msr challenge report). In *Proceedings of the International Workshop on Mining Software Repositories (MSR)* (2006), ACM Press, pp. 165–166.
- [27] BEYER, D., AND NOACK, A. Clustering software artifacts based on frequent common changes. In *Proceedings of the IEEE International Workshop on Program Comprehension (IWPC)* (2005), IEEE Computer Society Press, pp. 259–268.
- [28] BLADH, T., CARR, D., AND SCHOLL, J. Extending tree-maps to three dimensions: A comparative study. In *Proceedings of the Asia-Pacific Conference on Computer-Human Interaction (APCHI)* (2004), Springer Verlag, pp. 50–59.
- [29] BOWMAN, D., KRUIJFF, E., LAVIOLA, J. J., AND POUPYREV, I. An introduction to 3D user interface design. *Presence* 10, 1 (2001), 96–108.
- [30] BOWMAN, D., KRUIJFF, E., LAVIOLA, J. J., AND POUPYREV, I. *3D User Interfaces: Theory and Practice*. Addison Wesley, 2004.
- [31] BRITTLE, J., AND BOLDYREFF, C. Self-organizing maps applied in visualizing large software collections. In *Proceedings of the International Workshop on*

- Visualizing Software for Understanding and Analysis (VISSOFT)* (2003), IEEE Computer Society Press, pp. 104–109.
- [32] BROOKS, F. P. No silver bullet: essence and accidents of software engineering. *IEEE Computer* 20, 4 (1987), 10–19.
- [33] BROWN, M., AND NAJORK, M. *Software Visualization*. MIT Press, 1998, ch. Algorithm Animation Using Interactive 3D Graphics, pp. 119–135.
- [34] BROWN, M. H. *Algorithm Animation*. PhD thesis, Brown University, 1987. MIT Press.
- [35] BROWN, M. H. Zeus: A system for algorithm animation and multi-view editing. In *Proceedings of the IEEE Workshop on Visual Languages (VL)* (1991), IEEE Computer Society Press, pp. 4–9.
- [36] BROWN, M. H., AND HERSHBERGER, J. Color and sound in algorithm animation. *IEEE Computer* 25, 12 (1992), 52–63.
- [37] BROWN, M. H., AND NAJORK, M. A. Algorithm animation using 3D interactive graphics. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)* (1993), ACM Press, pp. 93–100.
- [38] BROWN, M. H., AND SEDGEWICK, R. A system for algorithm animation. In *Proceedings of the ACM Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (1984), ACM Press, pp. 177–186.
- [39] BRUEGGE, B., AND DUTOIT, A. *Object-Oriented Software Engineering, Conquering Complex and Changing Systems*. Prentice Hall, 2000.
- [40] BRUTZMAN, D. *Visualizing Information Using SVG and X3D, XML-based technologies for the XML-based Web*. Springer Verlag, 2005, ch. X3D-Edit Authoring Tool for Extensible 3D (X3D) Graphics, pp. 63–84.
- [41] BRUTZMAN, D., AND DALY, L. *X3D: Extensible 3D Graphics for Web Authors*. Morgan Kaufmann, 2007.
- [42] BRUTZMAN, D., HARNEY, J., AND BLAIS, C. *Visualizing Information Using SVG and X3D, XML-based technologies for the XML-based Web*. Springer Verlag, 2005, ch. X3D Fundamentals, pp. 285–292.
- [43] BURNETT, M., GOLDBERG, A., AND LEWIS, T., Eds. *Visual Object-Oriented Programming: Concepts and Environments*. Manning Publications, 1995.

- [44] CALLAGHAN, M., AND HIRSCHMULLER, H. 3D visualisation of design patterns and Java programs in computer science education. In *Proceedings of the ACM Conference on Integrating Technology into Computer Science Education (ITiCSE)* (1998), ACM Press, pp. 37–40.
- [45] CARD, S. K., MACKINLAY, J. D., AND SHNEIDERMAN, B. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.
- [46] CARPENDALE, M. S. T., COWPERTHWAIT, D. J., AND FRACCHIA, F. D. Extending distortion viewing from 2D to 3D. *IEEE Computer Graphics Applications* 17, 4 (1997), 42–51.
- [47] CASEY, K., AND EXTON, C. A Java 3D implementation of a geon based visualisation tool for UML. In *Proceedings of the International Conference on Principles and Practice of Programming in Java (PPPJ)* (2003), Computer Science Press, Inc., pp. 63–65.
- [48] CHARTERS, S., KNIGHT, C., THOMAS, N., AND MUNRO, M. Visualisation for informed decision making; from code to components. In *Proceedings of the ACM Conference on Software Engineering and Knowledge Engineering (SEKE)* (2002), ACM Press, pp. 765–772.
- [49] CHARTERS, S., THOMAS, N., AND MUNRO, M. The end of the line for software visualisation? In *Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)* (2003), IEEE Computer Society Press, pp. 110–112.
- [50] CHEN, C. *Information Visualization: Beyond The Horizon*. Springer Verlag, 2006.
- [51] CHI, E. H. A taxonomy of visualization techniques using the data state reference model. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)* (2000), IEEE Computer Society Press, pp. 69–76.
- [52] CHITTARO, L., AND RANON, R. Web3D technologies in learning, education and training: motivations, issues, opportunities. *Computers & Education* 49, 1 (2007), 3–18.
- [53] CHURCHER, N., AND CREEK, A. Building virtual worlds with the big-bang model. In *Proceedings of the Australasian Symposium on Information Visualisation (INVIS)* (2001), Australian Computer Society, Inc, pp. 87–94.

- [54] CHURCHER, N., AND IRWIN, W. Informing the design of pipeline-based software visualisations. In *Proceedings of the Asia-Pacific Symposium on Information Visualisation (APVIS)* (2005), Australian Computer Society, Inc, pp. 59–68.
- [55] CHURCHER, N., IRWIN, W., AND COOK, C. Inhomogeneous force-directed layout algorithms in the visualisation pipeline: from layouts to visualisations. In *Proceedings of the Australasian Symposium on Information Visualisation (INVIS)* (2004), Australian Computer Society, Inc, pp. 43–51.
- [56] CHURCHER, N., IRWIN, W., AND KRIZ, R. Visualising class cohesion with virtual worlds. In *Proceedings of the Asia-Pacific Symposium on Information Visualisation (APVIS)* (2003), Australian Computer Society, Inc, pp. 89–97.
- [57] CHURCHER, N., KEOWN, L., AND IRWIN, W. Virtual worlds for software visualisation. In *Proceedings of the Workshop on Software Visualisation Workshop (SoftVis)* (1999), pp. 9–16.
- [58] COCKBURN, A. Revisiting 2D vs 3D implications on spatial memory. In *Proceedings of the Australasian Conference on User Interfaces (AUIC)* (2004), Australian Computer Society, Inc, pp. 25–31.
- [59] COCKBURN, A., AND MCKENZIE, B. An evaluation of cone trees. In *People and Computers XIV: British Computer Society Conference on Human Computer Interaction* (2000), Springer Verlag, pp. 425–436.
- [60] COCKBURN, A., AND MCKENZIE, B. 3D or not 3D?: evaluating the effect of the third dimension in a document management system. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)* (2001), ACM Press, pp. 434–441.
- [61] COCKBURN, A., AND MCKENZIE, B. Evaluating the effectiveness of spatial memory in 2D and 3D physical and virtual environments. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)* (2002), ACM Press, pp. 203–210.
- [62] COCKBURN, A., AND MCKENZIE, B. Evaluating spatial memory in two and three dimensions. *International Journal of Human Computer Studies* 61, 3 (2004), 359–373.
- [63] COX, K. C., AND ROMAN, G.-C. Abstraction in algorithm animation. In *Proceedings of the IEEE Workshop on Visual Languages (VL)* (1992), IEEE Computer Society Press, pp. 18–24.

- [64] CRAFT, B., AND CAIRNS, P. Beyond guidelines: What can we learn from the visual information seeking mantra? In *Proceedings of the IEEE International Conference on Information Visualisation (IV)* (2005), IEEE Computer Society Press, pp. 110–118.
- [65] DANN, W., COOPER, S., AND PAUSCH, R. Using visualization to teach novices recursion. *ACM SIGCSE Bulletin* 33, 3 (2001), 109–112.
- [66] DAVIS, T., AND KENNETH. Kscope: A modularized tool for 3D visualization of object-oriented programs. In *Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)* (2003), IEEE Computer Society Press, pp. 98–103.
- [67] DIEHL, S. Vrml++: A language for object-oriented virtual-reality models. In *Proceedings of the IEEE International Conference on Technology of Object-Oriented Languages and Systems (TOOLS)* (1997), IEEE Computer Society Press, pp. 141–150.
- [68] DIEHL, S., Ed. *Revised Lectures on Software Visualization, International Seminar*. Springer Verlag, 2002.
- [69] DIEHL, S. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer Verlag, 2007.
- [70] DIGIANO, C. J., AND BAECKER, R. M. Program auralization: sound enhancements to the programming environment. In *Proceedings of the Conference on Graphics Interface (GI)* (1992), Morgan Kaufmann, pp. 44–52.
- [71] DREDGE, S. *Web 3D : New Perspectives*. Universe, 2002.
- [72] DREW, N. S., AND HENDLEY, R. J. Visualisation of complex systems. Tech. rep., University of Birmingham, 1995.
- [73] DREW, N. S., AND HENDLEY, R. J. Visualising complex interacting systems. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)* (1995), ACM Press, pp. 204–205.
- [74] DUIGNAN, M. Evaluating scalable vector graphics for software visualisation. Master's thesis, Victoria University of Wellington, 2003.
- [75] DUIGNAN, M., BIDDLE, R., AND TEMPERO, E. Evaluating scalable vector graphics for use in software visualisation. In *Proceedings of the Australasian Symposium on Information Visualisation (INVIS)* (2003), Australian Computer Society, Inc, pp. 127–136.

- [76] DWYER, T. Three dimensional UML using force directed layout. In *Proceedings of the Australasian Symposium on Information Visualisation (INVIS)* (2001), Australian Computer Society, Inc, pp. 77–85.
- [77] DWYER, T., AND ECKERSLEY, P. *Graph Drawing Software*. Springer Verlag, 2003, ch. WilmaScope - a 3D graph visualisation system, pp. 55–75.
- [78] EADES, P., AND ZHANG, K. *Software Visualisation*, vol. 7 of *Software Engineering and Knowledge Engineering*. World Scientific, 1996.
- [79] EICK, S. G., STEFFEN, J. L., AND ERIC E. SUMNER, J. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering* 18, 11 (1992), 957–968.
- [80] ELLERSHAW, S., AND OUDSHOORN, M. Program visualization - the state of the art. Tech. Rep. 94-19, Department of Computer Science, University of Adelaide, 1994.
- [81] ELLIS, G., AND DIX, A. An explorative analysis of user evaluation studies in information visualisation. In *Proceedings of the AVI Workshop on BEyond time and errors: novel evaluation methods for Information Visualization (BELIV)* (2006), ACM Press, pp. 1–7.
- [82] FEIJS, L., AND JONG, R. D. 3D visualization of software architectures. *Communications of the ACM* 41, 12 (1998), 73–78.
- [83] FOWLER, M., AND SCOTT, K. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 2nd ed. Addison Wesley, 2000.
- [84] FRANCK, G., SARDESAI, M., AND WARE, C. Layout and structuring object oriented software in three dimensions. In *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)* (1995), IBM Press, pp. 22–31.
- [85] FRONK, A. Evaluating 3D-visualisation of code structures in the context of reverse engineering. In *Proceedings of the Workshop on Empirical Studies in Reverse Engineering (WESRE)* (2006), IEEE Computer Society Press.
- [86] FRONK, A., BRUCKHOFF, A., AND KERN, M. 3D visualisation of code structures in Java software systems. In *Proceedings of the ACM Symposium on Software Visualization (SOFTVIS)* (2006), ACM Press, pp. 145–146.
- [87] FURNAS, G. W. Generalized fisheye views. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)* (1986), ACM Press, pp. 16–23.



- [88] GARLAND, K. *Mr. Beck's Underground Map*. Capital Transport Publishing, 1994.
- [89] GEROIMENKO, V., AND CHEN, C., Eds. *Visualizing Information Using SVG and X3D, XML-based technologies for the XML-based Web*. Springer Verlag, 2005.
- [90] GIL, J., AND KENT, S. Three dimensional software modelling. In *Proceedings of the International Conference on Software Engineering (ICSE)* (1998), IEEE Computer Society Press, pp. 105–114.
- [91] GLOOR, P. A. *Software Visualization*. MIT Press, 1998, ch. User Interface Issues for Algorithm Animation, pp. 145–152.
- [92] GOGOLLA, M., RADFELDER, O., AND RICHTERS, M. Towards three-dimensional animation of UML diagrams. In *Proceedings of the International Conference on Unified Modeling Language (UML)* (1999), Springer Verlag, pp. 489–502.
- [93] GOLDMAN, D., ECKERT, R., AND COHEN, M. Three-dimensional computation visualization for computer graphics rendering algorithms. In *Proceedings of the ACM Symposium on Computer Science Education (SIGCSE)* (1996), ACM Press, pp. 358–362.
- [94] GORDON, D., BIDDLE, R., NOBLE, J., AND TEMPERO, E. A technology for lightweight web-based visual applications. In *Proceedings of the IEEE Symposium on Human Centric Computing Languages and Environments (HCC)* (2003), IEEE Computer Society Press, pp. 245–247.
- [95] GRACANIN, D., MATKOVIC, K., AND ELTOWEISSY, M. Software visualization. *Innovations in Systems and Software Engineering, A NASA Journal* 1, 2 (2005), 221–230.
- [96] GRAHAM, H., YANG, H. Y., AND BERRIGAN, R. A solar system metaphor for 3D visualisation of object oriented software metrics. In *Proceedings of the Australasian Symposium on Information Visualisation (INVIS)* (2004), Australian Computer Society, Inc., pp. 53–59.
- [97] GREEVY, O., LANZA, M., AND WYSSEIER, C. Visualizing feature interaction in 3-D. In *Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)* (2005), IEEE Computer Society Press, pp. 114–119.

- [98] GREEVY, O., LANZA, M., AND WYSSEIER, C. Visualizing live software systems in 3D. In *Proceedings of the ACM Symposium on Software Visualization (SOFTVIS)* (2006), ACM Press, pp. 47–56.
- [99] HALLER, M., BILLINGHURST, M., AND THOMAS, B., Eds. *Emerging Technologies of Augmented Reality: Interfaces and Design*. IGI Global, 2006.
- [100] HANSEN, R. K. Why has Web3D not broken through yet? Honours Report, Cumbria Institute of Arts, February 2005.
- [101] HARRIS, R. L. *Information graphics a comprehensive illustrated reference: visual tools for analyzing, managing, and communicating*. Oxford University Press, 1999.
- [102] HARTLEY, D., CHURCHER, N., AND ALBERTSON, G. Virtual worlds for web site visualisation. In *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC)* (2000), IEEE Computer Society Press, pp. 448–455.
- [103] HARTMAN, J., AND WERNECKE, J. *The VRML 2.0 Handbook*. Addison Wesley, 1996.
- [104] HATCH, A., SMITH, M., TAYLOR, C., AND MUNRO, M. No silver bullet for software visualisation evaluation. In *Proceedings of The International Conference on Imaging Science, Systems, and Technology (CISST)* (2001).
- [105] HENDLEY, R. J., DREW, N. S., WOOD, A. M., AND BEALE, R. Case study: Narcissus: visualising information. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)* (1995), IEEE Computer Society Press, pp. 90–96.
- [106] HICKS, M., O'MALLEY, C., NICHOLS, S., AND ANDERSON, B. Comparison of 2D and 3D representations for visualising telecommunication usage. *Behaviour & Information Technology* 22, 3 (2003), 185–201.
- [107] HOLMBERG, N., WUENSCHÉ, B., AND TEMPERO, E. A framework for interactive web-based visualization. In *Proceedings of the Australasian Conference on User Interfaces (AUIC)* (2006), CRPIT, Australian Computer Society, Inc, pp. 137–144.
- [108] HOPKINS, J., AND FISHWICK, P. The rube framework for personalized 3-D software visualization. In *Revised Lectures on Software Visualization, International Seminar* (2002), Springer Verlag, pp. 368–380.

- [109] HUBONA, G. S., SHIRAH, G. W., AND FOUT, D. G. 3D object recognition with motion. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)* (1997), ACM Press, pp. 345–346.
- [110] IRANI, P., TINGLEY, M., AND WARE, C. Using perceptual syntax to enhance semantic content in diagrams. *IEEE Computer Graphics Applications* 21, 5 (2001), 76–85.
- [111] IRANI, P., AND WARE, C. Diagrams based on structural object perception. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI)* (2000), ACM Press, pp. 61–67.
- [112] IRANI, P., AND WARE, C. Diagramming information structures using 3D perceptual primitives. *ACM Transactions on Computer-Human Interaction (TOCHI)* 10, 1 (2003), 1–19.
- [113] IRWIN, W., AND CHURCHER, N. XML in the visualisation pipeline. In *Proceedings of the Pan-Sydney Area Workshop on Visual Information Processing (VIP)* (2001), Australian Computer Society, Inc, pp. 59–67.
- [114] IRWIN, W., AND CHURCHER, N. Object oriented metrics: Precision tools and configurable visualisations. In *Proceedings of the International Symposium on Software Metrics (METRICS)* (2003), IEEE Computer Society Press, pp. 112–123.
- [115] JEFFERY, C. L. *Program Monitoring and Visualization: An Exploratory Approach*. Springer Verlag, 1999.
- [116] JOHNSON, B., AND SHNEIDERMAN, B. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the IEEE Conference on Visualization (VIS)* (1991), IEEE Computer Society Press, pp. 284–291.
- [117] KADMON, N., AND SHLOMI, E. A polyfocal projection for statistical surfaces. *Cartograph* 15, 1 (1978), 36–41.
- [118] KEOWN, L. Virtual 3D worlds for enhanced software visualization. Master’s thesis, University of Canterbury, 2000.
- [119] KERREN, A., AND STASKO, J. T. Algorithm animation - introduction. In *Revised Lectures on Software Visualization, International Seminar* (2002), Springer Verlag, pp. 1–15.

- [120] KHALED, R., MACKAY, D., BIDDLE, R., AND NOBLE, J. A lightweight web-based case tool for sequence diagrams. In *Proceedings of the ACM SIGCHI New Zealand Conference on Computer-Human Interaction (CHINZ)* (2002), pp. 55–60.
- [121] KIM, T., AND FISHWICK, P. A. A 3D XML-based customized framework for dynamic models. In *Proceedings of the ACM International Conference on 3D Web Technology (Web3D)* (2002), ACM Press, pp. 103–109.
- [122] KLEIBERG, E., VAN DE WETERING, H., AND WIJK, J. J. V. Botanical visualization of huge hierarchies. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)* (2001), IEEE Computer Society Press, pp. 87–94.
- [123] KNIGHT, C., AND MUNRO, M. Using an existing game engine to facilitate multi-user software visualisation. Tech. Rep. 8/98, University of Durham, 1998.
- [124] KNIGHT, C., AND MUNRO, M. Comprehension with[in] virtual environment visualisations. In *Proceedings of the IEEE International Workshop on Program Comprehension (IWPC)* (1999), IEEE Computer Society Press, pp. 4–11.
- [125] KNIGHT, C., AND MUNRO, M. Virtual but visible software. In *Proceedings of the IEEE International Conference on Information Visualisation (IV)* (2000), pp. 198–205.
- [126] KNIGHT, C., AND MUNRO, M. Visualising Java uncertainty. In *Java/Jini Technologies, in the Multimedia Networks and Management Program of The Convergence of Information Technologies and Communication (ITCOM)* (2001), vol. 4521, SPIE.
- [127] KNIGHT, C., AND MUNRO, M. Program comprehension experiences with GXL; comprehension for comprehension. In *Proceedings of the IEEE International Workshop on Program Comprehension (IWPC)* (2002), IEEE Computer Society Press, pp. 147–156.
- [128] KOIKE, H. An application of three-dimensional visualization to object-oriented programming. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI)* (1992), World Scientific, pp. 180–192.
- [129] KOIKE, H. Three-dimensional software visualization: A framework and its applications. In *Proceedings of the International Conference of the Computer Graphics Society on Visual Computing* (1992), Springer Verlag, pp. 151–170.

- [130] KOIKE, H. The role of another spatial dimension in software visualization. *ACM Transactions on Information Systems (TOIS)* 11, 3 (1993), 266–286.
- [131] KOIKE, H., AND CHU, H.-C. How does 3-D visualization work in software engineering?: empirical study of a 3-D version/module visualization system. In *Proceedings of the International Conference on Software Engineering (ICSE)* (1998), IEEE Computer Society Press, pp. 516–519.
- [132] KOSARA, R., HEALEY, C. G., INTERRANTE, V., LAIDLAW, D. H., AND WARE, C. User studies: Why, how, and when? *IEEE Computer* 23, 4 (2003), 20–25.
- [133] KOSCHKE, R. Software visualization for reverse engineering. In *Revised Lectures on Software Visualization, International Seminar* (2002), Springer Verlag, pp. 138–150.
- [134] KOSCHKE, R. Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. *Journal of Software Maintenance* 15, 2 (2003), 87–109.
- [135] KOT, B., WUENSCHÉ, B., GRUNDY, J., AND HOSKING, J. Information visualisation utilising 3D computer game engines - case study: A source code comprehension tool. In *Proceedings of the ACM SIGCHI New Zealand Conference on Computer-Human Interaction (CHINZ)* (2005), ACM Press, pp. 53–60.
- [136] LAMPING, J., RAO, R., AND PIROLI, P. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)* (1995), ACM Press, pp. 401–408.
- [137] LANGELIER, G., SAHRAOUI, H., AND POULIN, P. Visualization-based analysis of quality for large-scale software systems. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2005), ACM Press, pp. 214–223.
- [138] LANGELIER, G., SAHRAOUI, H., AND POULIN, P. Animation coherence in representing software evolution. In *Proceedings of the ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE)* (2006), pp. 41–50.
- [139] LANZA, M. Codecrawler - lessons learned in building a software visualization tool. In *Proceedings of the European Conference on Software Maintenance and Reengineering (CSMR)* (2003), IEEE Computer Society Press, pp. 409–418.

- [140] LANZA, M., AND DUCASSE, S. *Tools for Software Maintenance and Reengineering*. RCost Software Technology Series, 2005, ch. CodeCrawler - An Extensible and Language Independent 2D and 3D Software Visualization Tool, pp. 74–94.
- [141] LANZA, M., MARINESCU, R., AND DUCASSE, S. *Object-Oriented Metrics in Practice*. Springer Verlag, 2005.
- [142] LEE, B., PLAISANT, C., PARR, C. S., FEKETE, J.-D., AND HENRY, N. Task taxonomy for graph visualization. In *Proceedings of the AVI Workshop on BEyond time and errors: novel evaLuation methods for Information Visualization (BELIV)* (2006), ACM Press, pp. 1–5.
- [143] LEUNG, Y. K., AND APPERLEY, M. D. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction (TOCHI)* 1, 2 (1994), 126–160.
- [144] LEWERENTZ, C., AND NOACK, A. *Graph Drawing Software*. Springer Verlag, 2003, ch. CrocoCosmos - 3D Visualization of Large Object-Oriented Programs, pp. 279–297.
- [145] LEWERENTZ, C., AND SIMON, F. Metrics-based 3D visualization of large object-oriented programs. In *Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)* (2002), IEEE Computer Society Press, pp. 70–77.
- [146] LIEBERMAN, H. A three-dimensional representation for program execution. In *Proceedings of the IEEE Workshop on Visual Languages (VL)* (1989), IEEE Computer Society Press, pp. 111–116.
- [147] LOWE, W., ERICSSON, M., LUNDBERG, J., PANAS, T., AND PETTERSSON, N. Vizzanalyzer - a software comprehension framework. In *Software Engineering Research and Practice (SERPS)* (2003).
- [148] MACKAY, D., NOBLE, J., AND BIDDLE, R. A lightweight web-based case tool for UML class diagrams. In *Proceedings of the Australasian Conference on User Interfaces (AUIC)* (2003), Australian Computer Society, Inc, pp. 95–98.
- [149] MACKINLAY, J. D., ROBERTSON, G. G., AND CARD, S. K. The perspective wall: detail and context smoothly integrated. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)* (1991), ACM Press, pp. 173–179.

- [150] MALETIC, J., LEIGH, J., AND MARCUS, A. Visualizing software in an immersive virtual reality environment. In *Proceedings of the ICSE Workshop on Software Visualization* (2001), IEEE Computer Society Press, pp. 49–54.
- [151] MALETIC, J., LEIGH, J., MARCUS, A., AND DUNLAP, G. Visualizing object-oriented software in virtual reality. In *Proceedings of the IEEE International Workshop on Program Comprehension (IWPC)* (2001), IEEE Computer Society Press, pp. 26–35.
- [152] MALETIC, J., AND MARCUS, A. CFB: a call for benchmarks - for software visualization. In *Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)* (2003), IEEE Computer Society Press, pp. 113–116.
- [153] MALETIC, J., MARCUS, A., AND COLLARD, M. A task oriented view of software visualization. In *Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)* (2002), IEEE Computer Society Press, pp. 32–40.
- [154] MARCUS, A., COMORSKI, D., AND SERGEYEV, A. Supporting the evolution of a software visualization tool through usability studies. In *Proceedings of the IEEE International Workshop on Program Comprehension (IWPC)* (2005), IEEE Computer Society Press, pp. 307–316.
- [155] MARCUS, A., FENG, L., AND MALETIC, J. I. 3D representations for software visualization. In *Proceedings of the ACM Symposium on Software Visualization (SOFTVIS)* (2003), ACM Press, pp. 27–36.
- [156] MARCUS, A., FENG, L., AND MALETIC, J. I. Comprehension of software analysis data using 3D visualization. In *Proceedings of the IEEE International Workshop on Program Comprehension (IWPC)* (2003), IEEE Computer Society Press, pp. 105–114.
- [157] MARSHALL, S. *Test Driving Reusable Components*. PhD thesis, Victoria University of Wellington, 2006.
- [158] MARSHALL, S., BIDDLE, R., AND NOBLE, J. Using software visualisation to enhance online component markets. In *Proceedings of the Australasian Symposium on Information Visualisation (INVIS)* (2004), Australian Computer Society, Inc, pp. 35–41.
- [159] MARSHALL, S., JACKSON, K., ANSLOW, C., AND BIDDLE, R. Aspects to visualising reusable components. In *Proceedings of the Australasian Symposium*

- on Information Visualisation (INVIS)* (2003), Australian Computer Society, Inc, pp. 81–88.
- [160] MARSHALL, S., JACKSON, K., BIDDLE, R., MCGAVIN, M., TEMPERO, E., AND DUIGNAN, M. Visualising reusable software over the web. In *Proceedings of the Australasian Symposium on Information Visualisation (INVIS)* (2001), Australian Computer Society, Inc, pp. 103–111.
- [161] MCGAVIN, M. Extracting software reuse information for visualisation tools. Honours Report, Victoria University of Wellington, October 2001.
- [162] MCGAVIN, M., WRIGHT, T., AND MARSHALL, S. Visualisations of execution traces (VET): An interactive plugin-based visualisation tool. In *Proceedings of the Australasian Conference on User Interfaces (AUIC)* (2006), Australian Computer Society, Inc, pp. 153–160.
- [163] MCINTOSH, P., HAMILTON, M., AND VAN SCHYNDEL, R. X3D-UML: enabling advanced UML visualisation through X3D. In *Proceedings of the ACM International Conference on 3D Web Technology (Web3D)* (2005), ACM Press, pp. 135–142.
- [164] MESNAGE, C., AND LANZA, M. White coats: Web-visualization of evolving software in 3D. In *Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)* (2005), IEEE Computer Society Press, pp. 40–45.
- [165] MICROSYSTEMS, S. Java debugger interface API, 2003. <http://java.sun.com/j2se/1.4.2/docs/guide/jpda/jdi>.
- [166] MUNZNER, T. H3: laying out large directed graphs in 3D hyperbolic space. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)* (1997), IEEE Computer Society Press, pp. 2–10.
- [167] MUNZNER, T., AND BURCHARD, P. Visualizing the structure of the world wide web in 3D hyperbolic space. In *Proceedings of the Symposium on Virtual Reality Modeling Language (VRML)* (1995), ACM Press, pp. 33–38.
- [168] NAJORK, M. Web-based algorithm animation. In *Proceedings of the Conference on Design Automation (DAC)* (2001), ACM Press, pp. 506–511.
- [169] NAJORK, M. A., AND BROWN, M. H. Three-dimensional web-based algorithm animations. Tech. Rep. SRC-RR-170, Compaq Systems Research Centre, 2001.



- [170] NESBITT, K. V. Using guidelines to assist in the visualisation design process. In *Proceedings of the Asia-Pacific Symposium on Information Visualisation (APVIS)* (2005), Australian Computer Society, Inc, pp. 115–123.
- [171] NIELSEN, J. *Usability Engineering*. Morgan Kaufmann, 1994.
- [172] NIELSEN, J. *Designing Web Usability : The Practice of Simplicity*. New Riders, 2000.
- [173] NIELSEN, J., AND LORANGER, H. *Prioritizing Web Usability*. New Riders, 2006.
- [174] NOBLE, J. *Abstract Program Visualisation*. PhD thesis, Victoria University of Wellington, 1996.
- [175] NOBLE, J., AND GROVES, L. Tarraingím - a program animation environment. In *Proceedings of the New Zealand Computer Science Conference* (1991).
- [176] NOBLE, J., GROVES, L., AND BIDDLE, R. Object oriented program visualisation in tarraingím. *Australian Computing Journal* 27, 4 (1995).
- [177] OUDSHOORN, M., WIDJAJA, H., AND ELLERSHAW, S. Aspects and taxonomy of program visualisation. In *Software Visualisation*. World Scientific, 1996, pp. 3–26.
- [178] PANAS, T., BERRIGAN, R., AND GRUNDY, J. A 3D metaphor for software production visualization. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)* (2003), IEEE Computer Society Press, pp. 314–319.
- [179] PANAS, T., LINCKE, R., AND LOWE, W. Online-configuration of software visualizations with Vizz3D. In *Proceedings of the ACM Symposium on Software Visualization (SOFTVIS)* (2005), ACM Press, pp. 173–182.
- [180] PARISI, T. Flux: lightweight, standards-based web graphics in XML. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Web Graphics* (2003), ACM Press, pp. 1–1.
- [181] PARKER, G., FRANCK, G., AND WARE, C. Visualization of large nested graphs in 3D: navigation and interaction. *Journal of Visual Languages and Computing* 9, 3 (1998), 299–317.
- [182] PAUW, W. D., HELM, R., KIMELMAN, D., AND VLISSIDES, J. Visualizing the behavior of object-oriented systems. In *Proceedings of ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)* (1993), ACM Press, pp. 326–337.

- [183] PAUW, W. D., JENSEN, E., MITCHELL, N., SEVITSKY, G., VLISSIDES, J., AND YANG, J. Visualizing the execution of Java programs. In *Revised Lectures on Software Visualization, International Seminar* (2002), Springer Verlag, pp. 151–162.
- [184] PAUW, W. D., KIMELMAN, D., AND VLISSIDES, J. Modeling object-oriented program execution. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP)* (1994), Springer Verlag, pp. 163–182.
- [185] PAUW, W. D., KIMELMAN, D., AND VLISSIDES, J. *Software Visualization*. MIT Press, 1998, ch. Visualizing Object-Oriented Software Execution, pp. 329–346.
- [186] PAUW, W. D., MITCHELL, N., ROBILLARD, M., SEVITSKY, G., AND SRINIVASAN, H. Drive-by analysis of running programs. In *Proceedings of the ICSE Workshop on Software Visualization* (2001), pp. 27–32.
- [187] PAUW, W. D., AND SEVITSKI, G. Visualizing reference patterns for solving memory leaks in Java. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP)* (1999), Springer Verlag, pp. 116–134.
- [188] PETRE, M., BLACKWELL, A., AND GREEN, T. *Software Visualization*. MIT Press, 1998, ch. Cognitive Questions in Software Visualization, pp. 453–480.
- [189] PETRE, M., AND DE QUINCEY, E. A gentle overview of software visualisation. Psychology of Programming Interest Group (PPIG) Newsletter, September 2006. <http://www.ppig.org/newsletters/2006-09.html>.
- [190] PLAISANT, C. The challenge of information visualization evaluation. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI)* (2004), ACM Press, pp. 109–116.
- [191] POLYS, N. F. Stylesheet transformations for interactive visualization: towards a Web3D chemistry curricula. In *Proceedings of the ACM International Conference on 3D Web Technology (Web3D)* (2003), ACM Press, pp. 85–90.
- [192] POLYS, N. F. *Visualizing Information Using SVG and X3D, XML-based technologies for the XML-based Web*. Springer Verlag, 2005, ch. Publishing Paradigms for X3D, pp. 153–180.
- [193] POULIN, J. S. *Measuring Software Reuse: principles, practices, and economic models*. Addison Wesley, 1997.
- [194] PRICE, B., BAECKER, R., AND SMALL, I. *Software Visualization*. MIT Press, 1998, ch. An Introduction to Software Visualization, pp. 3–27.

- [195] PRICE, B. A., BAECKER, R. M., AND SMALL, I. S. A principled taxonomy of software visualization. *Journal of Visual Languages and Computing* 4, 2 (1993), 211–266.
- [196] PRICE, B. A., SMALL, I. S., AND BAECKER, R. M. A taxonomy of software visualisation. In *Proceedings of the International Conference on System Sciences* (1992), pp. 597–606.
- [197] RADFELDER, O., AND GOGOLLA, M. On better understanding UML diagrams through interactive three-dimensional visualization and animation. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI)* (2000), ACM Press, pp. 292–295.
- [198] RAO, R., AND CARD, S. K. The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)* (1994), ACM Press, pp. 318–322.
- [199] REISS, S. P. A framework for abstract 3D visualization. In *Proceedings of the IEEE Symposium on Visual Languages (VL)* (1993), IEEE Computer Society Press, pp. 108–115.
- [200] REISS, S. P. 3-D visualization of program information. In *Proceedings of Graph Drawing* (1994), Springer Verlag, pp. 12–24.
- [201] REISS, S. P. An engine for the 3D visualization of program information. *Journal of Visual Languages and Computing* 6, 3 (1995), 299–323.
- [202] REISS, S. P. Bee/Hive: A software visualization back end. In *Proceedings of the ICSE Workshop on Software Visualization* (2001), IEEE Computer Society Press, pp. 44–48.
- [203] REISS, S. P. An overview of BLOOM. In *Proceedings of the ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE)* (2001), ACM Press, pp. 2–5.
- [204] REISS, S. P. A visual query language for software visualization. In *Proceedings of the IEEE Symposium on Human Centric Computing Languages and Environments (HCC)* (2002), IEEE Computer Society Press, pp. 80–82.
- [205] REISS, S. P. Event-based performance analysis. In *Proceedings of the IEEE International Workshop on Program Comprehension (IWPC)* (2003), IEEE Computer Society Press, pp. 74–83.

- [206] REISS, S. P. Jive: visualizing Java in action demonstration description. In *Proceedings of the International Conference on Software Engineering (ICSE)* (2003), IEEE Computer Society Press, pp. 820–821.
- [207] REISS, S. P. Visualizing Java in action. In *Proceedings of the ACM Symposium on Software Visualization (SOFTVIS)* (2003), ACM Press, pp. 57–65.
- [208] REISS, S. P. The paradox of software visualization. In *Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)* (2005), pp. 59–63.
- [209] REISS, S. P., AND RENIERIS, M. Generating Java trace data. In *Proceedings of the ACM Conference on Java Grande* (2000), ACM Press, pp. 71–77.
- [210] REISS, S. P., AND RENIERIS, M. Languages for dynamic instrumentation. In *Proceedings of the ICSE Workshop on Dynamic Analysis (WODA)* (2003), pp. 41–45.
- [211] REISS, S. P., AND RENIERIS, M. *Software Visualization: From Theory to Practice*. Kluwer Academic Publishers, 2003, ch. The Bloom Software Visualization System, pp. 311–357.
- [212] REISS, S. P., AND RENIERIS, M. Jove: Java as it happens. In *Proceedings of the ACM Symposium on Software Visualization (SOFTVIS)* (2005), ACM Press, pp. 115–124.
- [213] REKIMOTO, J., AND GREEN, M. The information cube: Using transparency in 3D information visualization. In *Proceedings of the Workshop on Information Technologies & Systems (WITS)* (1993), pp. 125–132.
- [214] RHEINGOLD, H. *Virtual Reality*. Summit Books, 1991.
- [215] RILLING, J., AND MUDUR, S. P. On the use of metaballs to visually map source code structures and analysis results onto 3D space. In *Proceedings of the Working Conference on Reverse Engineering (WCRE)* (2002), IEEE Computer Society Press, pp. 299–308.
- [216] RILLING, J., AND MUDUR, S. P. 3D visualization techniques to support slicing-based program comprehension. *Computers & Graphics* 29, 3 (2005), 311–329.
- [217] RILLING, J., WANG, J., AND MUDUR, S. Metaviz - issues in software visualizing beyond 3D. In *Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)* (2003), IEEE Computer Society Press, pp. 92–97.

- [218] RISDEN, K., CZERWINSKI, M. P., MUNZNER, T., AND COOK, D. An initial examination of ease of use for 2D and 3D information visualizations of web content. *International Journal of Human Computer Studies* 53, 5 (2000), 695–714.
- [219] ROBERTSON, G. G., CZERWINSKI, M., LARSON, K., ROBBINS, D. C., THIEL, D., AND VAN DANTZICH, M. Data mountain: Using spatial memory for document management. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)* (1998), ACM Press, pp. 153–162.
- [220] ROBERTSON, G. G., MACKINLAY, J. D., AND CARD, S. K. Cone trees: animated 3D visualizations of hierarchical information. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)* (1991), ACM Press, pp. 189–194.
- [221] ROMAN, G.-C., AND COX, K. C. A taxonomy of program visualization systems. *IEEE Computer* 26, 12 (1993), 11–24.
- [222] ROMAN, G.-C., COX, K. C., WILCOX, D., AND PLUN, J. Pavane: A system for declarative visualization of concurrent computations. *Journal of Visual Languages and Computing* 3, 2 (1992), 161–193.
- [223] RYMASZEWSKI, M., AU, W. J., WALLACE, M., WINTERS, C., ONDREJKA, C., AND BATSTONE-CUNNINGHAM, B. *Second Life: The Official Guide*. Sybex, 2006.
- [224] SARKAR, M., AND BROWN, M. H. Graphical fisheye views of graphs. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)* (1992), ACM Press, pp. 83–91.
- [225] SELMAN, D. *Java 3D Programming*. Manning Publications, 2002.
- [226] SEVITSKY, G., PAUW, W. D., AND KONURU, R. An information exploration tool for performance analysis of Java programs. In *Proceedings of the IEEE International Conference on Technology of Object-Oriented Languages and Systems (TOOLS)* (March 2001), pp. 85–101.
- [227] SHNEIDERMAN, B. Tree visualization with tree-maps: 2-D space-filling approach. *ACM Transactions on Graphics (TOG)* 11, 1 (1992), 92–99.
- [228] SHNEIDERMAN, B. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the IEEE Symposium on Visual Languages (VL)* (1996), pp. 336–343.

- [229] SHNEIDERMAN, B. Why not make interfaces better than 3D reality? *IEEE Computer Graphics Applications* 23, 6 (2003), 12–15.
- [230] SHNEIDERMAN, B., AND PLAISANT, C. Strategies for evaluating information visualization tools: multi-dimensional in-depth long-term case studies. In *Proceedings of the AVI Workshop on BEyond time and errors: novel evaluation methods for Information Visualization (BELIV)* (2006), ACM Press, pp. 1–7.
- [231] SPENCE, R. *Information Visualization*. Addison Wesley, 2001.
- [232] SPENCE, R. *Information Visualization: Design For Interaction*. Addison Wesley, 2007.
- [233] SPENCE, R., AND APPERLEY, M. Data base navigation: An office environment for the professional. *Behaviour in Information Technology* 1, 1 (1982), 43–54.
- [234] STASKO, J. The path-transition paradigm: A practical methodology for adding animation to program interfaces. *Journal of Visual Languages and Computing* 1, 3 (1990), 213–236.
- [235] STASKO, J. Tango: A framework and system for algorithm animation. *IEEE Computer* 23, 9 (1990), 27–39.
- [236] STASKO, J. *Software Visualization*. MIT Press, 1998, ch. Smooth, Continuous Animations for Portraying Algorithms and Processes, pp. 103–118.
- [237] STASKO, J., BROWN, M., AND PRICE, B., Eds. *Software Visualization: Programming as a Multimedia Experience*. MIT Press, 1998.
- [238] STASKO, J., AND KRAEMER, E. A methodology for building application-specific visualizations of parallel programs. *Journal of Parallel and Distributed Computing* 18, 2 (1993), 258–264.
- [239] STASKO, J. T. *Tango: A Framework and System for Algorithm Animation*. PhD thesis, Brown University, May 1989.
- [240] STASKO, J. T., AND WEHRLI, J. F. Three-dimensional computation visualization. In *Proceedings of the IEEE Symposium on Visual Languages (VL)* (1993), IEEE Computer Society Press, pp. 100–107.
- [241] STOECKLE, H., GRUNDY, J., AND HOSKING, J. Approaches to supporting software visual notation exchange. In *Proceedings of the IEEE Symposium on Human Centric Computing Languages and Environments (HCC)* (2003), IEEE Computer Society Press, pp. 59–66.

- [242] STORER, T., AND DUNCAN, I. 3D animation of Java program execution for teaching object oriented concepts. In *Proceedings of the International Conference on Visualisation, Imaging and Image Processing (VIIP)* (2007), ACTA Press, pp. 76–81.
- [243] STOREY, M.-A. D., CUBRANIC, D., AND GERMAN, D. M. On the use of visualization to support awareness of human activities in software development: a survey and a framework. In *Proceedings of the ACM Symposium on Software Visualization (SOFTVIS)* (2005), ACM Press, pp. 193–202.
- [244] SUTHERLAND, I. E. A head-mounted three dimensional display. In *Proceedings of the Fall Joint Computer Conference* (1968), pp. 757–764.
- [245] TAVANTI, M., AND LIND, M. 2D vs 3D, implications on spatial memory. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)* (2001), IEEE Computer Society Press, pp. 139–145.
- [246] TELEA, A., MACCARI, A., AND RIVA, C. An open toolkit for prototyping reverse engineering visualizations. In *Proceedings of the symposium on Data Visualisation (VISSYM)* (2002), Eurographics Association, pp. 241–249.
- [247] TESLER, J., AND STRASNICK, S. FSN: the 3D file system navigator. Tech. rep., Silicon Graphics, 1992.
- [248] THE OBJECT MANAGEMENT GROUP. Unified Modeling Language (UML), version 2.1.1, 2007. <http://www.omg.org/technology/documents/formal/uml.htm>.
- [249] THORNE, C., AND WEILEY, V. Earth’s avatar: the web augmented virtual earth. In *SIGGRAPH ’03: ACM SIGGRAPH 2003 Web Graphics* (2003), ACM Press, pp. 1–1.
- [250] TOLLIS, I. G., BATTISTA, G. D., EADES, P., AND TAMASSIA, R. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1998.
- [251] TORY, M., AND MOLLER, T. Evaluating visualizations: Do expert reviews work? *IEEE Computer* 25, 5 (2005), 8–11.
- [252] TUDOREANU, M. E., WU, R., HAMILTON-TAYLOR, A., AND KRAEMER, E. Empirical evidence that algorithm animation promotes understanding of distributed algorithms. In *Proceedings of the IEEE Symposium on Human Centric Computing Languages and Environments (HCC)* (2002), IEEE Computer Society Press, p. 236.

- [253] TUFTE, E. R. *The visual display of quantitative information*. Cheshire: Graphics Press, 1983.
- [254] VALIATI, E. R. A., PIMENTA, M. S., AND FREITAS, C. M. D. S. A taxonomy of tasks for guiding the evaluation of multidimensional visualizations. In *Proceedings of the AVI Workshop on BEyond time and errors: novel evaLUation methods for Information Visualization (BELIV)* (2006), ACM Press, pp. 1–6.
- [255] VAN WIJK, J. J. The value of visualization. In *Proceedings of the IEEE Conference on Visualization (VIS)* (2005), IEEE Computer Society Press, pp. 79–86.
- [256] VAN WIJK, J. J., VAN HAM, F., AND VAN DE WETERING, H. Rendering hierarchical data. *Communications of the ACM* 46, 9 (2003), 257–263.
- [257] WALSH, A. E., AND BOURGES-SEVENIER, M. *Core Web 3D*. Prentice Hall, 2000.
- [258] WARE, C. *Information Visualization: Perception for Design*. Morgan Kaufmann, 2004.
- [259] WARE, C., ARTHUR, K., AND BOOTH, K. S. Fish tank virtual reality. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)* (1993), ACM Press, pp. 37–42.
- [260] WARE, C., AND FRANCK, G. Viewing a graph in a virtual reality display is three times as good as a 2D diagram. In *Proceedings of the IEEE Symposium on Visual Languages (VL)* (1994), IEEE Computer Society Press, pp. 182–183.
- [261] WARE, C., AND FRANCK, G. Evaluating stereo and motion cues for visualizing information nets in three dimensions. *ACM Transactions on Graphics (TOG)* 15, 2 (1996), 121–140.
- [262] WARE, C., FRANCK, G., PARKHI, M., AND DUDLEY, T. Layout for visualizing large software structures in 3D. In *Proceedings of the International Conference on Visual Information Systems (VISUAL)* (1997), Springer Verlag, pp. 215–223.
- [263] WARE, C., HUI, D., AND FRANCK, G. Visualizing object oriented software in three dimensions. In *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)* (1993), IBM Press, pp. 612–620.
- [264] WEB3D CONSORTIUM. Virtual Reality Modelling Language (VRML97) specification. ISO/IEC 14772:1997, 1997. <http://www.web3d.org/x3d/specifications/vrml/>.



- [265] WEB3D CONSORTIUM. Extensible 3D (X3D) Graphics specification. ISO/IEC 19775:2004, 2004. <http://www.web3d.org/x3d/specifications/x3d>.
- [266] WETTEL, R., AND LANZA, M. Visualizing software systems as cities. In *Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)* (2007), IEEE Computer Society Press, pp. 92–99.
- [267] WIGGINS, M. An overview of program visualization tools and systems. In *Proceedings of the ACM Southeast Regional Conference* (1998), ACM Press, pp. 194–200.
- [268] WIJK, J. J. V., AND VAN DE WETERING, H. Cushion treemaps: Visualization of hierarchical information. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)* (1999), IEEE Computer Society Press, pp. 73–78.
- [269] WINTER, A., KULLBACH, B., AND RIEDIGER, V. An overview of the GXL graph exchange language. In *Revised Lectures on Software Visualization, International Seminar* (2002), Springer Verlag, pp. 324–336.
- [270] WISS, U., CARR, D., AND JONSSON, H. Evaluating three-dimensional information visualization designs: A case study of three designs. In *Proceedings of the IEEE International Conference on Information Visualisation (IV)* (1998), IEEE Computer Society Press, pp. 137–144.
- [271] WISS, U., AND CARR, D. A. An empirical study of task support in 3D information visualizations. In *Proceedings of the IEEE International Conference on Information Visualisation (IV)* (1999), IEEE Computer Society Press, pp. 392–399.
- [272] WYSSEIER, C. Interactive 3D visualization of feature-traces. Master’s thesis, University of Berne, 2005.
- [273] XIE, X., POSHYVANYK, D., AND MARCUS, A. 3D visualization for concept location in source code. In *Proceedings of the International Conference on Software Engineering (ICSE)* (2006), ACM Press, pp. 839–842.
- [274] YOUNG, P. Three dimensional information visualisation. Tech. Rep. 12, University of Durham, 1996.
- [275] YOUNG, P., AND MUNRO, M. A new view of call graphs for visualising code structures. Tech. Rep. 03/97, University of Durham, 1997.

- [276] ZHANG, K., Ed. *Software Visualization: From Theory to Practice*. Kluwer Academic Publishers, 2003.
- [277] ZHU, N., GRUNDY, J., AND HOSKING, J. Pounamu: A meta-tool for multi-view visual language environment construction. In *Proceedings of the IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC)* (2004), IEEE Computer Society Press, pp. 254–256.
- [278] ZUK, T., AND CARPENDALE, S. Theoretical analysis of uncertainty visualizations. In *Proceedings of the Visualization and Data Analysis Conference at Electronic Imaging* (2006), vol. 6060, SPIE, p. 606007.
- [279] ZUK, T., SCHLESIER, L., NEUMANN, P., HANCOCK, M. S., AND CARPENDALE, S. Heuristics for information visualization evaluation. In *Proceedings of the AVI Workshop on BEyond time and errors: novel evaLuation methods for Information Visualization (BELIV)* (2006), pp. 1–6.