

# Virtual Worlds for Software Visualisation

Neville Churcher, Lachlan Keown, Warwick Irwin

*Software Visualisation Group, Department of Computer Science, University of Canterbury,  
Private Bag 4800, Christchurch, New Zealand  
{neville,lmk29,wal}@cosc.canterbury.ac.nz*

## Abstract

Understanding, quickly, completely and correctly, is crucial to every phase of the software development process. As system size and complexity continues to grow, effective visualisation of system components, together with their properties and relationships, becomes increasingly important in achieving understanding. Virtual worlds allow more information to be presented in visualisations while minimising the impact of cognitive information overload. In this paper, we describe our application of virtual worlds to visualisation of software engineering artifacts and present some examples from our work on object oriented software systems.

**Keywords:** Software visualisation, software engineering, software metrics, virtual reality, VRML

## 1 Introduction

The process of software development has changed dramatically over the last three decades. The size and complexity of software systems has increased markedly. It is not uncommon for software engineers to be working on projects involving millions of lines of code—far beyond the ability of individuals to comprehend fully. In addition to code, many other component types—including specifications, test plans and version information—are involved and their interrelationships are correspondingly complex. The problems of designing, implementing and maintaining large systems are substantial and have contributed to the failure of many projects [11]. Such pressures have led to various attempts to attack the issue of complexity.

Software reuse [12, 19] requires rapid understanding of both available components and the client application. Software metrics [7, 10, 16] involves extracting knowledge about a software system by measurement. We believe that effective visualisation is a prerequisite to significant advances in these and many other areas.

Much of the code used today is written in languages with some degree of object orientation. While OO simplifies many aspects of software development there are ways in which its richer underlying semantic model actually complicates matters. As discussed elsewhere [6] a data model appropriate for representing OO architectures is substantially more complex, comprising more component categories and relationships than the corresponding models for conventional languages. For example, factors such as overloading and overriding complicate the process of trying to establish the “neighbourhood” of a particular component as required for activities such as reuse.

Without effective and convenient visualisation tools, software engineers face very difficult problems in dealing with system complexity. Our approach is to attempt to augment the tools available to software engineers with non-immersive virtual reality visualisations. We are specifi-

cally interested in developing systems which are effective on standard workstations.

The remainder of the paper is structured as follows. In the next section, we outline some of the reasons we advocate the use of visualisations in software engineering. In section 3 we discuss the advantage of virtual worlds and VRML in developing software visualisations. Some examples of our current applications of these ideas are given in section 4 and some conclusions and suggestions for our future work are contained in section 5.

## 2 Visualisation issues in software engineering

Many attempts have been made to visualise both static and dynamic aspects of software [8]. These can be categorised in various ways [23].

Many diagramming techniques have been developed to represent various aspects of software systems. These include structure charts, data flow diagrams, entity-relationship diagrams and many others [21, for example]. Common techniques for analysing metrics data include histograms and scatter-plots. Other techniques such as kivi charts, Chernoff faces [3], tree-maps [17] and fish-eye views [13, 26], developed to enhance the presentation of complex multivariate data, have been applied to software engineering situations.

Realistic models generally involve multiple dimensions and are challenging to represent visually, particularly in two dimensions. Despite significant advances in the graphing capabilities of tools such as spreadsheets, the representation of systems of realistic size remains problematical. Reasons for this include the following.

- The number variables may be very high (perhaps several hundred) and it is not always possible to determine in advance which combinations will be most significant for particular purposes.
- The ranges of data values can be extreme.

- Outliers are often of particular importance.
- Distributions of data values are generally highly skewed.
- Numeric values may have little absolute significance and subjective comparisons with data from other systems may yield valuable insight.

Such features make software visualisation difficult in general with simple representations such as histograms.

### 3 Virtual worlds for software visualisation

Virtual reality (VR) is an idea which has been around for a number of years. Though initially the realm of fiction [14, for example], many applications, potential and actual, have been identified. General discussions of VR [24, 32, 27] as well as material of a more technical nature [31, 9, 30, 15] are available. We will not be concerned here with immersive VR—involving data gloves, helmets and specialised environments. Non-immersive VR is achieved by using an ordinary display screen to give the impression of navigating through a 3-dimensional space.

Though it is easy to imagine applications of immersive VR to software visualisation, we are primarily interested in developing tools which are available to software engineers using standard workstation platforms. Options include languages such as VRML, which are based on scene-descriptions, or libraries such as java3d which extend general purpose programming languages.

#### 3.1 VRML

The Virtual Reality Modelling Language (VRML) [2] is a language for scene description rather than programming. A VRML world consists of a tree (scene graph) of nodes, each of which has a number of fields. Over 50 primitive nodes types of various categories (such as grouping, geometry, sensors and interpolators) are provided. Some nodes generate events which are routed to other nodes to achieve dynamic behaviour. Events are generated by sensor nodes or when the field values of nodes change. A wiring diagram shows how events, generated by sensor nodes or changes in field values, are routed to other nodes to achieve dynamic behaviour. Custom node types may be written and dynamic behaviour is extended through Java or JavaScript code wired to script nodes.

VRML has many appealing features. VRML files have a simple text format and may be created using a standard editor or generated simply by software tools. Browsers for displaying VRML worlds are freely available for many platforms, often as plug-ins for Internet browsers such as Netscape.

Figure 1 shows a VRML browser implemented as a Netscape plug-in. A range of controls is provided to allow navigation through and manipulation of the world. The world shown contains 1000 generic components laid out on a regular grid. These might represent system modules with

colour indicating most recent modification time and with modules currently being edited rotating. Two-dimensional representations of such situations have been described by Ball and Eick [1]. An artificial horizon is provided to prevent the user from becoming disoriented.

#### 3.2 VR advantages

We advocate the use of non-immersive VR as a powerful tool for software visualisation. The enhanced feeling of “being there” is particularly conducive to exploratory data inspection.

An effective visualisation is inevitably tailored to a specific subset of an application domain. Developing a visualisation is generally a high-effort task requiring the services of specialists in areas such as psychology, HCI or graphics as well as domain experts. The potential benefits include enhanced understanding and availability of information otherwise buried in data.

In contrast, techniques such as the simple histograms and scatter-plots described earlier are much more limited but are much more straightforward to produce.

The third spatial dimension adds much more than the ability to squeeze in another variable. In effect, the representations of every component and connection can now have a “depth”. Furthermore, gradients of variables such as colour and size can be used to convey information. Perspective effects are provided via the browser, avoiding the overheads of fish-eye distortion techniques—though these may be used for additional effects.

VRML’s dynamic features allow applications such as animated display of system evolution, construction of guided tours to be constructed and interaction with applets.

Our goals include providing a degree of control over the visualisation process to the software engineer. Figure 2 shows the architecture of the system used to produce some of the worlds described in this paper. An HTML-like language, XXXX, is used to represent information abstracted from specific programming languages. Generation of the VRML worlds uses mappings specifying the correspondences between the XXXX and VRML variables and properties [18]. For example, a mapping may specify that classes are represented by spheres whose opacity indicates the number of public methods.

## 4 Examples

In this section we present some examples of our current work and outline the software engineering applications and visualisation techniques involved.

#### 4.1 Visualising inheritance structure with cone trees

We will take as an example the object oriented brewery system used by Sharble and Cohen [29, 28] in a study of the differences in the systems resulting from using two different OO design methodologies given the same initial specifications. A simple visualisation of the system architecture resulting from a data-driven design technique is shown

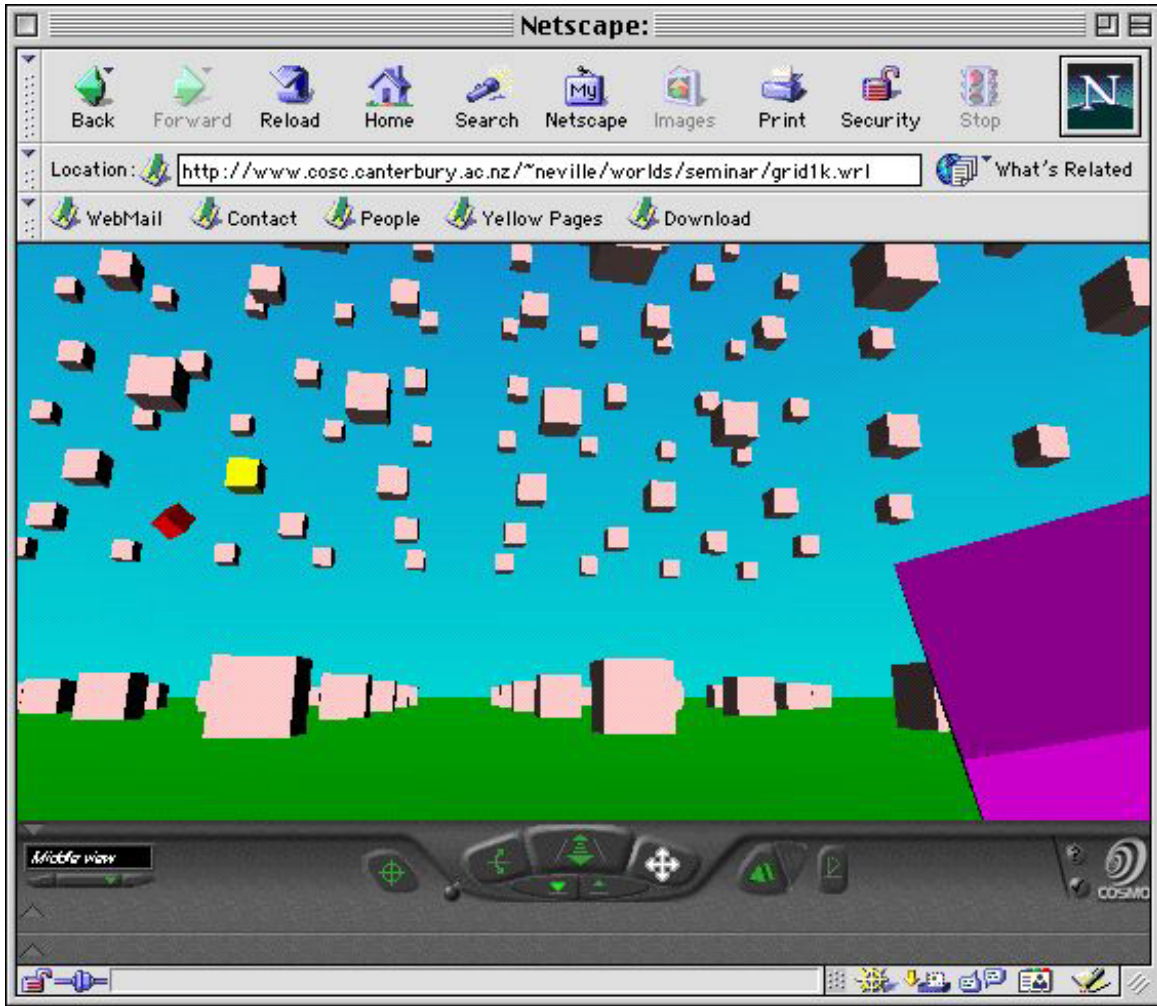


Figure 1: A VRML browser in action

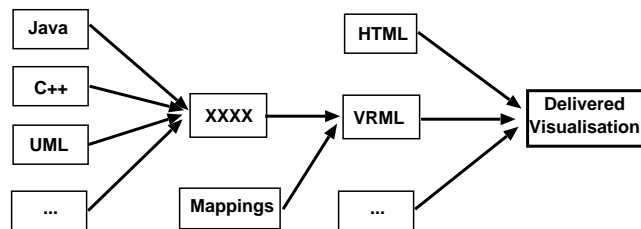
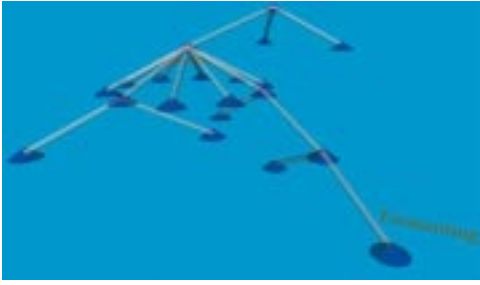


Figure 2: Visualisation generation pipeline



(a) data-driven



(b) responsibility-driven

Figure 3: Cone tree representations of Sharble & Cohen OO brewery architectures

in figure 3(a) and the corresponding architecture resulting from use of a responsibility-driven technique is shown in figure 3(b).

In each case, the individual classes are represented by cones. Connections (extruded cylinders) between classes denote inheritance. The spatial layout is based on a cone tree [25] representation with roots of the type lattice appearing, conventionally, highest in the “up” ( $z$ ) direction. Text labels appear only when the user is close to an object (figure 3(a)). We have also experimented with dashboard style head-up displays for displaying text. Where large amounts of text are involved, the use of HTML frames is an effective solution.

#### 4.2 Visualising inheritance structure with metrics

Sharble and Cohen were interested in using OO software metrics such as the suite proposed by Chidamber and Kemerer [4, 5] and the “law of demeter” [20] to characterise differing architectures.

The Chidamber and Kemerer suite includes two metrics which have potential to help characterise the inheritance structure of a system. These are the Depth In Tree (DIT), which (assuming single inheritance) gives the path length from a class to the root of the tree, and Number Of Children (NOC), which is the number of direct subclasses of a given class.

Distributions of structural fan-in and fan-out enabled systems to be characterised according to their morphology [22] from which other characteristics may be predicted. Similarly, DIT and NOC distributions have potential to highlight features of the inheritance structure of OO systems. In particular, we are interested in ways of highlighting “interesting” features in systems which may be too large for ready comprehension.

Unfortunately, observed distributions of these metrics exhibit the familiar skew and are generally too similar to be useful in detecting features. In an earlier study [6] we introduced the idea of plotting NOC as a function of DIT to reveal features which are otherwise washed out when aggregate representations such as the customary separate

DIT and NOC histograms are used. For example, a large number of children at a large depth is commonly associated with a large number of leaf classes ( $\text{NOC} = 0$ ) at the next level. Such a wide “fan-out” has different design implications at high or low DIT values.

In a study carried out in collaboration with a local developer, we obtained data about many aspects of a core library and several client applications which used its services. All software was written in C++. The core library consisted of 769 classes while the client applications contributed over 4000 additional classes (not including structs or anonymous classes). The primary purpose of this study was to investigate the amount and effectiveness of reuse and detailed results will be reported elsewhere.

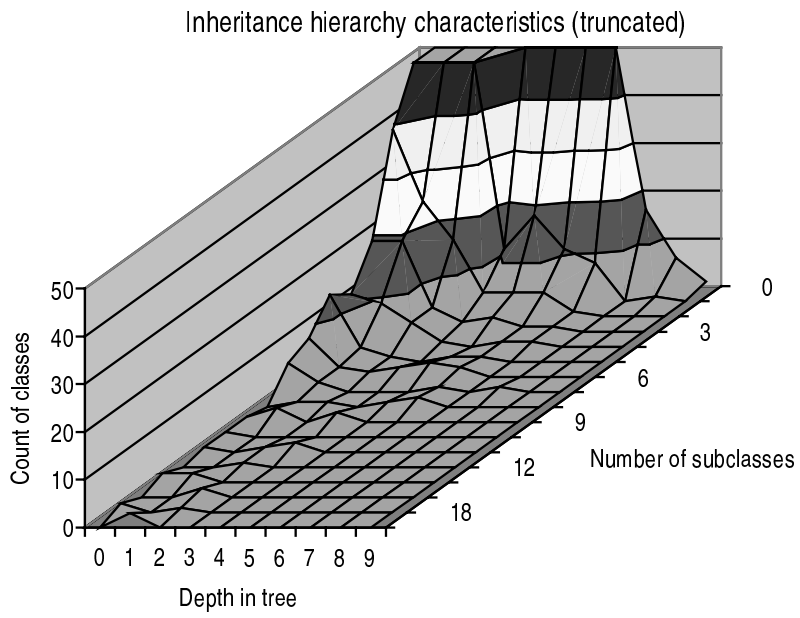
Figure 4(a) shows the distribution of NOC as a function of DIT for the entire system (core library plus client applications) plotted using a conventional diagramming tool. The diagram has been truncated. If this is not done then the large range of values (the peak is over 1300) and the high skew combine to wash out the important detail at high NOC values. However, this approach causes detail to be lost around the peaks and is difficult to manage in a consistent manner.

Figure 4(b) shows a landscape populated with similar plots, constructed from VRML ElevationGrid nodes, for the core library and individual clients. The user can navigate freely among them, zoom in and out to change the field of view, rotate the world and look up or down. The problems evident in figure 4(a) are no longer so serious. If the user is standing at the high NOC end examining detail then the large peaks are simply soaring out of view and looking up is sufficient to examine them also.

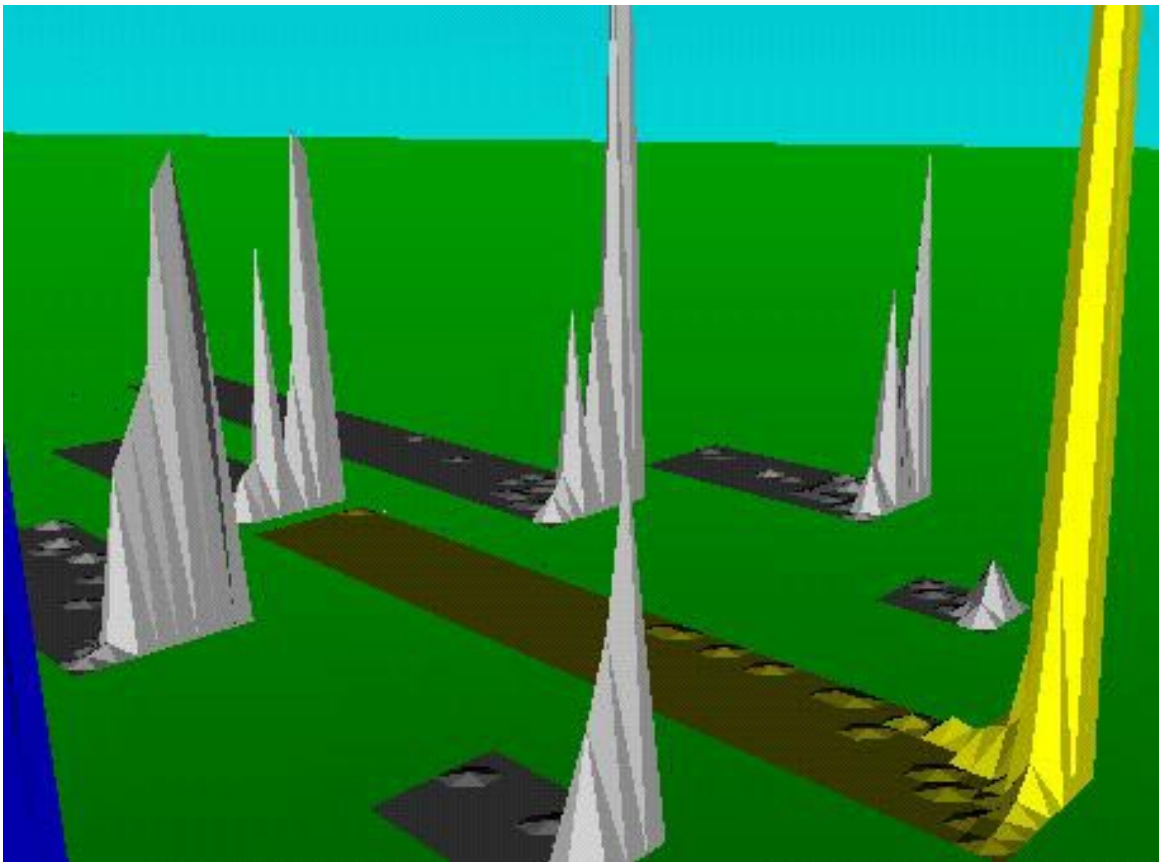
Where precise measurements are required, it is straightforward to use colour to show height ranges and to include grids and transparent planes for orientation and scales. These have been omitted for clarity.

#### 4.3 Visualising hierarchies with tree maps

As a final example, we consider the generalisation of tree maps [17]. Tree maps provide a space-filling



(a) Distribution of NOC(DIT) truncated to show detail



(b) Forest of NOC(DIT) surfaces for individual applications

Figure 4: NOC(DIT) metrics for C++ core library and client applications

representation of hierarchical structures. Leaf nodes are represented by rectangles whose orientation depends on DIT and whose area depends on some arbitrary weighting scheme.

A simple VRML tree map is shown in figure 5(a). It is a representation of an example taken from Johnson and Shneiderman's paper [17]. Nested tree maps also show the structure of internal nodes but these are not commonly used because of their demands on valuable screen real-estate. Many hierarchical structures, such as the inheritance structure of OO software, arise in software visualisation and internal nodes are important.

Figure 5 shows several variations. Even the simplest 3-D form (figure 5(a)) can be remarkably effective. Simply being able to walk around and rotate the slab adds much to the experience.

Weighting in the tree maps could be used to reflect properties such as size or age of the components. Similarly, colour, slab thickness and reflective properties can be used to represent further variables and can be implemented for individual nodes.

Figure 5(b) shows how nesting can be represented easily and effectively in 3D by separating the levels. The user can freely navigate between the layers to explore details of the structure from the perspectives of individual nodes. The regions from where the nested node contents have been raised are transparent so the "flat" tree map may readily be recovered by simply rotating the nested version (or, equivalently, viewing it from above or below). Variations include using the distance between slabs to indicate edge weightings and lateral displacements to represent other properties.

Figure 5(c) shows a "compound" tree map where the conventional tree structure is also included. This view is useful for emphasising the relationship between the weighted tree map and the original structure. However, this combination is particularly useful for situations where a tree map is appropriate at lower levels and would be too complex for the whole of a large tree.

## 5 Conclusions and further work

Virtual worlds are a valuable supplement to existing software engineering tools for system comprehension. VRML worlds are an effective means of providing rapid visualisations of object oriented software systems and overcome many of the difficulties inherent in conventional diagramming or pictorial techniques. Satisfactory performance is achievable with standard workstations and an Internet browser plug-in is the only required software, making the technology readily accessible to practitioners and educators. There is always a trade-off between the effectiveness of a visualisation and a facility for *ad hoc* queries. However, systems such as ours can allow rapid generation of custom visualisations to assist software engineers.

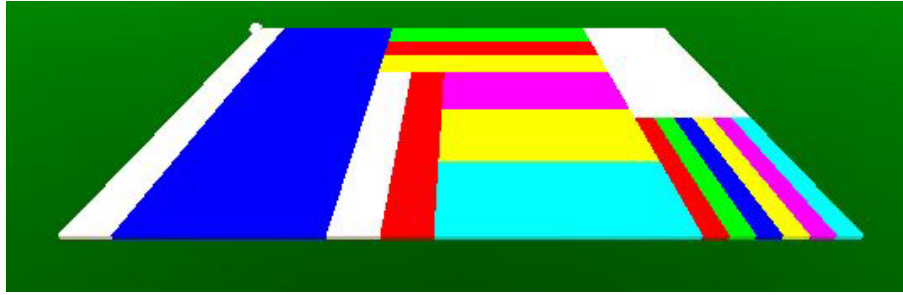
We expect that 3-D visualisations of the structure and characteristics of software artifacts and processes will

complement existing tools available to software engineers, researchers and educators.

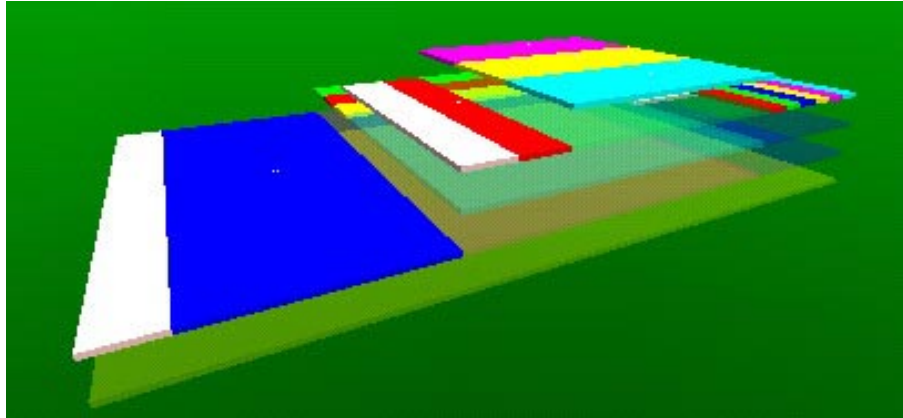
We are currently developing a range of tools for studying reuse in object oriented software systems and interpreting object oriented software metrics. Once user evaluations have been performed we will be able to identify the most effective visualisation models to explore further.

## References

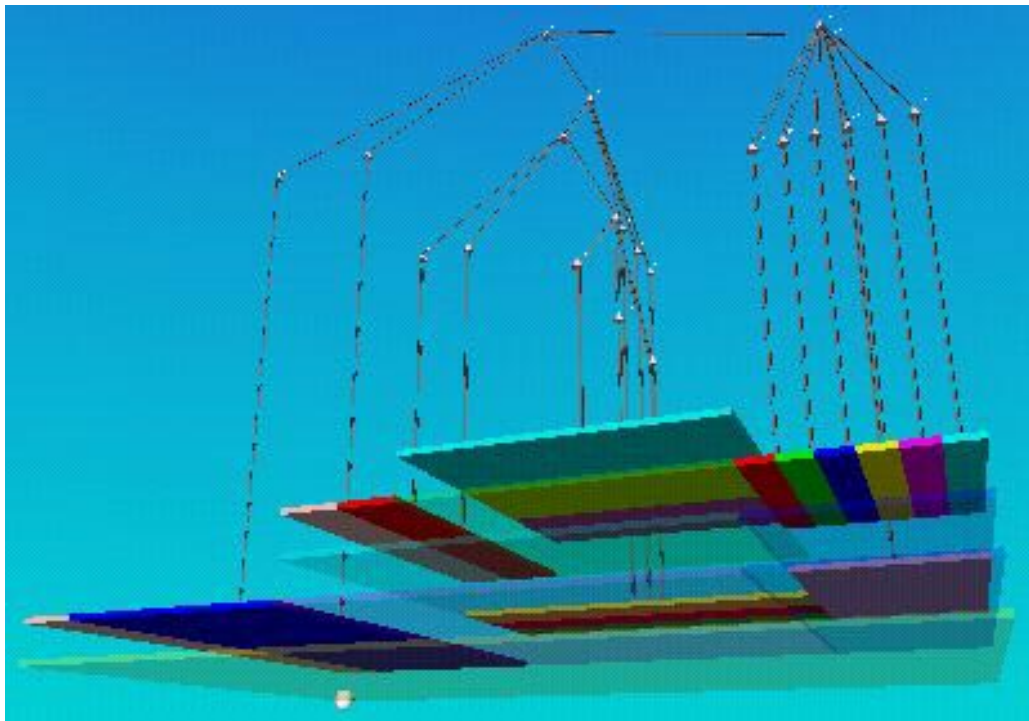
- [1] T. Ball and S.G. Eick. Software visualization in the large. *IEEE Computer*, 29(4):33–43, April 1996.
- [2] R. Carey and G. Bell. *The Annotated VRML 2.0 Reference manual*. Addison-Wesley, 1997.
- [3] H. Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68(342):361–368, 1973.
- [4] S. Chidamber and C. Kemerer. Towards a metric suite for object oriented design. *ACM SIGPLAN Notices*, 26(11):197–211, November 1991.
- [5] S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.
- [6] N.I. Churcher and M.J. Shepperd. Towards a conceptual framework for oo software metrics. *ACM SIGSOFT Software Engineering Notes*, 20(2):69–75, April 1995.
- [7] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. *Software Engineering Metrics and Models*. Benjamin Cummings, 1986.
- [8] P. Eades and K. Zhang, editors. *Software Visualisation*, volume 7 of *Series on Software Engineering and Knowledge Engineering*. World Scientific, 1996.
- [9] R.A. Earnshaw, M.A. Gigante, and H. Jones, editors. *Virtual Reality Systems*. Academic Press, 1993.
- [10] N.E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous & Practical Approach*. International Thompson Computer Press, 2 edition, 1997.
- [11] S. Flowers. *Software Failure: Management Failure—Amazing Stories and Cautionary tales*. J. Wiley & Sons, 1996.
- [12] W. Frakes and C. Terry. Software reuse: Metrics and models. *ACM Computing Surveys*, 28(2):415–435, June 1996.
- [13] G.W. Furnas. Effective view navigation. In S. Pemberton, editor, *CHI 97*, pages 367–374, Atlanta, GA, March 1997.
- [14] W. Gibson. *Neuromancer*. Gollancz, London, 1984.



(a) Simple VRML tree map



(b) Nested VRML tree map



(c) Compound labelled tree map

Figure 5: Variations on tree maps (based on an example from [17]) implemented in VRML

- [15] F. Hamit. *Virtual Reality and the Exploration of Cyberspace*. SAMS Publishing, 1993.
- [16] D. Ince and M. Shepperd. *The Derivation and Validation of Software Metrics*. Oxford University Press, 1992.
- [17] Brian Johnson and Ben Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In G.M. Nielson and L. Rosenblum, editors, *proc. Visialization '91*, pages 284–291, Los Alamitos, CA, October 1991. IEEE Computer Society Press.
- [18] L. Keown. Visualisation of object oriented software. Master's thesis, University of Canterbury, Department of Computer Science, 1999. in preparation.
- [19] C.W. Krueger. Software reuse. *ACM Computing Surveys*, 24(2):131–183, June 1992.
- [20] K.J. Lieberherr and I.M. Holland. Assuring good style for object-oriented programs. *IEEE Software*, pages 38–48, 1989.
- [21] James Martin and Carma McClure. *Diagramming Techniques for Analysts and Programmers*. Prentice Hall, 1985.
- [22] M. Page-Jones. *The practical guide to structured systems design*. Yourdon Press Computing Series. Prentice Hall, Englewood Cliffs., N.J, 2 edition, 1988.
- [23] B.A. Price, R.M. Baecker, and I.S Small. A principled taxonomy of software visualization. *Journal of Visual Languages and Computing*, 4(3):211–266, 1993.
- [24] H. Rheingold. *Virtual Reality*. Summit Books, 1991.
- [25] G.G. Robertson, J.D. Mackinley, and S.K. Card. Cone trees: Animated 3d visualizations of hierarchical information. In *Proc. ACM SIGCHI '91 Conf. on Human Factors in Computing Systems*, pages 189–194, New Orleans, Louisiana, April 1991.
- [26] M. Sarkar and M.H. Brown. Graphical fisheye views. *Communications of the ACM*, 37(12):73–84, December 1994.
- [27] R. Schroeder. *Possible Worlds: the Social Dynamic of Virtual Reality Technology*. Westview Press, 1996.
- [28] R.C. Sharble and S.S. Cohen. The object-oriented brewery: A comparison of two object-oriented development methods. Boeing Company Report BCS-G4059, The Boeing Co., Seattle, Wa, 1992.
- [29] R.C. Sharble and S.S. Cohen. The object-oriented brewery: A comparison of two object-oriented development methods. *ACM SIGSOFT Software Engineering Notes*, 18(2):60–73, 1993.
- [30] J. Vince. *Virtual Reality Systems*. Addison-Wesley, 1995.
- [31] A. Wexelblat, editor. *Virtual Reality: Applications and Explorations*. Academic Press, 1993.
- [32] B. Woolley. *Virtual Worlds*. Blackwell, 1992.