To be presented at the IEEE Symposium on Information Visualization (INFOVIS'99), San Francisco, October 25-26, 1999

Cushion Treemaps: Visualization of Hierarchical Information

Jarke J. van Wijk   Huub van de Wetering
Eindhoven University of Technology
Dept. of Mathematics and Computing Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{*vanwijk, wstahw*}*@win.tue.nl*

Abstract

A new method is presented for the visualization of hierarchical information, such as directory structures and organization structures. Cushion treemaps inherit the elegance of standard treemaps: compact, space-filling displays of hierarchical information, based on recursive subdivision of a rectangular image space. Intuitive shading is used to provide insight in the hierarchical structure. During the subdivision ridges are added per rectangle, which are rendered with a simple shading model. The result is a surface that consists of recursive cushions. The method is efficient, effective, easy to use and implement, and has a wide applicability.

1 Introduction

Hierarchical structures of information are ubiquitous: family trees, directory structures, organization structures, catalogues, computer programs, etcetera. Small hierarchical structures are very effective to locate information, but the content and organization of large structures is much harder to grasp.

We present a new visualization method for such large hierarchical structures: Cushion Treemaps. The method is based on treemaps, developed by Shneiderman and Johnson [11, 8]. Treemaps are efficient and compact displays, which are particularly effective to show the size of the final elements in the structure. Cushion Treemaps provide shading as a strong extra cue to emphasize the hierarchical structure. For a quick impression, figure 2 and 3 show treemaps, figure 5 and 6 show the corresponding cushion treemaps.

In section 2 we discuss existing methods to visualize hierarchical structures. The new method is presented in section 3. The embedding of the method in an interactive system for tree visualization is described in section 4. Finally, we discuss extensions and alternatives in section 5, and we summarize the results in section 6.

2 Background

Many methods exist to display and browse through hierarchical information structures, or, for short, trees. File browsers are the best known example. Usually a listing of the files and directories is used, where the levels in the hierarchy are shown by means of indentation. The number of files and directories that can be shown simultaneously is limited, which is no problem if one knows what to search for. However, if we want to get an overview, or want to answer a more global question, such as: "Why is my disk full?", scrolling, and opening and closing of subdirectories have to be used intensively. During this process it is hard to form a mental image of the overall structure [3].

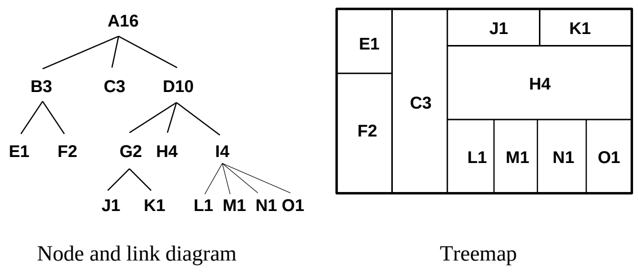


Figure 1. Tree representations

Many techniques have been proposed to show such structures more effectively. An important category are node and link diagrams (fig. 1). Elements are shown as nodes, relations are shown as links from parent to child nodes. Sophisticated techniques have been presented to improve the efficiency and aesthetic qualities of such diagrams, both in 2D and in 3D [9, 7, 1, 2, 10]. Such diagrams are very effective for small trees, but usually fall short when more than a couple of hundred elements have to be visualized simultaneously. The main reason for this limitation is simply that node and link diagrams use the display space inefficiently: Most of the pixels are used as background.

Treemaps [11, 8] were developed to remedy this problem.

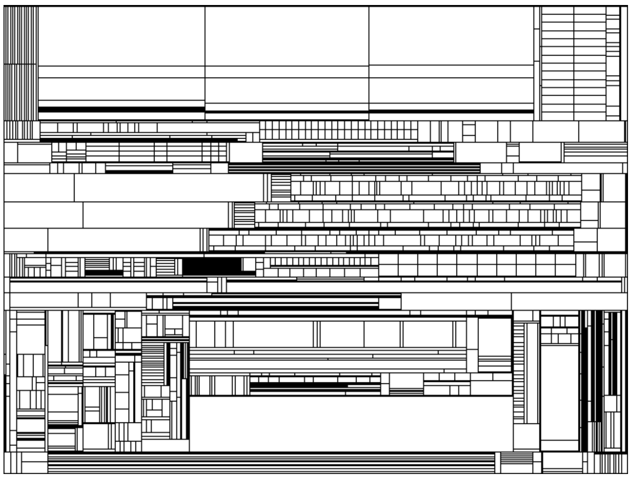


Figure 2. Treemap of file system

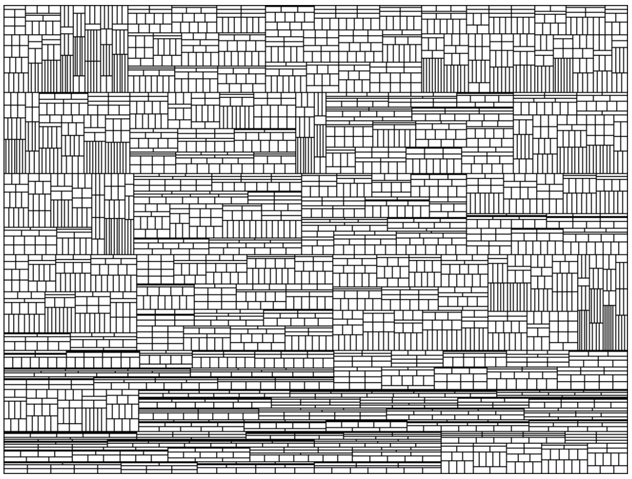


Figure 3. Treemap of organization

The full display space is used to visualize the contents of the tree. Here we present an overview of the concept, an in depth treatment is given in the original references. Figure 1 shows an example.

Each node (as shown in the node and link diagram) has a name (a letter) and an associated size (a number). The size of leaves may represent for instance the size of individual files, the size of non-leave nodes is the sum of the sizes of its children. The treemap is constructed via recursive subdivision of the initial rectangle. The size of each sub-rectangle is proportional to the size of the node. The direction of subdivision alternates per level: first horizontally, next vertically, etcetera. As a result, the initial rectangle is partitioned into smaller rectangles, such that the size of each rectangle reflects the size of the leaf. The structure of the tree is also reflected in the treemap, as a result of its construction. Color and annotation can be used to give extra information about the leaves.

Treemaps are very effective when size is the most important feature to be displayed. Figure 2 shows an overview of a file system: 1400 files are shown and one can effortlessly determine the largest ones. Labels are not shown here, but can be shown interactively by pointing at the areas of interest.

However, treemaps have limitations [4]. The problem addressed here is that treemaps often fall short to visualize the structure of the tree. The worst case is a balanced tree, where each parent has the same number of children and each leaf has the same size. The tree-map degenerates here into a regular grid. Indeed, leaves that are close in the tree are also close on the screen, but the reverse is not always true. As an example, figure 3 shows an (artificial) organization chart, modeled after the structure of our university. The university has seven faculties, subdivided into departments, which in turn are divided into sections. Each section is divided into units. Each unit contains four different types of staff members (full, associate, and assistant professor, PhD-student). The final 3060 rectangles denote individual employees. Questions such as ”What is the largest section?” or ”Is the division into units balanced?” are hard to answer from such an image.

Nested treemaps [8] are a partial remedy. During the subdivision process instead of the initial rectangle a slightly smaller rectangle is used, such that each group of siblings is enclosed by a margin. However, this consumes screen space and the visual interpretation, especially for deeply nested trees, requires effort from the viewer. The variation of properties of the surrounding lines is another option. However, the number of steps in linewidth or intensity that can be discerned without effort is small, and also here the user is required to trace lines in a maze-like image. Coloring the rectangles would not work either. Color does not provide a natural hierarchical structure, and furthermore, we want to use color to show other attributes of the elements.

One alternative has not been exploited yet: The use of shading to visualize the structure. In the remainder of this paper we will show how this can be done.

3 Cushion treemaps

3.1 Method

How can we use shading to show the tree structure? The human visual system is trained to interpret variations in shade as illuminated surfaces [6]. Hence, we can answer the question by constructing a surface which shape encodes the tree structure.

We introduce our solution with a simple one-dimensional example: binary subdivision of an interval. First, we subdivide the interval and add a bump to each of the two sub-

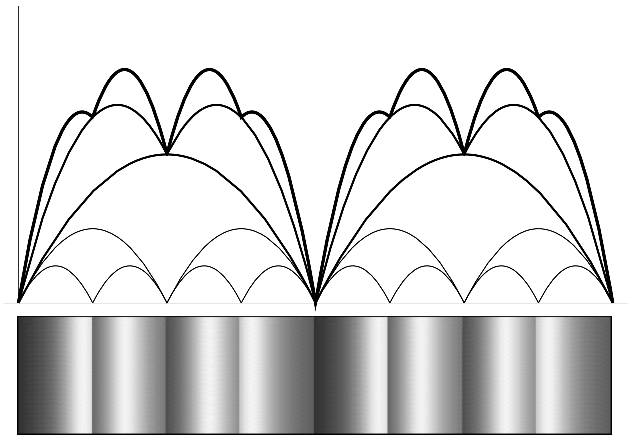


Figure 4. Binary subdivision of interval

intervals. Next, we repeat this step recursively. To each new sub-interval we add a bump again, with the same shape but half of the size of the previous one. If we do this for three levels, the results are eight segments and the top-most curve in figure 4. If we interpret this curve as a side view of a bent strip, and render it as viewed from above, the bumps transform into a sequence of ridges. The separate segments are clearly visible, each is bounded by the sharp discontinuities in the shading. Furthermore, also the binary tree structure is clearly visible, because the depth of the valleys between the segments is proportional to the distance between segments in the tree.

We can generalize this idea to the two-dimensional case. Suppose that the x-axis is horizontal, the y-axis is vertical, and that the z-axis points towards the viewer. If we subdivide the rectangle in the x-direction, we add ridges aligned with the y-direction, and vice versa for subdivision in the y-direction. As a result, cushions are generated: The summation of orthogonal ridges gives a cushion-like shape. Numerically, the simplest bump that can be used is a parabola, hence for each rectangle of the treemap we use a segment of a parabolic surface. The height z of such a surface is given by

$$z(x, y) = ax^2 + by^2 + cx + dy + e. \tag{1}$$

Initially, the surface is flat: all coefficients are zero. Consider now a new rectangle which results from subdivision along the x-axis. The ridge Δz we add is:

$$\Delta z(x, y) = \frac{4h}{x_2 - x_1}(x - x_1)(x_2 - x), \tag{2}$$

where x_1 and x_2 are the bounds of the rectangle in the x-direction. The height of this ridge is 0 for $x = x_1$ and $x = x_2$, and equal to $(x_2-x_1)h$ in the center $(x_1+x_2)/2$. The parameter h denotes the height proportional to the width, hence it controls only the shape of the ridge. The ridge Δz in (2) does not depend on y, the bump has the same shape at each cross section $y = C$. Subdivision along the y-axis is done similarly, here the ridge Δz that is added is:

$$\Delta z(x, y) = \frac{4h}{y_2 - y_1}(y - y_1)(y_2 - y). \tag{3}$$

The same value for h for each level of the tree gives a self-similar surface. A decreasing value for h is useful to emphasize the global structure of the tree. A convenient solution is to use:

$$h_i = f^i h \tag{4}$$

where h_i is the actual value of h at level i, and f a scale factor between 0 and 1.

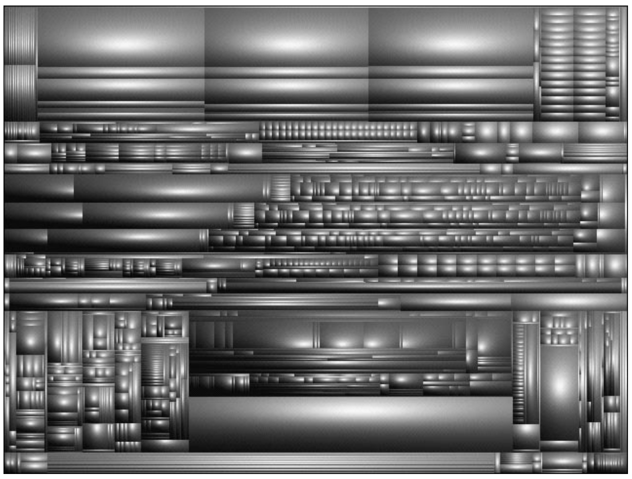


Figure 5. Cushion treemap, h = 0.5, f = 1

For the shading of the geometry a simple model, i.e. diffuse reflection, suffices [5]. The normal $\mathbf{n}$ follows from:

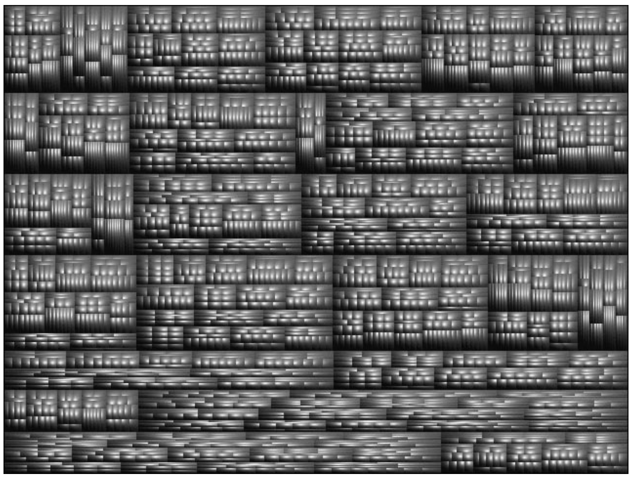
$$\begin{aligned} \mathbf{n} &= [1, 0, \partial z/\partial x] \times [0, 1, \partial z/\partial y] \\ &= [-\partial z/\partial x, -\partial z/\partial y, 1] \\ &= [-(2ax + c), -(2by + d), 1]. \end{aligned} \tag{5}$$

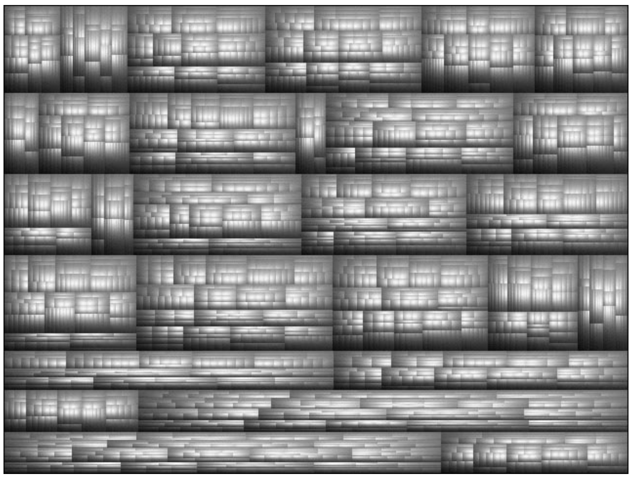
The intensity I is then given by:

$$I = I_a + I_s \max(0, \frac{\mathbf{n}\cdot\mathbf{l}}{|\mathbf{n}||\mathbf{l}|}) \tag{6}$$

where I_a is the intensity of ambient light, I_s is the intensity of a directional light source, and $\mathbf{l}$ is a vector that points towards this light source.

Results of this method are shown in figure 5 and figure 6: a cushion treemap of the file system, and three cushion treemaps of the organization, with different values for the scale factor f. All images have a resolution of 640×480 pixels. If we compare these with the treemap versions, it is clear that the shading provides a strong cue for the hierarchical structure: substructures can be identified effortlessly. With the scale factor f a continuous trade off between the visualization of global and detailed information can be made.


$h = 0.5,\ f = 1$


$h = 0.5,\ f = 0.75$

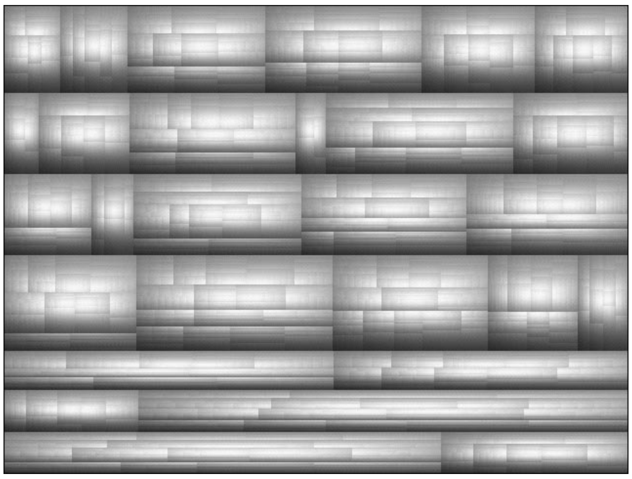

$h = 0.5,\ f = 0.5$

Figure 6. Cushion treemaps of organization

3.2 Algorithm

With the ingredients supplied in the previous section, the complete algorithm can be derived. We present it here in full detail, not because of its complexity, but to show its simplicity. First, we define a few data types. The directions X and Y are encoded via array-indexing to enable a compact algorithm. The surface is described by its linear and quadratic coefficients for the X- and Y-direction. For a Surface s the coefficients a, b, c, and d of equation (1) correspond to s[X,2], s[Y,2], s[X,1], and s[Y,1] respectively. The constant coefficient e can be ignored here, because it is not used in the shading calculation.

Each tree record has an associated size, and a pointer to its parent, to its first child, and to the next sibling.

```
type Dir = (X,Y);
    Bound = (Min, Max);
    Degree = (1, 2);
    Rectangle = array[Dir, Bound] of real;
    Surface = array[Dir, Degree] of real;
    TreeRecord = record
                     real size;
                     Tree parent, child, next;
                   end;
    Tree = pointer to TreeRecord;
```

The procedure CTM generates the cushion treemap recursively, following the same lines as the original treemap algorithm. The main extension is that during the generation of the rectangles the surface s is constructed. The surface is bent in the direction d. If the tree is a leaf, the cushion is rendered, else the direction is changed and its children are visited. The height h is updated according to equation (4).

```
procedure CTM(Tree t; Rectangle r; real h, f; Dir d; Surface s)
var Tree tc; real w;
begin
  if t.parent ≠ nil then
    AddRidge(r[d, Min], r[d, Max], h, s[d,1], s[d,2]);

  if t.child = nil then RenderCushion(r, s)
  else
  begin
    if d = X then d := Y else d := X;
    w := (r[d, Max] – r[d, Min])/t.size;
    tc := t.child;
    while tc ≠ nil do
    begin
      r[d, Max] := r[d, Min] + w*tc.size;
      CTM(tc, r, h*f, f, d, s);
      r[d, Min] := r[d, Max];
      tc := tc.next
    end
  end
end{CTM};
```

The main input for CTM consists of the root of the tree to be rendered, the initial rectangle to be used, and settings for h and f. A simple driver routine:

```
procedure MakeCushionTreeMap(Tree root; integer width, height)
var Rectangle r; Surface s;
begin
  r[X, Min] := 0; r[X, Max] := width;
  r[Y, Min] := 0; r[Y, Max] := height;
  s[X, 1] := 0; s[X, 2] := 0;
  s[Y, 1] := 0; s[Y, 2] := 0;
  CTM(root, r, 0.5, 0.75, X, s)
end; {MakeCushionTreeMap}
```

The procedure AddRidge takes care of the update of the coefficients of the parabolic surface according to equation (2) and (3):

```
procedure AddRidge(real x1, x2, h; var real s1, s2)
begin
  s1 := s1 + 4*h*(x2+x1)/(x2–x1);
  s2 := s2 – 4*h/(x2–x1)
end;{AddRidge}
```

If in CTM a leaf node has to be rendered, RenderCushion is called. The rectangle, defined in continuous space, is scan converted via sampling the centers of the pixels that fall within the rectangle. The intensity of a pixel is calculated according to equation (5) and (6). A straightforward implementation is given below. For clarity, no optimization is applied here. Early outs (if the rectangle does not cover the center of any pixel), a priori calculation of common sub-expressions, removal of constant sub-expressions out of the loops and incremental evaluation of the quadratic expression can easily be added. Other extensions are the rendering of lines that separate the rectangles and the use of color to visualize some property of the leaf. The fixed settings used here for the light source give good results. The frontal light is slightly offset to the right and to above: $\mathbf{l} = [1, 2, 10]$.

```
procedure RenderCushion(Rectangle r; Surface s)
const Ia = 40;
      Is = 215;
      Lx = 0.09759; Ly = 0.19518; Lz = 0.9759;
var   integer ix, iy; real nx, ny, cosa;
begin
  for iy := trunc(r[Y, Min]+0.5) to trunc(r[Y, Max]–0.5) do
  for ix := trunc(r[X, Min]+0.5) to trunc(r[X, Max]–0.5) do
  begin
    nx := –(2*s[X,2]*(ix+0.5) + s[X,1]);
    ny := –(2*s[Y,2]*(iy+0.5) + s[Y,1]);
    cosa := (nx*Lx + ny*Ly + Lz)/ sqrt(nx*nx + ny*ny + 1.0);
    SetPixel(ix, iy, Ia + max(0, Is*cosa))
  end
end;{RenderCushion}
```

This concludes the presentation of the complete algorithm. It shows that cushion treemaps can be easily implemented in a compact way.

4 Interaction

Presentation of hierarchical structures is only one aspect, for effective visualization of such structures interaction is equally important. We have embedded cushion treemaps in SEQUOIAVIEW, our interactive system for the analysis and visualization of large tree structures. The cushion treemaps are generated with a slightly extended version of the previous algorithm. This takes less than a second on an SGI O2, even for larger images and trees. Upon each interaction the image is copied to the screen, annotation is overlaid, selected rectangles are colored by superimposing transparent rectangles. Various coloring options are available, to show the size, the level, and other attributes of the leaves. Regeneration of the image is done only if the tree or one of the image parameters change.

Many options are provided for navigation and selection. The user can click on rectangles, upon which the properties of the leaf are displayed in a separate window. The current node is highlighted with a red outline. The arrow keys can be used to select siblings (left, right), the parent (up) and the first child (down). The selected elements are continuously updated and highlighted, which enables fast and accurate navigation. Elements within a user-defined size range can be selected, and elements can be selected by matching their name with regular expressions. The user can zoom in on sub-trees, and zoom out again.

SEQUOIAVIEW has been used for various applications: file systems, tries, organization charts, lexical parse trees, and software structures. The cushion treemaps, supported by multiple options for navigation and selection, turned out to be highly effective.

5 Discussion

Both simpler and more complex variants of the cushion treemap presented here are conceivable. We consider both here to show that the version presented is optimal, in the sense that these variants are no improvements.

We have chosen to use a geometric model, which is shown as a shaded surface viewed from above. We could also have used a direct, 2D model for the shading. This could lead to a more efficient algorithm, without the normalization of the normal per pixel. For instance, the value of $z(x, y)$ as defined before could directly denote an intensity. However, such a simple model does not work satisfactorily. The result is that each rectangle is filled with an ellipse-shaped spot. More sophisticated 2D models could work, but their control is cumbersome, when compared to the intuitive model with two parameters (h, f) presented here. The use of a geometrically based shading leads automatically to an image that is easy to interpret.

Another option is to view the surface from an oblique angle as a 3D surface. However, 3D views do not pay off. The 3D view is more expensive to generate, the height itself does not provide a direct cue on the structure, and the only view where all rectangles can be viewed simultaneously is the top view.

Rather, we will spend effort in other areas. Some open questions are:

– How can anti-aliasing be provided, to handle rectangles that fall between pixels?
– How can we effectively present multi-dimensional attributes per leaf?
– How can graph-information (e.g. symbolic links in file systems) be included?
– Does the combination of this representation with other types of presentations pay off?
– Can the presentation of size be improved, such that perceptual characteristics are taken into account?

Summarizing, still much work has to be done in the area of tree visualization.

6 Conclusions

We have presented a new method to visualize hierarchical information. Cushion treemaps inherit the elegance of standard treemaps, and add intuitive shading to provide insight in the hierarchical structure. Their features can be summarized as follows:

– *efficient*: generation of an image typically takes less than a second;
– *effective*: the structure is visualized much more effective compared with standard treemaps. This is obvious if we compare for instance figure 3 with figure 6;
– *compact*: the display area is used very efficiently, more than 3000 leaves can be displayed easily in an image with 640$\times$480 resolution. As a result, no scrolling or opening/closing of nodes is needed to view the whole structure;
– *easy to implement*: the complete algorithm fits on one page;
– *easy to control*: the appearance can be controlled with a few intuitive parameters, for which default values often suffice.

Finally, cushion treemaps address one of the most important topics in visualization. Their wide applicability is probably what has struck us most during our research.

References

[1] A. Bruggemann-Klein and D. Wood. Drawing trees nicely with tex. *Electronic Publishing*, 2(2):101–115, 1989.

[2] S.K. Card, G.G. Robertson, and J.D. Mackinlay. The information visualizer, an information workspace. In *Proc. of ACM CHI'91, Conference on Human Factors in Computing Systems*, pages 181–188, 1991.

[3] R. Chimera, K. Wolman, and B. Shneiderman. Evaluation of three interfaces for browsing hierarchical tables of contents. Technical Report Technical Report CAR-TR-539, CS-TR-2620, University of Maryland, February 1991.

[4] S.G. Eick. Visualization and interaction techniques. In *CHI97 Tutorial notes on Information Visualization*. ACM SIGCHI, March 1997.

[5] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics – Principles and practice, 2nd edition*. Addison-Wesley, Reading, MA, 1996.

[6] W. T. Freeman, E. H. Adelson, and A. P. Pentland. Shape-from-shading Analysis with Shadelets and Bumplets. In *Inv. Opth. and Vis. Sci. (supp), 31*, page 410. Association for Research in Vision and Ophthalmology, Spring 1990.

[7] G.W. Furnas. Generalized fisheye views. In *Proc. of ACM CHI'86, Conference on Human Factors in computing systems*, pages 16–23, 1986.

[8] B. Johnson and B. Shneiderman. Treemaps: a space-filling approach to the visualization of hierarchical information structures. In *Proc. of the 2nd International IEEE Visualization Conference*, pages 284–291, October 1991.

[9] D.E. Knuth. *Fundamental algorithms, art of computer programming*, volume 1. Addison-Wesley, Reading, MA, 1973.

[10] G.G. Robertson, J.D. Mackinlay, and S.K. Card. Cone trees: animated 3d visualizations of hierarchical information. In *Proc. of ACM CHI'91, Conference on Human Factors in Computing Systems*, pages 189–194, 1991.

[11] B. Shneiderman. Tree visualization with tree-maps: a 2d space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, September 1990.