

Chapter 2

Visualization Principles and Techniques

The previous chapter described many monitoring systems' execution models and methods of extracting behavior information from programs that they monitor. These "hard" tasks are the primary subject of this book, but observing behavior is pointless unless information is presented to the user in a way that meets his or her needs. Visualization is the art of presenting large amounts of information in accessible graphical form. Appropriate graphic representations allow the rapid delivery of vast amounts of information that would overwhelm the user if presented in textual form. This chapter introduces the principles that underline effective visualizations, suggests an array of visualization techniques, and presents an incremental methodology by which such tools may be developed.

It is worth noting that there is more than one kind of visualization. *Scientific visualization* is the graphic rendering of gigantic n -dimensional data sets by various means of projection and abstraction. In contrast, *software visualization* is the depiction of software artifacts such as directories, user data, or system log files.

Program visualization as described in this book is a subfield of software visualization focused on the dynamic behavior of programs themselves, rather than the data they manipulate. The preceding chapter did not present broad coverage of related work in the general area of software visualization, since it has been described admirably elsewhere [1].

Compared with scientific visualization, program visualization is more abstract, since program behavior above the hardware level does not map easily onto real-world geometries. Program visualization evolved from the hand-written diagrams and notations used by programmers and computer scientists to describe their structures prior to the advent of automated visual tools.

Before there was visualization, there was graphic design. Visualization emerged as a subdiscipline of graphic design when computer screens began to replace printed paper. Visualization includes graphic design, but has additional constraints imposed by hardware and software capabilities and user requirements.

2.1 Graphic Design Principles

It is not worth implementing elaborate computer graphics if the graphic design does not convey information clearly. Some principles of graphic design are self-evident, such as abstracting away irrelevant detail; other principles are learned through experience. Some of the best references on graphic design are by Tufte [2–4] and Bertin [5]. Tufte’s observations concerning graphic excellence are summed up by the following:

Graphical excellence is that which gives to the viewer the greatest number of ideas in the shortest time with the least ink in the smallest space.

To achieve such excellence in designs, Tufte advocates five principles:

- if you do nothing else, at least show the information
- show as much as you can with as little ink as possible
- remove ink that isn’t showing useful information
- remove redundant information
- revise and edit

The reader may consult Tufte’s work for numerous examples of these principles in practice.

2.1.1 *Show the Information*

If a graphic fails to deliver the required information, it is worse than no graphic. While this mistake might not sound likely, it is not so uncommon for a visualization to present only part of the required information. Depending on the context this can be misleading, or it can be disastrous. Omitting critical information may be as serious as distorting it, a violation of this principle that takes place in myriad forms [6].

Perhaps the most common cause of lost information is when some other information is mapped onto the same location and obscures an item of interest. The viewer may not even know that something is covered up. The situation is not much better if the information is presented but the user cannot decipher it. Visualizations should be self-explanatory. Axes and units, both for geometry and the temporal interpretation of animations, should be clearly explained. On a print graphic, labels and legends might pose an obstacle themselves, but on a computer display they can be toggled on when needed and disappear just as easily.

2.1.2 *Maximize Information Density*

A high information density allows more information in the available space, or a given amount of information in a smaller space. Tufte gives published examples that range over between two and three orders of magnitude from the least dense to

the most dense! Many computer visualization tools are produced by computer scientists who are experts in fields such as architecture or parallel algorithms and do not bother to study graphic design before they publish their visualization work; not surprisingly, some of these tools achieve monumentally low information density.

Information density is increased by eliminating empty space by either adding more information or shrinking the display. Only the actual information depicted counts toward high density, not labels, legends, axes, arrows, or notes present. If you shrink your display not to the point of unreadability, but to the minimal size it requires, you will make room to run multiple visualization tools side by side. The architecture described in this book supports a large number of simultaneous visualization tools.

High information density comes at a price. Humans have a minimal resolution of perception. Humans can see individual pixels on modern displays, but to perceive discrete objects animated on a screen, the minimal comfortable size may vary, depending on the resolution and size of the display. Don't rely too heavily on an individual pixel to make an important point.

Information density should be increased in a way that keeps the graphic organized. A dense, complex graphic may be so cluttered that it is difficult to sort out. It turns out that *organized complexity* allows rapid assimilation and is aesthetic in appearance, while *disorganized complexity* is difficult for humans to handle [7].

2.1.3 Remove Useless Information

Many or most visualization tasks have more information available than room in which to display it. Even if there is enough room, nonessential information distracts and detracts from important information. The user has only so much attention to go around. For this reason, filtering out useless information is a basic task. Of course, not all information can be designated as useless or useful; information occupies a range of utility, especially in visualization situations in which the users are not sure what they are looking for. A corollary of the principle of removing useless information is that the emphasis (and space) allocated to information should be proportional to its usefulness.

Consider drawing common tree structures where many nodes have parent-child relationships, and each node contains its own information. Figure 2.1 shows such a tree using a classically attractive layout algorithm due to Moen [8]. This layout is

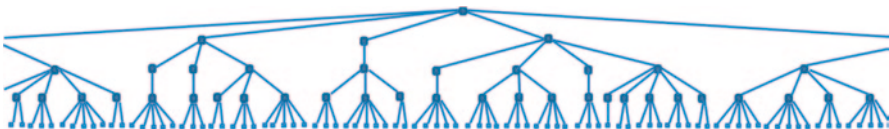
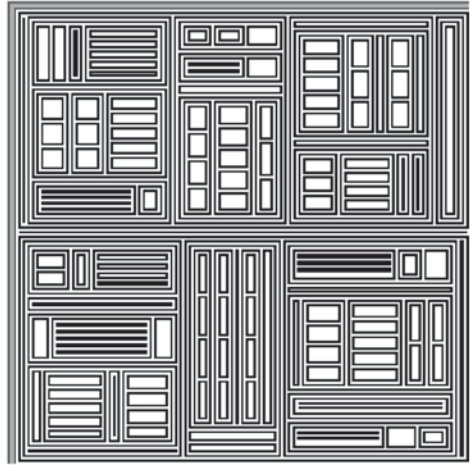


Fig. 2.1 A classical tree layout

Fig. 2.2 A tree-map emphasizes nodes instead of edges



fine if the user is mainly interested in the size and shape of the tree structure itself; almost all screen space is devoted to structure.

In many applications, however, the user may be more interested in the information presented at the nodes; the tree structure may be a side issue. In this case, a layout such as the tree-map [9] shown in Fig. 2.2 may be more relevant, even though its portrayal of the tree structure is less clear. The layout you select for drawing trees should depend on what is important to the user at that particular moment.

2.1.4 Remove Redundant Information

In user-interface design, it is good for users to have many different ways to accomplish whatever they are trying to do. It is similarly good to have several different ways of viewing data—one way at a time. Providing several simultaneous views of the same data is wasting scarce pixels and resolution that can be better applied to providing a single, richer view. This is even more true for visualization than for ordinary graphics, since visualization typically involves abstracting away a richer set of data than can be portrayed in detail. Useless information is bad, and so is duplicated information.

2.1.5 Iterate

It is naive to expect that your first approach to visualizing something will be optimal. The graphic design will evolve. These refining iterations are costly and time consuming for printed graphics, and they can be equally difficult for visualization programming. Using a language and graphics toolkit that accommodates this principle is common sense.

2.2 Visualization Principles

Most visualization efforts can start by adapting a well-known technique from printed graphics. Where graphic designers say *ink* one may generally substitute *pixel writes* to the computer's display. A typical bitmapped computer graphics display presents a million or more bytes of information to the user at a time. The raw number of pixels and colors available defines the capability of the display to render static images analogous to print graphics.

Static computer images can be evaluated in terms of graphic design principles such as information density. Most visualizations, however, are dynamic; the graphics hardware and software define limits on the rate at which the display may change. In this context, information density is a three-dimensional measure that includes the time interval during which the changing display is viewed.

Visualization of dynamic execution behavior is different from visualizing a static data set in several ways. These differences motivate the techniques presented in the rest of this book. They may be summarized in the following basic concepts:

- animation
- metaphors
- interconnection
- interaction
- dynamic scale
- static backdrop

2.2.1 Animation

The ability to depict temporal relationships by animating dynamic behavior is a crucial tool. There are tradeoffs between visual sophistication and the associated computational cost and programming time required. Widely applicable techniques are ones that can be animated on low-cost hardware.

Thanks to the computer games market, our definition of low-cost hardware now includes support for animating 3D scenes of moderate complexity. While ubiquitous, many specific workstations and operating systems do not support 3D graphics, so they are not universally available. The hardware is not really the main problem.

The main obstacle to widespread development of animated 3D visualizations is inadequate software. OpenGL, the closest thing to a universal standard, is a low-level specification more oriented towards hardware capabilities than ease of programming. Higher-level languages and toolkits for 3D programming are nonportable, expensive, or both.

In order to achieve universally available, easily programmed animations, this book focuses on simple 2D graphic designs. The output of monitoring could be piped into an existing visualization package such as IDL or Khoros for more sophisticated graphics rendering including 3D views, but this would reduce the degree of interactivity and control provided by the tools.

2.2.2 *Least Astonishment*

Visualizations should obey the principle of least astonishment. This is important in print graphics, but it is even more important in animated visualizations where the display is changing and the users cannot continuously study each image at their leisure. When possible, visualizations should present information in a manner with which the users are already familiar.

Tufte observes that in the absence of a reason to do otherwise, most graphics should utilize the *golden rectangle*, with a primary horizontal axis 1.6 times wider than the vertical axis. The arguments in favor of the golden rectangle range from human psychology to alleged evolutionary skill at scanning the horizon for predators and prey. Computer displays favor the horizontal axis. Text labels are read horizontally and are understood more rapidly when written in a single line than when split onto multiple lines, again favoring a primary horizontal axis. Because most graphics depict information similarly, presentation of data will produce less astonishment when the horizontal axis represents the *cause*, and the vertical axis represents the *effect* described by the graph.

2.2.3 *Visual Metaphors*

The mapping from program information to window geometry often is artificial or unintuitive, especially when no natural geometry is inherent in the information to be presented. A familiar or readily inferred visual metaphor for the behavior being presented can lower the cognitive load imposed on the user and increase the rate of comprehension.

Although some metaphors are drawn naturally from a specific application domain or a notation in common use among programmers, others are drawn from nature or from nontechnical symbols found in daily life.

2.2.4 *Interconnection*

Understanding a complex piece of software entails an understanding of a variety of distinct behaviors and the relationships between them. For example, control flow, data structures, memory allocation behavior, and input/output all have distinct but interrelated patterns in program execution. Visualizations that consume most or all of the screen do not allow for simultaneous display of other forms of execution behavior.

Under any circumstances, we cannot hope to portray all of program execution in a single 2D or 3D graphic. And we cannot anticipate, in general, which subset of the available information a given user will need. Our emphasis is on multiple visualiza-

tion tools selected by the user, executing simultaneously, showing different aspects of program behavior.

2.2.5 Interaction

Visualizations are more effective when the user can navigate and steer them in appropriate directions. Operations such as panning and zooming are vital in order to present some information in more detail than others, because the user remains vital in prioritizing which information to emphasize. The importance of navigation increases in 3D visualizations. Depictions of 3D objects on a computer screen may be ambiguous unless those objects are seen in motion.

A graphic design used in visualization should allow for natural interactive controls, an issue not addressed in static design. For example, a user should be able to select objects and request details about them, or specify that they should be watched, and execution should pause when they are modified.

2.2.6 Dynamic Scale

The scale imposed in the depiction of dense information on a computer screen is extreme, but in addition, the scales are highly dynamic. If the scale does not change dynamically, most visualizations waste space and lose detail over most of the execution being observed. On the other hand, changing scale too frequently is both computationally expensive and disorienting.

There are scaling alternatives to redrawing the entire window to use larger or smaller units. There are several different ways to utilize a scale that varies in a single image in a consistent way. If one of these techniques is used, it must be evident to the user or it can work harm instead of help. Logarithmic scales are one option, but they are not always appropriate and typically need to be tuned to the size of the dataset involved.

A more generally useful, but treacherous dynamic scaling technique, is the fish-eye view [10]. Fisheye views introduce one or more focus points of attention; a distortion is applied to all output to the window, scaling it by some function of its distance from the focus or foci. Figure 2.3 shows a simple fisheye view with a single focus point applied to a text file. A scalable font and simple arithmetic were all that was needed in order to render this view.

Fisheye views can be applied to arbitrary graphics, and distortion functions may account for multiple focus points with varying degrees of importance [11]. Figure 2.4 shows a 2D map of a simple graph in which each node is distorted by its own factor. The nodes in this case represent solar systems, and they are annotated by orbiting planets and text labels that are all scaled proportional to the size of the star itself, which is drawn as a yellow circle. Planet details and text labels are omitted for systems drawn beneath a minimum threshold size.

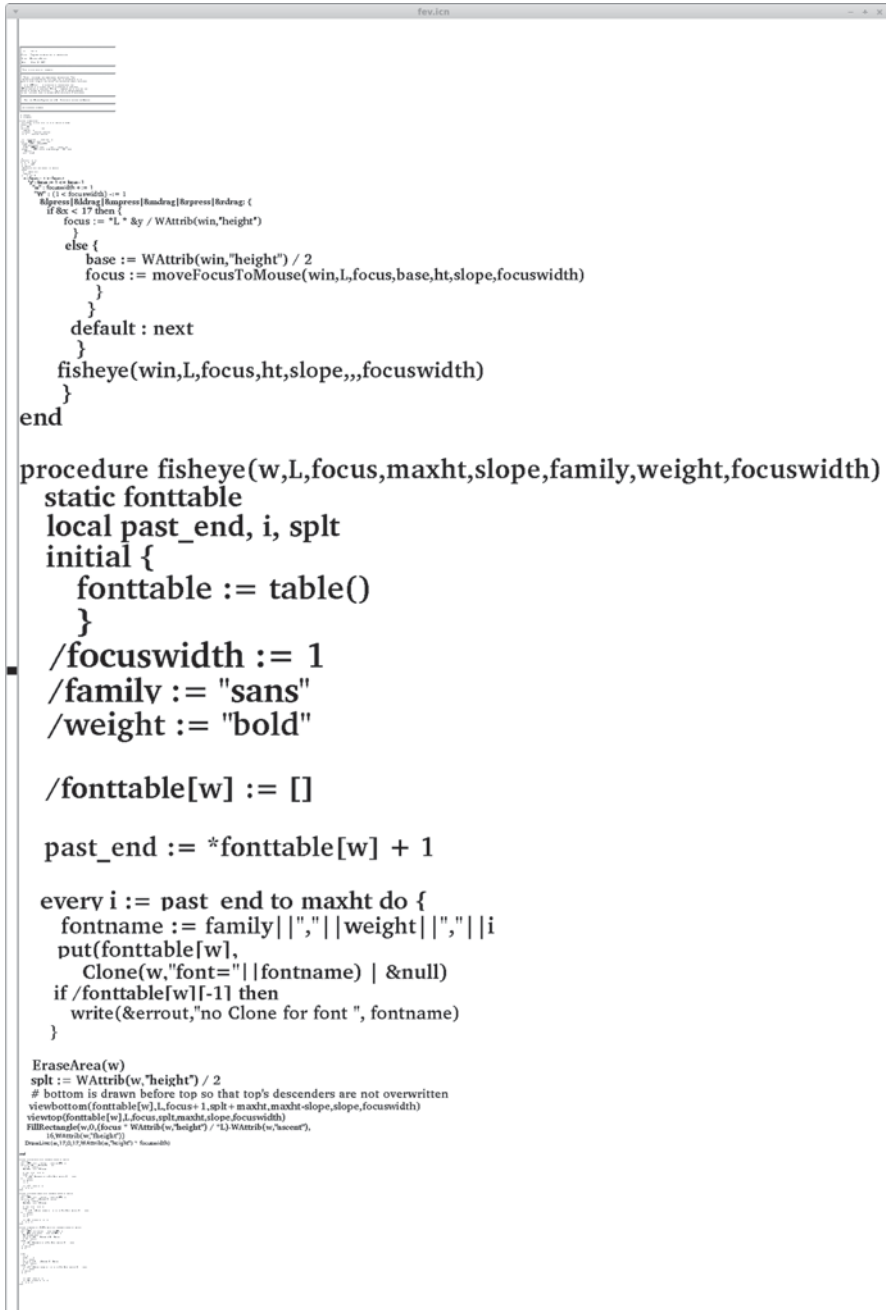


Fig. 2.3 A fisheye view with a single focus point

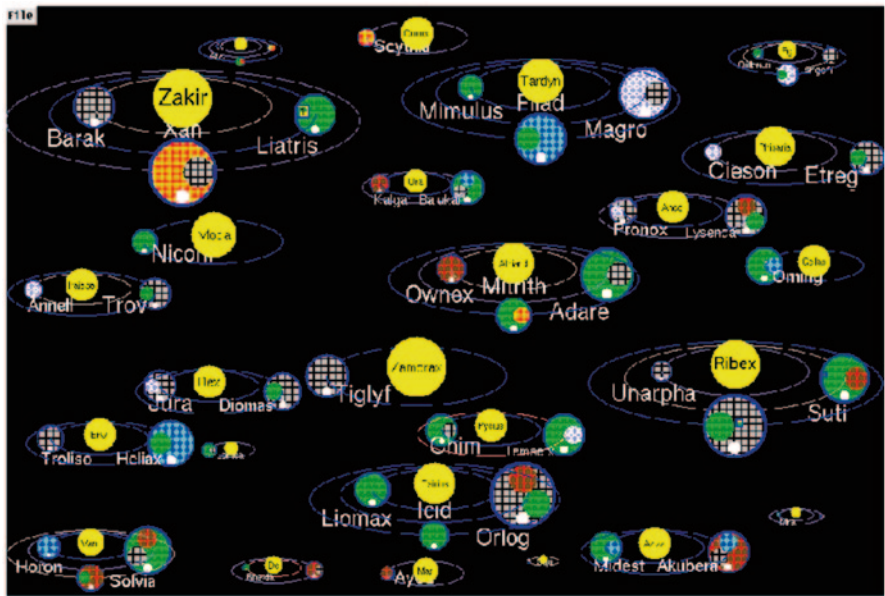


Fig. 2.4 A graphical fisheye view

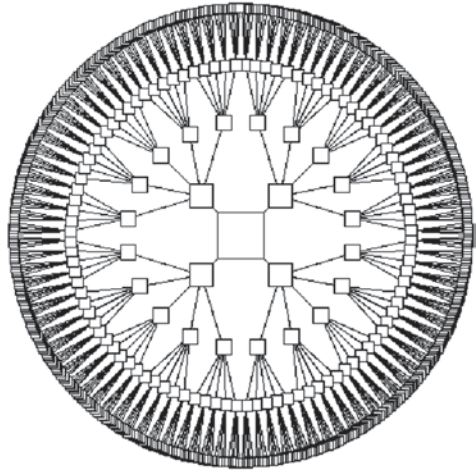
2.2.7 Static Backdrop

Dynamic analysis tools are often best interpreted when superimposed upon a context consisting of information acquired by static analysis; the static information can provide a map that programmers are familiar with. Examples of static backdrops are a program’s call graph, or even its source code.

2.3 Visualization Techniques

The visualization author faces the problem of rendering the selected graphics with an acceptable real-time performance, characterized by animation frame rate as well as interactive responsiveness and navigability. A visualization can be measured as graphics hardware is measured, in pixels or polygons per second, or the frame rate at which the screen is updated. While this may tell you whether a workstation with faster graphics would improve your visualization, it says nothing about whether the user understood the information. Humans cannot follow details very rapidly, so the optimal rate of change for the graphic display is more likely to be bounded by the human reader than by the hardware or the visualization’s graphic rendering algorithm.

Fortunately, some of the simplest graphics are effective, easy to implement, and are familiar to users. Time series graphs, bar charts, pie charts, and scatterplots are all examples of graphic designs that are easily programmed but may need adapta-

Fig. 2.5 A circular tree

tion for visualization purposes. Part III of this book includes many examples of such adaptation.

2.3.1 Incremental Algorithms

Redrawing the entire screen each time something changes is not an efficient approach. Incremental algorithms may be required in order to achieve smooth animation. An incremental algorithm is smart enough to render graphics only for the objects on screen that are affected by an operation, and not redraw the others.

It is easy to write incremental algorithms for some graphic designs, and not easy for others. The importance of speedy animation often dictates that a graphic design be selected based on the availability of an incremental algorithm. For this reason, simpler graphic designs may win out over ones that are prettier or more sophisticated.

2.3.2 Radial Coordinates

A very interesting visual effect is obtained by adopting a radial mapping in which execution sequence or time rotates around a point. Such mappings are used in a variety of visual metaphors. A radial mapping may represent similar information to that of a Cartesian mapping, but the user may recognize different patterns due to the metaphor employed. The center of the image provides a natural focus of attention for priority items such as the root of a tree. Figure 2.5 shows a circular tree with hundreds of nodes. More sophisticated radial techniques are possible with the introduction of hyperbolic geometry [12].

Writing Virtual Environments for Software Visualization

Jeffery, C.; Al-Gharaibeh, J.

2015, XVII, 155 p. 65 illus., 24 illus. in color., Hardcover

ISBN: 978-1-4614-1754-5