Visual Realism for the Visualization of Software Metrics

Conference Paper · January 2005

DOI: 10.1109/VISSOF.2005.1684299 · Source: DBLP

3 authors:

Danny Holten
IBM Weert, The Netherlands
28 PUBLICATIONS **2,290** CITATIONS
SEE PROFILE

Roel Vliegen
MagnaView
4 PUBLICATIONS **214** CITATIONS
SEE PROFILE

Jarke J. van Wijk
Eindhoven University of Technology
83 PUBLICATIONS **3,895** CITATIONS
SEE PROFILE

Visual Realism for the Visualization of Software Metrics

Danny Holten, Roel Vliegen, and Jarke J. van Wijk
Technische Universiteit Eindhoven
Department of Mathematics and Computer Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
d.h.r.holten@tue.nl, roel.vliegen@magnaview.nl, vanwijk@win.tue.nl

Abstract

The visualization techniques used in current software visualization frameworks make use of a limited set of graphical elements to highlight relevant aspects of a software system. Typical examples of such elements are text, simple geometric shapes and uniform color fills. Although human visual perception enables rapid processing of additional visual cues like shading and texture, they are not used. We contend that such 2D and 3D computer graphics techniques for achieving visual realism can be used to increase the information throughput of software visualization techniques. Visualization results are presented to show how treemaps, cushions, color, texture, and bump mapping can be used to visualize software metrics of hierarchically organized elements of a software system.

1. Introduction

Current software visualization frameworks employ one or more established techniques to visualize the different aspects of existing software systems, such as node-link diagrams and graphs. Such techniques use a limited set of graphical elements, typically consisting of text, simple geometric shapes and uniform color fills to highlight the relevant aspects of the software system being visualized. Additional visual cues like shading (lighting effects) and texture are not used, although the human visual system is very well capable of rapidly processing them [6, 10, 11]. The use of such additional visual cues could therefore help viewers to gain a better understanding of a visualization result in a shorter amount of time while requiring only a minimum amount of perceptual and cognitive load.

We argue that established techniques for achieving visual realism from the field of 2D and 3D computer graphics can be used to enrich the graphical palette of software visualization techniques to establish a higher information throughput. Irani and Ware also state that the use of 3D shaded components for diagram elements can facilitate interpretation and recall of diagrams [13]. Such an approach could be beneficial for the design of new applications for managing and interpreting complex software systems [20].

As an example of how software visualization could benefit from the combination of different visualization and computer graphics techniques, we developed a tool to visualize metrics of elements of a software system, where the elements have a hierarchical relation. We used this to study the hierarchical structure of a Java program consisting of 18,000 lines of code, while simultaneously showing multiple software metrics at the leaf level of the hierarchical structure.

The techniques that we combined for achieving visual realism are treemaps [15], cushions [17], color, texture [12], and bump mapping [3].

Section 2 discusses related work regarding the structural and metric-based visualization of software systems, followed by section 3, in which the techniques mentioned above for achieving visual realism are explained in more detail. Section 4 presents the details of a couple of our visualization results. Conclusions and recommendations are presented in section 5.

2. Related work

Treemaps have been used earlier to visualize software; SeeSys and GAMMATELLA are two examples [1, 14].

SeeSys is used for visualizing statistics associated with code that is divided hierarchically into subsystems, directories, and files [1]. SeeSys uses color to visualize metrics, animation to show software evolution, and standard treemaps to visualize the hierarchical structure of the data. Additional information is displayed by vertically filling each rectangle. A potential problem with this approach is that the additional horizontal boundaries introduced by these fills and the rectangular shape of a fill itself might cause visual interference with the rectangles and lines that are already present within the treemap (see figure 1b). The

use of texture for the visualization of additional information is mentioned, but it is not implemented by SeeSys.

The GAMMATELLA Program Visualizer displays program-execution data gathered from deployed software products [14]. A treemap is used for the system level view (the highest level of abstraction) in which rectangles are proportional in size to the number of statements in a file. Bivariate metric information is displayed using a continuous [red, yellow, green] hue range combined with varying levels of brightness. An issue with such a color-based bivariate visualization technique is that bivariate chromatic maps are notoriously difficult to read [19].

Balzer et al. present a hierarchy-based visualization approach for software metrics using visually appealing layouts based on arbitrary polygons instead of standard rectangle-based treemaps [2]. The layouts are computed by means of iterative relaxation of Voronoi tessellations. An informal study showed that hierarchy and size parameters of the objects were better recognized with Voronoi treemaps than with rectangle-based treemaps. The method is computationally expensive, requiring a couple of minutes instead of seconds to generate a view. This makes the technique less suitable for real-time visualization and exploration of software systems.

Our approach to multivariate, treemap-based software visualization is presented in the following section.

3. Overview of techniques

This section provides a more detailed explanation of the techniques mentioned in the introduction.

3.1. Treemaps

The treemap algorithm uses a space-filling approach to map a tree structure into rectangles with each rectangle representing a node [15]. The size of a rectangle is proportional to the size of the corresponding node. Internal rectangles correspond to child nodes and are visualized by recursively subdividing the containing rectangle ("slice and dice" method). The direction of subdivision alternates per level between horizontal and vertical. A tree and its corresponding treemap are shown in figure 1.

We make use of an alternative to the "slice and dice" subdivision method to generate squarified treemaps in which the rectangles approximate squares, since slicing and dicing often gives thin, elongated rectangles [4]. We use the squarified treemap technique to generate a comprehensible view of how the software is hierarchically subdivided into smaller components.

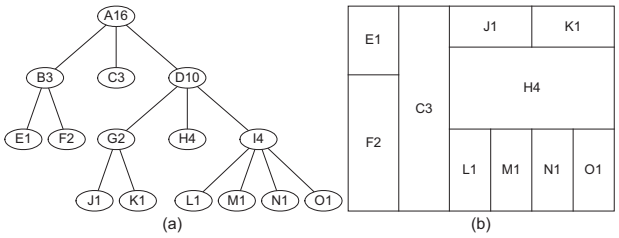

Figure 1. Treemap construction. A tree (a) and its corresponding treemap (b). The numeric part of a node label indicates the size of the node.

3.2. Cushions

Cushions are an extension to standard treemaps that provide a better insight in the structure of hierarchical data by using surfaces that consist of intuitively shaded, recursive cushions [17]. We use cushion treemaps in our freeware tool SequoiaView to visualize the hierarchical contents of a hard disk [18]. The use of cushions to emphasize the hierarchical structure within a treemap is illustrated by means of the one-dimensional example shown in figure 2.

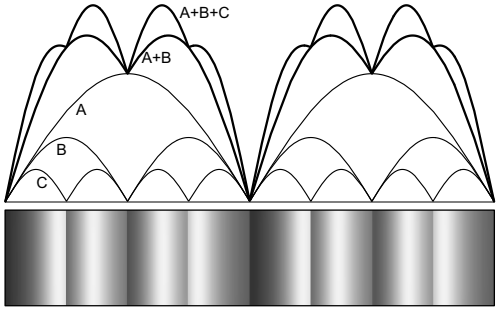

Figure 2. Use of shading to show hierarchical structure. Visualization of the hierarchical structure of a tree by adding the height values of levels A, B, and C together. The resulting height profile is subsequently rendered as viewed from above.

The hierarchy consists of levels A, B, and C. Each level has an associated height profile (based on its depth in the hierarchy) and the resulting height profile for the complete hierarchy ("A+B+C") is calculated by adding the height values of levels A, B, and C together. If the height profile is interpreted as a side view and subsequently rendered as viewed from above, the bumps transform into a sequence of ridges. Because the depth of the valleys between the segments is proportional to the distance between nodes in the tree, the tree structure is clearly visible.

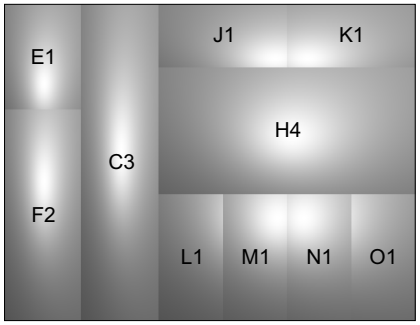

Figure 3. Hierarchical structure shown using a cushion treemap. The hierarchical structure of the tree in figure 1a is visualized using recursive cushions.

3.3. Color

Color ranges are used to visualize a single software metric at the leaf level. The ranges are defined by interpolating between a start and end color in either RGB or HSV space and are mapped to the numerical range of the selected software metric. A grayscale range has been used for the examples in this paper since the proceedings are printed in black and white.

3.4. Texture

To visualize an additional software metric at the leaf level we use a perceptually based texture synthesis model for isotropic textures. The model is based on the additive synthesis of logarithmically binned noise in the discrete 2D frequency domain [12]. Textures synthesized by the model are specifically designed to be usable for the visualization of arbitrary types of ordinal and scalar data. A texture's visual characteristics are specified by using an input model that has been defined on the basis of user experiments and subsequent MDS analysis. This UVC model enables the intuitive and direct manipulation of a texture's spatial frequency (U), regularity (V) and contrast (C) characteristics. Examples of textures that were generated with this model are shown in figure 4.

Furthermore, the texture synthesis model is capable of generating seamlessly tileable textures, which makes the application of the model for the shading of large areas feasible. An extension for generating colored textures is provided as well.

Figure 5 shows the result of combining a grayscale range with textures. Brightness varies from black to white along the horizontal axis while texture contrast and wavelength (spatial frequency^{-1}) vary from minimum to maximum along the vertical axis.

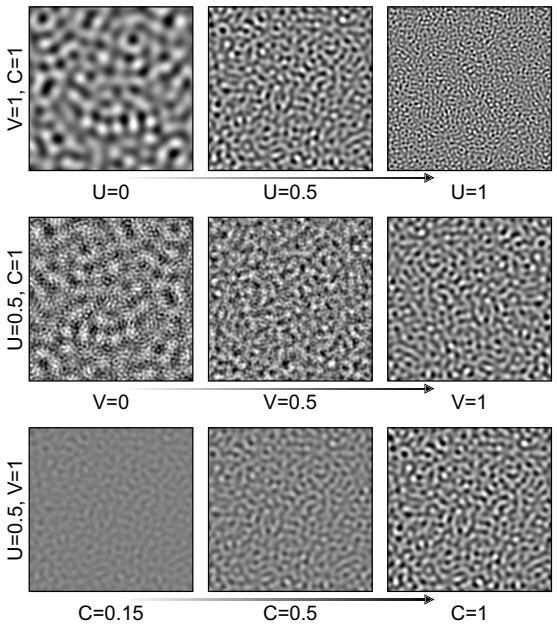

Figure 4. Textures with varying levels of spatial frequency, regularity and physical contrast. The textures were generated using the texture synthesis model presented in [12]. Textures with increasing values of (a) spatial frequency U, (b) regularity V, and (c) physical contrast C are shown.

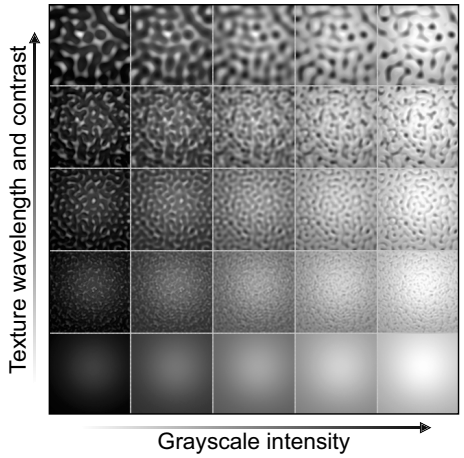

Figure 5. Combining texture and grayscale intensity.

3.5. Bump mapping

Bump mapping is a technique to perturb the surface normals of the object being rendered by using an array of dis-

placements called a bump map [3].

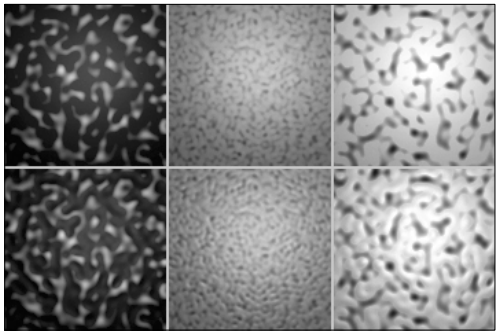

Figure 6. Diffuse texture mapping and bump mapping. Top row: shaded cushions using only diffuse texture mapping; bottom row: shaded cushions using both diffuse texture mapping and bump mapping.

Application of this technique results in images with realistic looking surface wrinkles without the need to model each wrinkle as a separate surface element.

To increase the visual realism, bump mapping can be combined with diffuse texture mapping in which an image is mapped onto the object being rendered, like a decal pasted onto a flat surface [5]. Figure 6 shows the difference between shaded cushions using only diffuse texture mapping, and cushions using both diffuse texture mapping and bump mapping. We used bump mapping in our tool in addition to diffuse texture mapping, such that the texture fuses better with the 3D surfaces of the cushions.

4. Visualization results

Two example visualizations are shown that were generated using the hierarchical structure of a Java program consisting of 18,000 lines of code. The program was developed as part of a software engineering project for third year computer science students. Java packages comprise the highest level of the hierarchical structure, classes comprise the intermediate level, and methods comprise the lowest level, i.e., the treemap leaf nodes. RevJava was used to calculate various software metrics at different levels of the hierarchy [8].

Two hierarchical levels were used to render the cushions: one corresponding to the method level and the other corresponding to the underlying class level. A third cushion level could have been used for the package level, but this caused the shading to become too pronounced. We therefore used a black bounding rectangle to enclose those classes that comprise a single package.

Text labels are shown within the center of each package / class rectangle whenever possible, i.e., if the label fits within the corresponding rectangle. Package labels use a larger font size than class labels to avoid confusion. Method labels are shown as dynamic mouse cursor tooltips instead of static labels to keep the visualization from overwhelming and confusing the viewer. It may be difficult to show text as clearly wrapped on a 3D-shaded object. If the object is textured this is especially likely to interfere with the readability unless the texture is subtle [13]. To make text stand out from the (textured) background we used white, anti-aliased characters surrounded by a black outline and an additional drop shadow. Viewers can zoom in on a package (and eventually on a class) by double-clicking within its corresponding rectangle. This causes the treemap to be redrawn using the full display area to provide an enlarged view of the desired sub level.

The first visualization (figure 7) uses grayscale intensity and texture to show fan-in and fan-out, respectively. The fan-in of method M refers to the number of methods that call M, whereas the fan-out of M refers to the number of methods that are called by M. It is generally desirable that methods with high-fan in have low fan-out to keep such critical methods from depending on (too many) other methods. The second visualization (figure 8) uses grayscale intensity and texture to show fan-in and code smell, respectively. Code smells describe patterns in code that indicate that refactoring can be applied [9]. It is based on the number of critics per method as provided by RevJava.

The visualizations shown in figures 7 and 8 provide a large amount of information at once without being visually overwhelming or confusing:

- The squarified cushion treemaps clearly show the hierarchical structure of the code, i.e., the division into packages, classes, and methods.
- The number of statements per method is proportional to the size of a rectangle, providing information regarding the size of each method.
- Textual information about packages and classes is provided by means of labels, whereas method-level information is shown using tooltips.
- One method-level software metric is visualized using color (a grayscale range in this case due to the reproduction requirements of the proceedings) and a second metric is visualized using diffuse texture mapping, resulting in emergent features, i.e., specific combinations of color and texture, that indicate potential problem areas (see figures 7c and 8c).
- Additional use of the texture as a bump map emphasizes the texture features as well as the shading, fa-

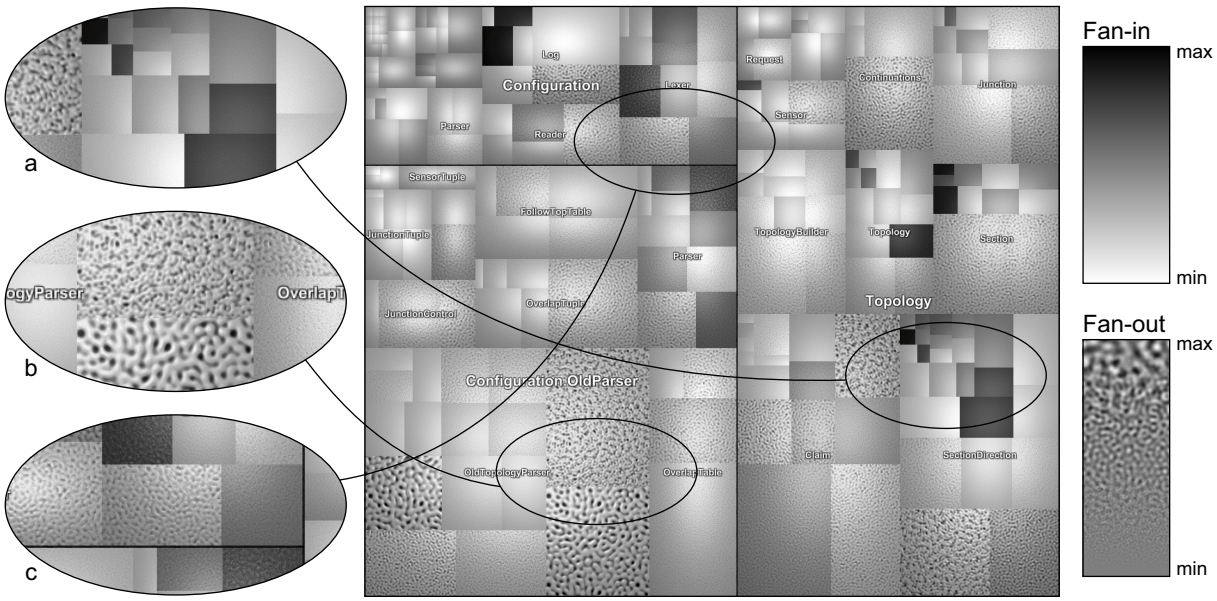

Figure 7. Multivariate visualization of fan-in and fan-out: (a) Methods with high fan-in and low fan-out (permissible situation); (b) Methods with low fan-in and high-fan out (low-priority situation); (c) Two methods (top left, bottom right) with fairly high fan-in and medium fan-out, indicating a potential problem area.

cilitating texture discrimination and recognition of the hierarchical structure, respectively.

- The combination of color and texture is advantageous since this eliminates the need for a visualization technique based on bivariate chromatic maps, which are known to be difficult to read [19]. Explicit visualization of both metrics instead of a combined metric, e.g., the fan-in/fan-out ratio, also has the beneficial property that no information is lost.

5. Conclusions

We have shown how established visualization and computer graphics techniques for achieving visual realism can be combined to increase the expressiveness of software visualizations.

We experienced the resulting visualizations of software systems as esthetically pleasing while providing a high information density without being visually overwhelming or confusing.

5.1. Recommendations

User experiments will have to be performed to gain insight in how other users experience the visualizations and to determine the practical feasibility of such an approach.

Furthermore, the current zooming technique instantly switches between two levels of detail without providing an intuitive transition, which could result in navigational disorientation. An approach using animated zooming and panning could be employed to resolve this [7, 16].

Finally, the techniques that were used are especially suited for implementation on programmable graphics hardware to maintain real-time performance when used to visualize large software systems.

6. Acknowledgements

This work was performed as part of the Reconstructor Project supported by the Netherlands Organization for Scientific Research (NWO) Jacquard program under research grant no. 638.001.408.

References

[1] M. J. Baker and S. G. Eick. *Readings in Information Visualization: Using Vision to Think*, section Space-Filling Software Visualization, pages 160–182. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[2] M. Balzer, O. Deussen, and C. Lewerentz. Voronoi Treemaps for the Visualization of Software Metrics. In *Proceedings of the 2005 ACM Symposium on Software Visualization (SOFTVIS 2005)*, pages 165–172, 2005.

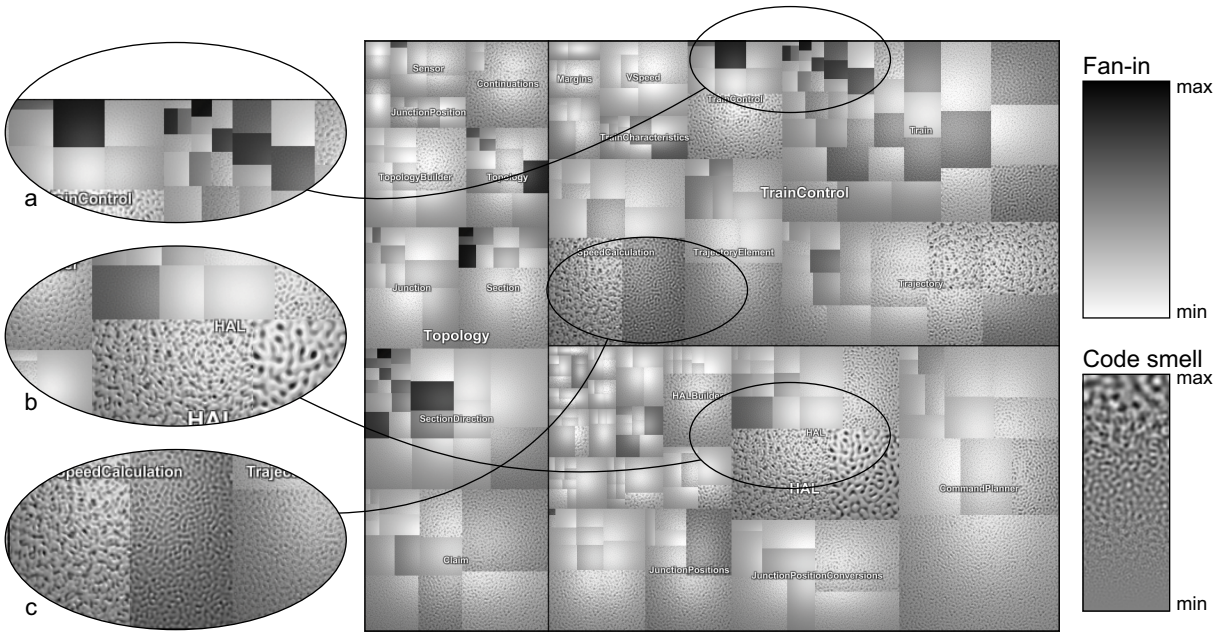

Figure 8. Multivariate visualization of fan-in and code smell: (a) Methods with high fan-in and low code smell (permissible situation); (b) Methods with low fan-in and high code smell (medium-priority situation); (c) A method with medium fan-in and fairly high code smell, indicating a potential problem area.

[3] J. F. Blinn. Simulation of Wrinkled Surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 78)*, pages 286–292, 1978.

[4] D. M. Bruls, C. Huizing, and J. J. van Wijk. Squarified Treemaps. In *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization (TCVG 2000)*, pages 33–42, 2000.

[5] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, University of Utah, 1974.

[6] J. T. Enns. *Three-Dimensional Features that Pop Out in Visual Search*. Taylor & Francis, New York, NY, 1990.

[7] J.-D. Fekete and C. Plaisant. Interactive Information Visualization of a Million Items. In *Proceedings of the 2002 IEEE Symposium on Information Visualization (INFOVIS 2002)*, pages 117–124, 2002.

[8] G. Florijn. RevJava - A Review Assistant for Java Programs, 2003. http://www.serc.nl/people/florijn/work/-designchecking.RevJava.htm.

[9] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Reading, MA, 1999.

[10] C. G. Healey and J. T. Enns. Building Perceptual Textures to Visualize Multidimensional Datasets. In *Proceedings of the Conference on Visualization '98*, pages 111–118, 1998.

[11] C. G. Healey and J. T. Enns. Large Datasets at a Glance: Combining Textures and Colors in Scientific Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2(7):145–167, 1999.

[12] D. Holten. Texture Synthesis for Data Visualization. Master's thesis, Technische Universiteit Eindhoven, 2005. http://www.win.tue.nl/ dholten/mscthesis.pdf.

[13] P. Irani, C. Ware, and M. Tingley. Using Perceptual Syntax to Enhance Semantic Content in Diagrams. *IEEE Computer Graphics and Applications*, 21(5):76–85, 2001.

[14] A. Orso, J. Jones, and M. J. Harrold. Visualization of Program-Execution Data for Deployed Software. In *Proceedings of the 2003 ACM Symposium on Software Visualization (SOFTVIS 2003)*, pages 67–76, 2003.

[15] B. Shneiderman. Tree Visualization with Tree-Maps: 2-D Space-Filling Approach. *ACM Transactions on Graphics (TOG)*, 11(1):92–99, 1992.

[16] J. J. van Wijk and W. A. A. Nuij. Smooth and Efficient Zooming and Panning. In *Proceedings of the 2003 IEEE Symposium on Information Visualization (INFOVIS 2003)*, pages 15–22, 2003.

[17] J. J. van Wijk and H. van de Wetering. Cushion Treemaps: Visualization of Hierarchical Information. In *Proceedings of the 1999 IEEE Symposium on Information Visualization (INFOVIS 1999)*, pages 73–78, 1999.

[18] J. J. van Wijk, H. van de Wetering, K. Huizing, F. van Ham, M. Bruls, J. Geerlings, and E. Melby. SequoiaView, 2003. http://www.win.tue.nl/sequoiaview/index.html.

[19] H. Wainer and C. Francolini. An Empirical Inquiry Concerning Human Understanding of Two-Variable Color Maps. *The American Statistician*, 34(2):81–93, 1980.

[20] C. Ware. Designing with a $2\frac{1}{2}$D Attitude. *Information Design Journal*, 3(10):255–262, 2001.