

Clinton Jeffery · Jafar Al-Gharaibeh

# Writing Virtual Environments for Software Visualization

 Springer

# Writing Virtual Environments for Software Visualization

Clinton Jeffery • Jafar Al-Gharaibeh

# Writing Virtual Environments for Software Visualization



Springer

Clinton Jeffery  
Department of Computer Science  
University of Idaho  
Moscow  
ID  
USA

Jafar Al-Gharaibeh  
R&D Department  
Architecture Technology Corporation  
Eden Prairie  
MN  
USA

ISBN 978-1-4614-1754-5

ISBN 978-1-4614-1755-2 (eBook)

DOI 10.1007/978-1-4614-1755-2

Library of Congress Control Number: 2014957537

Springer New York Heidelberg Dordrecht London

© Springer Science+Business Media, LLC 2015

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))

# Preface

This book combines two emergent research areas of the computer graphics world. On the one hand, there is software visualization. After 30 years, it remains an under-achiever with enormous potential to impact how we develop software. This book looks at how to visualize dynamic program behavior, while it is happening. The techniques are readily applicable to other areas of software visualization. Our ultimate goal is to make visible many practical aspects of program behavior that are currently invisible or difficult to see at best, such as how to find bugs and performance bottlenecks.

A second major focus of this book is on virtual environments. Lots of programmers want to create them, but for most that challenge is just *too* hard; virtual environments integrate advanced 3D graphics, animation and networking in ways that most ordinary developers can't manage in a practical time frame. We are writing this book to do what we can to help conquer that obstacle.

It is obvious that these two genres of computer graphics should be combined. The networking and persistence afforded by virtual environments are exactly what software visualization needs in order to become more collaborative, shared, ubiquitous, and educational. All we have to do is figure out how to build it.

*Unicon* is an innovative, very high-level programming language hosted at [unicon.org](http://unicon.org) and the popular open source site SourceForge. It is descended from Icon, a language developed at the University of Arizona as a successor to the SNOBOL language family from AT&T. Unicon's 3D and networking facilities turn out to be great for virtual environments, and its execution monitoring facilities make it second to none in the area of software visualization. Programs in other languages can often be instrumented to extract similar program behavior information, but in Unicon 120 or so kinds of program execution behavior are continuously available, any time they are of interest.

This book is organized into two parts. Part I is an overview of execution monitoring and program visualization. It presents Unicon's monitor architecture and the framework for monitoring Unicon programs using a series of example visualization tools that observe many kinds of execution. Part II presents virtual environments, including 3D modeling and network communication necessary for multi-user collaboration. It introduces methods of propagating visualization

information and collaborating within a multi-user virtual environment. The ultimate goal is to produce game-like real-time graphical ways of understanding and talking about bugs, bottlenecks, and other aspects of program behavior. Following Part II is a collection of appendices including some detailed program examples and a description of the implementation of the monitoring framework.

# Acknowledgments

Intellectual contributions to our work were made in the University of Idaho and New Mexico State University graduate courses on program monitoring and visualization, and on collaborative virtual environments. The authors wish to thank the students from those courses.

This book descends from the 1999 tome *Program Monitoring and Visualization*, whose acknowledgements section recognizes Ralph Griswold, Rick Snodgrass, Mary Bailey, Norm Hutchinson, Wenyi Zhou, Kevin Templer, Gregg Townsend, Ken Walker, Michael Brazell, Darren Merrill, Mary Cameron, Jon Lipp, Nick Kline, Song Liang, Kevin Devries, Steve Wampler, Thanawat Lertpradist, Laura Connor, Anthony Jones, Khan Mai, and Niem Tang.

Brett Kurzman of Springer is responsible for initiating this book. Mary James and Rebecca Hytowitz at Springer provided the kind of patience and support that all authors want.

This work was supported in part by the National Science Foundation under grant DUE-0402572, and by the National Library of Medicine Division of Specialized Information Services.

Moscow, Idaho and Eden Prairie,  
Minnesota, September 2014

Clinton L. Jeffery and  
Jafar M. Al-Gharaibeh

# Contents

<b>1 Introduction</b> .....	1
1.1 Software Visualization and Program Behavior .....	1
1.2 Types of Software Visualization Tools.....	2
1.3 Virtual Environments .....	3
1.4 Scope of This Book.....	3
1.5 Contributions.....	5
References.....	6
<b>Part I Software Visualization</b>	
<b>2 Visualization Principles and Techniques</b> .....	9
2.1 Graphic Design Principles.....	10
2.1.1 Show the Information.....	10
2.1.2 Maximize Information Density .....	10
2.1.3 Remove Useless Information .....	11
2.1.4 Remove Redundant Information .....	12
2.1.5 Iterate.....	12
2.2 Visualization Principles.....	13
2.2.1 Animation.....	13
2.2.2 Least Astonishment .....	14
2.2.3 Visual Metaphors.....	14
2.2.4 Interconnection.....	14
2.2.5 Interaction .....	15
2.2.6 Dynamic Scale .....	15
2.2.7 Static Backdrop .....	17
2.3 Visualization Techniques.....	17
2.3.1 Incremental Algorithms.....	18
2.3.2 Radial Coordinates.....	18
2.3.3 Mapping Data to Colors, Textures, and Shapes .....	19
References.....	21



- 3 Software Instrumentation and Data Collection** ..... 23
  - 3.1 The Role of Static Information..... 23
  - 3.2 Log Files and Real-Time Event Monitoring ..... 24
  - 3.3 Inventory of Architecture Components..... 25
  - 3.4 Standard Execution Monitoring Scenario ..... 26
    - 3.4.1 Sources of Relevant Execution Behavior..... 27
    - 3.4.2 Selecting or Developing Appropriate Monitors ..... 28
    - 3.4.3 Running the Target Program ..... 28
  - 3.5 Framework Characteristics..... 29
    - 3.5.1 Multitasking ..... 29
    - 3.5.2 Independence..... 30
    - 3.5.3 Access..... 30
    - 3.5.4 Multiple Monitors and Monitor Coordinators..... 30
    - 3.5.5 Execution Control ..... 31
    - 3.5.6 Visualization Support ..... 33
    - 3.5.7 Simplified Graphics Programming ..... 33
    - 3.5.8 Support for Dual Input Streams ..... 33
    - 3.5.9 Inter-task Access Functions..... 34
  - 3.6 Events..... 35
    - 3.6.1 Event Codes and Values ..... 35
    - 3.6.2 Event Reporting and Masking..... 36
    - 3.6.3 Obtaining Events Using evinit ..... 37
    - 3.6.4 Setting Up an Event Stream ..... 37
    - 3.6.5 EvGet()..... 37
    - 3.6.6 Event Masks, and Value Masks..... 38
    - 3.6.7 Artificial Events ..... 39
  - 3.7 Instrumentation in the Unicon Runtime System ..... 40
    - 3.7.1 Explicit Source-Related Events..... 40
    - 3.7.2 Implicit Runtime System Events..... 40
  - 3.8 Anatomy of an Execution Monitor..... 41
    - 3.8.1 Handling User Input..... 42
    - 3.8.2 Querying the Target Program for more Information ..... 42
    - 3.8.3 2D Graphics Capabilities ..... 43
    - 3.8.4 Some Useful Library Procedures ..... 44
  - 3.9 Typical Evolution of a Visualization Tool..... 45
    - 3.9.1 Generate Log Files ..... 45
    - 3.9.2 Depict the Log Files..... 45
    - 3.9.3 Scale to Handle Real Problems ..... 46
    - 3.9.4 Focus on Behaviors of Interest..... 46
    - 3.9.5 Add User-Directed Navigation..... 46
  - 3.10 Instrumenting Code Written in Other Languages ..... 46
- References ..... 47

- 4 Visualizing Aspects of Program Behavior** ..... 49
  - 4.1 Following the Locus of Execution ..... 49
    - 4.1.1 Location Events..... 50
    - 4.1.2 A Simple Line Number Monitor ..... 50
    - 4.1.3 A Location Profile Scatterplot..... 51
    - 4.1.4 Tracking Source File Changes ..... 54
  - 4.2 Monitoring Procedure Activity ..... 55
    - 4.2.1 Activation Trees ..... 55
    - 4.2.2 An Animated Call-Result Scatterplot..... 59
    - 4.2.3 Algae ..... 63
    - 4.2.4 Maintaining the Current Source File..... 67
  - 4.3 Monitoring Memory Usage..... 69
    - 4.3.1 Allocation by Type ..... 69
    - 4.3.2 Cumulative Allocation by Type..... 72
    - 4.3.3 Running Allocation by Type ..... 75
    - 4.3.4 Survival Rates Across Collections ..... 77
  - 4.4 Monitoring String Scanning..... 78
    - 4.4.1 Overview of String Scanning..... 78
    - 4.4.2 String Scanning Events ..... 79
    - 4.4.3 Absolute and Relative Position Changes..... 79
    - 4.4.4 The Scanning Environment Tree..... 82
  - 4.5 Monitoring Structure and Variable Usage..... 85
    - 4.5.1 Visualizing Lists and List Accesses ..... 85
    - 4.5.2 Visualizing Heterogeneous Structures ..... 89
    - 4.5.3 Monitoring Variable References ..... 92
  - References..... 96
  
- 5 Integrating Multiple Views**..... 97
  - 5.1 Monitor Coordinators..... 97
  - 5.2 Eve: The Reference Monitor Coordinator..... 99
    - 5.2.1 Computation of the Minimal Event Set ..... 100
    - 5.2.2 The Event Code Table ..... 101
    - 5.2.3 Event Handling..... 101
    - 5.2.4 Eve’s Main Loop..... 102
    - 5.2.5 Interactive Error Conversion..... 103
  - 5.3 Writing Monitors that Run Under a Monitor Coordinator ..... 104
    - 5.3.1 Client Environment ..... 104
    - 5.3.2 General-Purpose Artificial Events..... 104
    - 5.3.3 Monitor Communication Example..... 105
  - 5.4 Integrating Visualizations Using Subwindows ..... 105
  - References..... 106
  
- 6 Sharing Visualizations Across a Network** ..... 107
  - 6.1 Connections: Firewalls, Hole Punching, and Tunneling..... 107
    - 6.1.1 Firewalls..... 108
    - 6.1.2 Resorting to Existing Protocols..... 108

- 6.1.3 Hole Punching..... 108
- 6.1.4 Tunneling..... 109
- 6.2 Network Protocols and Message Types ..... 109
  - 6.2.1 Transmitting Strings and Structures..... 109
  - 6.2.2 Serialization..... 110
  - 6.2.3 Buffering ..... 110
- 6.3 Messages that Convey Graphics Output ..... 110
- 6.4 Messages for Events and Program Behavior ..... 111
- 6.5 A Remote Memory Allocation Visualizer ..... 112

**Part II Virtual Environments**

- 7 An Overview of Virtual Environments..... 115**
  - 7.1 Categories..... 115
  - 7.2 Features and Usage Patterns..... 117
  - 7.3 Users and Avatars..... 117
  - 7.4 Collaborative Views ..... 118
  - 7.5 Virtual Objects..... 118
  - 7.6 CVE: A Brief History ..... 118
  - References ..... 119
- 8 Virtual Worlds Graphics and Modeling ..... 121**
  - 8.1 3D Graphics and Modeling 101 ..... 121
  - 8.2 The Programming Language as a Graphics Engine ..... 122
    - 8.2.1 3D Graphics API ..... 122
    - 8.2.2 Improving 3D graphics performance ..... 124
  - References ..... 125
- 9 Non-Player Characters and Quests ..... 127**
  - 9.1 NPCs ..... 128
    - 9.1.1 NPC Profiles..... 129
    - 9.1.2 Knowledge Model..... 129
    - 9.1.3 Dialogue Model..... 129
    - 9.1.4 Behavior Model..... 130
  - 9.2 Quests..... 131
    - 9.2.1 Quest Repository..... 132
    - 9.2.2 Quest Rating and User Reward via Peer Review..... 132
  - 9.3 Design and Implementation ..... 133
    - 9.3.1 NPC Architecture ..... 133
    - 9.3.2 Implementation Discussion..... 134
  - References ..... 138
- 10 Dynamic Texturing in Virtual Environments..... 139**
  - 10.1 Reloading Textures..... 139
  - 10.2 Textures as Windows..... 140
  - 10.3 Dynamic Texture Examples ..... 141

**11 Embedding Visualizations in a Virtual Environment** ..... 145

- 11.1 Placement of Visualizations in Predefined Locations ..... 145
- 11.2 Distributed Clones ..... 147
- 11.3 Control Flow and Time Warp ..... 147
- 11.4 Bug Hunting ..... 149

**Appendices**

- Appendix A: Event Codes ..... 151
  - A.1 Classes of Events ..... 151
  - A.2 Individual Events ..... 152
- Appendix B: Software and Supporting Documentation ..... 155
- References ..... 155

# List of Figures

<b>Fig. 2.1</b>	A classical tree layout.....	11
<b>Fig. 2.2</b>	A tree-map emphasizes nodes instead of edges.....	12
<b>Fig. 2.3</b>	A fisheye view with a single focus point.....	16
<b>Fig. 2.4</b>	A graphical fisheye view.....	17
<b>Fig. 2.5</b>	A circular tree.....	18
<b>Fig. 2.6</b>	A cone tree.....	20
<b>Fig. 2.7</b>	Rotating subtrees to reveal hidden children.....	20
<b>Fig. 3.1</b>	Unicon’s execution monitoring architecture.....	25
<b>Fig. 3.2</b>	Monitoring starts with a user, a program, and questions.....	27
<b>Fig. 3.3</b>	Behavior depends on the language, not just the program.....	27
<b>Fig. 3.4</b>	EMs can answer questions about TP behavior.....	28
<b>Fig. 3.5</b>	EM and TP are separately loaded coroutines.....	29
<b>Fig. 3.6</b>	Multiple EMs.....	31
<b>Fig. 3.7</b>	An execution monitor coordinator.....	31
<b>Fig. 3.8</b>	Layers in the Unicon implementation.....	32
<b>Fig. 3.9</b>	Event-driven control of TP.....	33
<b>Fig. 3.10</b>	Monitors have two input streams.....	34
<b>Fig. 4.1</b>	A simple line number monitor.....	50
<b>Fig. 4.2</b>	Monitoring adjacent pairs of lines.....	52
<b>Fig. 4.3</b>	Mapping CPU clock ticks to pixel columns.....	52
<b>Fig. 4.4</b>	A location profile scatterplot.....	53
<b>Fig. 4.5</b>	An activation tree.....	56
<b>Fig. 4.6</b>	A Unicon representation of an activation tree.....	57
<b>Fig. 4.7</b>	A scatterplot with motion.....	59
<b>Fig. 4.8</b>	Algae.....	64
<b>Fig. 4.9</b>	Pinwheel.....	70
<b>Fig. 4.10</b>	Nova.....	72
<b>Fig. 4.11</b>	Frequent large allocations suggest a problem. The program runs twice as fast after a two-line change.....	72
<b>Fig. 4.12</b>	An animated bar graph.....	74
<b>Fig. 4.13</b>	A pie chart.....	75

**Fig. 4.14** A memory allocation strip chart ..... 76

**Fig. 4.15** A string scanning environment..... 78

**Fig. 4.16** Absolute string position..... 80

**Fig. 4.17** Relative string position..... 81

**Fig. 4.18** Scanning environment trees and operations ..... 82

**Fig. 4.19** A list access monitor..... 86

**Fig. 4.20** A moderate number of lists..... 86

**Fig. 4.21** A large number of lists..... 87

**Fig. 4.22** Heterogeneous structure visualization..... 90

**Fig. 4.23** Before and after a merge of two components..... 91

**Fig. 4.24** Monitoring variables in active procedures ..... 93

  

**Fig. 5.1** Monitoring a monitor; monitoring multiple TPs..... 98

**Fig. 5.2** Forwarding events to an assistant..... 98

**Fig. 5.3** Monitor coordinators ..... 99

**Fig. 5.4** Eve’s control window ..... 100

**Fig. 5.5** Eve’s interactive error converter..... 103

  

**Fig. 7.1** Virtual reality: flight simulator..... 116

**Fig. 7.2** The use of augmented reality in a sport show ..... 117

  

**Fig. 8.1** A 3D window in Unicon keeps track of all the scene content using a display list ..... 123

  

**Fig. 9.1** NPCs and other users in CVE. An NPC is marked with red ball above their heads ..... 128

**Fig. 9.2** Part of the grammar for the PNQ NPC behavior model..... 130

**Fig. 9.3** A sample CVE quest invitation dialog ..... 132

**Fig. 9.4** An example quest as it appears in a web page ..... 133

**Fig. 9.5** PNQ NPC architecture ..... 134

**Fig. 9.6** NPC quest messages between the NPC, the server and the client..... 136

**Fig. 9.7** A very simple and compact NPC client example ..... 137

  

**Fig. 10.1** *Left:* the original texture shown on three sides of the cube. *Right:* the texture after getting updated with another image ..... 142

**Fig. 10.2** *Left:* original texture. *Right:* the result of drawing on the texture dynamically ..... 143

  

**Fig. 11.1** A user in CVE watching two virtual computer screens sharing the same dynamic texture ..... 146

**Fig. 11.2** Two 3D visualizations running at two different speeds inside CVE..... 149

# List of Tables

<b>Table 3.1</b>	Unicon interprogram access functions .....	35
<b>Table 3.2</b>	Some useful graphics functions.....	43
<b>Table 3.3</b>	Additional library procedures for monitors.....	44
<b>Table 4.1</b>	Scope Codes .....	92
<b>Table 9.1</b>	The major parts in an NPC profile file .....	129
<b>Table 9.2</b>	Summary of the most important NPC protocol messages.....	135

# Chapter 1

## Introduction

As the demand for computer software grows to include more diverse and larger applications, software developers desperately need better tools to aid in the understanding of dynamic aspects of program behavior during various phases of the software life cycle, including debugging, performance tuning, and maintenance.

Spectacular improvements in graphics, concurrency and networks during the past decade have put us in a position where we ought to be able to see what is going on while our programs are running, in rich, animated 3D, and to easily share and consult with each other while studying such visualizations of program behavior. It ought to be no harder than, say, playing *World of Warcraft*.

This book applies videogame technology to software development by making the vision of multi-user online collaborative 3D software visualization a reality. The topic will be explored by demonstrating an example implementation, getting down into its code.

### 1.1 Software Visualization and Program Behavior

The reason to want software visualization is to gain a better understanding of a program's behavior. Some program-understanding systems convey very specific information about a small portion of a program, such as the workings of a single algorithm. Others are concerned with explaining the role that a program or a collection of programs plays within a larger computational system. This book addresses a common problem in between these two extremes: understanding the workings of a single (possibly large) program.

People who need to understand a program usually have two alternatives: studying the source code, or running the program to see what it does. Ideally, a program would be understandable using one or the other of these methods; in practice, reading source code is impractically cumbersome for many programs, and construction of test cases to explain program behavior is often a tedious and speculative undertaking. These difficulties motivate the development of special programs that are used to help explain the behavior of other programs.



Program-understanding systems are used in a variety of applications. The most common motive for program-understanding is *debugging*. Programs that produce incorrect output or fail to complete their execution due to bugs are prime candidates for tools that assist program developers and maintainers in program-understanding tasks. A *debugger* is a program designed specifically to help with the debugging process. General-purpose program-understanding tools are also used to assist in *debugging*.

A second major application of program-understanding systems is *performance tuning* or *performance debugging*. A correct, working program may be of limited usefulness if its performance is poor. Frequently, a program's authors or maintainers can improve execution speed by using different programming techniques or modifying the program's algorithms and data structures. By providing an accounting of which resources the program is using and which sections of code are primarily responsible, performance tuning systems can direct programmers' efforts to where they are most needed.

A third application of program-understanding is *software instruction* and *orientation*. The internal workings of a program may be of special interest to students learning important algorithms, data structures, or programming techniques; this situation frequently arises when learning a new language. People assigned to maintain or improve a program written by someone else similarly need to *orient* themselves as to its general operation. In both of these cases, the people involved may be entirely unfamiliar with the program source code, and can benefit from information provided by program understanding tools before consulting source code, or without referring to it at all.

In addition to these established uses for program-understanding systems, program-understanding tools can provide language implementors with valuable assistance in the task of *language implementation tuning*. Program-understanding tools that provide information about the execution of programs also directly or indirectly provide information about the language's implementation. This information can be used to improve performance or address problems in the implementation.

## 1.2 Types of Software Visualization Tools

Programs that provide information about other programs can be separated into two main categories based on the kind of information they provide. *Static analysis tools* examine the program text and, in conjunction with knowledge of the language, provide information about a program that is true for all executions of that program independent of its input [1]. Compiler code optimizers, pretty printers, and syntax-directed editors frequently employ static analysis techniques. This book employs static analysis mainly to provide a spatial context, the terrain within which a user interprets dynamic behavior, perceived as animated objects.

*Dynamic analysis tools* provide information about a specific program execution on a specific set of input data [1]. Since dynamic analysis involves extracting