Computer-Human Collaboration in the Design of Graphics

A thesis presented
by

Kathleen Ryall

to
The Division of Engineering and Applied Sciences
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Computer Science

Harvard University
Cambridge, Massachusetts

August 1997

Abstract

Delineating the roles of the user and the computer in a system is a central task in user interface design. As interactive applications become more complex, it is increasingly difficult to design interface methods that deliver the full power of an application to users, while enabling them to learn and to use effectively the interface to a system. The challenge is finding a balance between user intervention and computer control within the computer-user interface.

In this thesis, we propose a new paradigm for determining this division of labor, which attempts to exploit the strengths of each collaborator, the human user and the computer system. This collaborative framework encourages the development of semi-automatic systems, through which users can explore a large number of candidate solutions, while evaluating and comparing various alternatives. Under the collaborative framework, the problem to be solved is framed as an optimization problem, which is then decomposed into local and global portions. The user is responsible for global aspects of the problem: placing the computer into different areas of the search space, and determining when an acceptable solution has been reached. The computer works at the local level, computing the local minima, displaying results to the user, and providing simple interface mechanisms to facilitate the interaction. Systems employing this approach make use of task-specific information to leverage the actions of users, performing fine-grained details while leaving the high-level aspects to the user specifiable through gross interface gestures.

We present applications of our collaborative paradigm to the design and implementation of semi-automatic systems for three tasks from the domain of graphic design: network diagram layout, parameter specification for computer graphics algorithms and floor plan segmentation. The collaborative paradigm we propose is well-suited for this domain. Systems designed under our framework support an iterative design process, an integral component of graphic design. Furthermore, the collaborative framework for computer-user interface design exploits people's expertise at incorporating aesthetic criteria and semantic information into finding an acceptable solution for a graphic design task, while harnessing a computer's computational power, to enable users to explore a large space of candidate solutions.

Acknowledgements

I guess things always come back to haunt you. I used to tease my older sister about being a professional student (she is now a pediatrician). Little did I know that I would decide to pursue a PhD, and that it would take me seven years to earn it! After a mere twenty-five years of continually being a student, I'm now ready to face the "real world".

This thesis would not have been possible without the members of my thesis committee. I must start by thanking my advisor Stuart Shieber for his support and guidance during my time as a graduate student. He encouraged me to find a topic that I really loved and gave me the time and space to find it. Barbara Grosz has also been incredibly supportive through the years. I am especially grateful to her for the AI research group that provided a forum for me to present my ideas and receive valuable feedback. Special thanks to Joe Marks for his time and efforts towards my research and professional development and for originally sparking my interest in graphics and interfaces. This material is based upon work supported in part by the National Science Foundation under Grant Nos. IRI-9157996, IRI-9350192, and IRI-9618848, and in part by MERL — A Mitsubishi Electric Research Laboratory.

I would also like like to thank other members of the Harvard faculty: Harry Lewis for being a great teaching role model, Tom Cheatham for being on my qualifying committee, and Margo Seltzer for her advice.

Many of my fellow graduate students (past and present) and other members of the Harvard community have made my time here easier and certainly more enjoyable: Nadia Shalaby (my first real friend in the department, my first (and only) officemate, and a person with more patience than I deserve), Azer Bestavros (for letting me know that there was a CS department at Harvard), Cecile Balkanski and Karen Lochbaum (for help with my qualifying exam), Ted Nesson (a great lunch partner), the pasta night crew (Andy Kehler, Jon Christensen, Stan Chen, Josh Goodman, Rebecca Hwa, Wheeler Ruml), Lillian Lee (my thesis and job cohort), and other Aiken (may it rest in peace) inhabitants (Ellie Baker, Keith Smith, Chris Small, Karen Daniels). Thanks also to non-CS types Erik Evensen (an awesome housemate and co-Fellow), Margaret Hsu (down the block in Chemistry), and Susan Zawalich (house administrator extraordinaire).

Contrary to popular belief, there is life outside Harvard. The following friends have helped me to keep my sanity by providing dinner, procrastination opportunities (often in interesting places), and a sympathetic ear: Stu Berman (my first TA back at Wes), Sandy Cross (on to South America?), Heidi Webster, Meg Derr, Liz Echausse, Jessica Rose, and Ryan Pence. Special thanks to Barry Jaspan for being there and believing in me (even when I didn't), for showing me the value of persistence and perseverance, for letting me lead sometimes, and for knowing when to break out the turkey and mashed potatoes. ;-)

Finally, deepest thanks to my family who have always believed in me, and challenged

me to do more: Johnny, Tommy and Amie, and Moo and Lou (hopefully you won't have to help me move any more). Extra special thanks to my parents, John and Nancy, who have always said (and somehow convinced me) that I could do anything I wanted to do.

Contents

1 Introduction **1**
- 1.1 Designing Graphics . . . 1
- 1.2 Traditional Approach . . . 2
- 1.3 Collaborative Framework . . . 3
- 1.4 Problem Domains . . . 6

2 Background **8**
- 2.1 What Is An Interface? . . . 8
- 2.2 The Design of Graphics . . . 10
  - 2.2.1 Informational Graphics . . . 10
  - 2.2.2 Representational Graphics . . . 11
- 2.3 Summary . . . 14

3 GLIDE **15**
- 3.1 Introduction . . . 15
- 3.2 Our Approach . . . 16
- 3.3 Example Interaction . . . 19
- 3.4 System Design . . . 22
  - 3.4.1 Constraint Formulation . . . 23
  - 3.4.2 Constraint Satisfaction . . . 26
  - 3.4.3 Visual Presentational Issues . . . 27
  - 3.4.4 Implementation . . . 28
- 3.5 Comparison to Existing Approaches . . . 29
  - 3.5.1 Drawing Packages . . . 29
  - 3.5.2 Graph Drawing Algorithms . . . 30
  - 3.5.3 Constraint-Based Editors . . . 31
- 3.6 Summary . . . 33

4 Design Galleries **34**
- 4.1 Introduction . . . 34
- 4.2 Design Gallery Methodology . . . 36
- 4.3 Sample Application: Medical Imaging . . . 37
- 4.4 Arrangement and System Design . . . 40
  - 4.4.1 Icons . . . 40
  - 4.4.2 Layout . . . 42
  - 4.4.3 Navigation . . . 43
  - 4.4.4 Interaction . . . 44

4.4.5 Application-Specific Functionality . . . 44
4.5 Comparison to Existing Approaches . . . 44
4.5.1 Parameter Specification . . . 45
4.5.2 Information Visualization and Navigation . . . 47
4.6 Summary . . . 49

5 Floor Plan Segmentation **50**
5.1 Introduction . . . 50
5.2 Our Approach . . . 51
5.2.1 Proximity Field . . . 52
5.2.2 Region Definitions . . . 54
5.2.3 Weaknesses . . . 55
5.3 System Design . . . 56
5.4 Example Interaction . . . 58
5.5 Previous Work . . . 60
5.5.1 Systems . . . 62
5.5.2 Vision and Image Processing . . . 64
5.5.3 Document Analysis . . . 65
5.5.4 Drawing Tool Techniques . . . 66
5.6 Summary . . . 68

6 Conclusion **69**

A GLIDE: Additional Example Diagrams **71**
A.1 Exploring Design Alternatives . . . 71
A.2 Understanding and Organizing an Unknown Data Set . . . 76

B Design Galleries: Example Application Areas **82**
B.1 Scientific Visualization . . . 82
B.2 Two Dimensional Animation . . . 82
B.3 Three Dimensional Animation . . . 84
B.4 Particle Systems . . . 85

C Floor Plan Segmentation: Additional Example Diagrams **88**

List of Figures

1.1 Traditional Master-Slave Framework. 4
1.2 New Collaborative Framework. 4

3.1 Visual Organization Features. 17
3.2 A Local Minimum. 18
3.3 Scanned diagram on which the example was based. 19
3.4 Interface to the GLIDE system. 20
3.5 User adds an *Alignment* VOF to each row. 20
3.6 User adds three more VOFs. 21
3.7 User adds three text labels, and three VOFs. 22
3.8 The finished layout. 22
3.9 Reduction of some sample VOFs to systems of generalized springs. 24

4.1 Components and interactions of a Design Gallery Application. 36
4.2 Human hip data set rendered using different opacity transfer functions. . . 38
4.3 Instantiated components for a medical imaging Design Gallery application. 39
4.4 A Design Gallery for medical imaging. 39
4.5 A schematic of the Design Gallery user interface. 41

5.1 How dogs interpret floor plans. 51
5.2 Motivation for the proximity field. 53
5.3 Computing the nearest local minimum. 55
5.4 Computer-user interface for the system. 57
5.5 The original scanned bitmap. 59
5.6 The false-colored proximity field generated for the bitmap. 59
5.7 The regions delineated according to the proximity map in Figure 5.6. 60
5.8 The user adds two false walls. 61
5.9 The user joins the door swings to the upper hall region. 61
5.10 The final region definitions. 62
5.11 Regions delineated by two methods. 67

A.1 Starting point for exploring a design alternative. 71
A.2 User adds a *T-Shape* VOF to four nodes. 72
A.3 User adds a second *T-Shape* VOF to the left three nodes. 72
A.4 User manipulates the root node of each *T-Shape* VOF. 73
A.5 The intermediary layout with the *T-Shape* VOFs in the right places. 73
A.6 User intervention aids global optimization. 74
A.7 User adds two more VOFS: *Vertical Alignment* and *Vertical Symmetry* . . . 74

A.8 User adds two *Horizontal Alignment* VOF. 74
A.9 User adds a *Vertical Even Spacing* VOF. 75
A.10 The final layout. 75
A.11 A random data set with random initial layout. 77
A.12 A first attempt by a novice user. 78
A.13 A second layout by a novice user. 79
A.14 A sink-to-source layout by an expert user. 80
A.15 A symmetry-based layout by an expert user. 81

B.1 A Design Gallery with different opacity and color transfer functions. 83
B.2 Pop-up display depicting opacity and color transfer functions. 83
B.3 A two-dimensional double pendulum. 84
B.4 A Design Gallery for an actuated two-dimensional double pendulum. 85
B.5 HopperDog: A three-dimensional articulated dog. 86
B.6 Design Gallery for HopperDog. 86
B.7 A Design Gallery for a particle system. 87

C.1 MIT floor plan. 89
C.2 Berkeley floor plan. 89

Chapter 1

Introduction

In this thesis, we describe a new interface design paradigm for determining the division of labor between people and computers, one which attempts to exploit the strengths of each collaborator. We apply this approach to three distinct problems in the design of graphics, presenting novel implemented systems for each problem.

1.1 Designing Graphics

Graphics are a rich medium available to aid both computers and people in their communication goals. A picture is worth the proverbial thousand words. As we move through the Age of Information and into information overload the ability to organize and present information graphically will be more important than ever. Moreover, as electronic presentation of information becomes more prevalent, people will require more support in creating and refining graphics on-line. Together, computers and users can often find solutions to problems that neither could have discovered alone. The question is how to strike a balance between user intervention and computer control.

As a result of computer support for the design and refinement of graphics, graphics are becoming a more expressive and interactive medium. In turn, the number of possible graphics, which we will call the *design space*, grows exponentially with the number of parameters in the design. The complexity of the resulting design space increases as well, making it difficult to explore design alternatives and thus to generate desirable graphics. Many argue that the complexity of the design space is the root of the problem. While the design space is indeed large and rich, this is not the inherent problem. The difficulty comes in relying on the computer to navigate and explore the space, with little or no assistance from a person. Problems arise from both the user and computer perspectives.

From the user perspective, people often do not know what they want in a graphic. Although some graphics are better than others, it is often difficult to formalize (in a precise manner) what is desired in a graphic. We refer to this as the problem of *unquantifiable output characteristics*; it is the intangible qualities that lead people to prefer one display to another. Next, even when people can enumerate desired characteristics, it is still difficult for them to communicate this information to the computer. Communication is typically done through an objective function (a mathematical specification of a desired graphic) or through parameter setting. Neither of these methods is particularly convenient for people.

From the computer perspective, some problems are computationally hard. Optimally labeling the point features of a map, for example, has been shown to be NP-hard (Marks and

Shieber, 1991). Other NP-hard design problems include page layout and pagination (Plass, 1981), and minimizing edge crossings and area of graphs (Johnson, 1982; Johnson, 1984). As a result of such computational complexity, computers must rely on heuristics to generate solutions to many problems. In addition, while people find it difficult to specify aesthetic criteria mathematically, computers are not well equipped to evaluate aesthetic criteria as part of their calculations. Computers may not have access to sufficient information to choose between two design alternatives because arbitrary semantic information may be needed to choose among several acceptable alternatives. In such cases, a person must often intervene to decide between two or more equally correct solutions or design alternatives; computers should not and cannot be expected to make this final determination.

The graphics design process, the sequence of actions taken in designing a graphic, should be a dynamic process based on iterative refinement. It can be decomposed into two tasks: *conceptualization* (selecting the objects to include, and how to organize them) and *articulation* (determining precise locations of the objects). Applications need to support and assist people in both tasks. The design process can also be characterized by the goals it is being used to achieve. A person may be trying to *instantiate* a particular design; the goal is to figure out how to get the computer to produce the graphic. Alternatively, the person may be interested in *exploring* possible design alternatives. This type of browsing activity assists a user in conceptualizing a particular design; viewing related graphics can help people identify what they do (or do not) want in a graphic. The design process often involves both tasks. A person may start with a particular design, but will often refine it after comparing it to other graphics of its type. This interweaving of search and modification is the crux of an iterative refinement process.

System designers must determine the roles of the user and computer in the graphics design process. The question of how to assign responsibility within the graphics design process is a difficult one to answer. In the following sections we examine the traditional approach to this problem and its associated drawbacks. We then introduce a novel collaborative framework for determining the division of labor. We conclude with an overview of the three graphics design tasks for which we have designed and implemented systems based on this collaborative framework.

1.2 Traditional Approach

Historically people have relied upon the *user interface* as the single point of contact between computers and people. The interface was thought of as "an input language for the user, an output language for the machine, and a protocol for interaction" (Chi, 1985). This view results in the traditional *master-slave* framework, in which the interface is a means for people to control computers. Under this approach, the division of labor is such that the user issues commands to the computer; the computer plays a passive role, responding to the user's actions.

Software packages for graphics design based on the master-slave framework range from manual to automatic. *Tool-based* packages typify the manual end. They provide a user with computer versions of real world artifacts, implemented with direct manipulation interfaces. Drawing packages, for example, provide users art materials such as paper, pens, pencils, paints, scissors, and paste. The computer acts as a recording device, merely noting the solution that the user generated. Rather than aiding the design process, tool-based applications may frustrate a user because of the difficulty of using commonplace tools in

an unfamiliar environment. Although computer-based graphics packages are often a step up from the days of designing on paper, they place too much responsibility on the user, and may be tedious to use. The user is responsible for both the conceptualization and the articulation of the design, while the computer plays a passive role.

On the automatic end, *oracle-based* software employs complicated algorithms to relieve people from having to do complex computations by hand. In such applications, the user poses a question to the computer, and the computer provides "the answer". Expert systems are a common example of this approach. An expert's knowledge and expertise is encoded into a program which will then be used by non-experts to solve problems in a particular domain. However, it is hard to know what question to ask, and difficult to state because doing so requires setting various parameters or specifying an objective function. In such automated applications, it is difficult to refine computer-generated designs. Editing is usually an indirect process — a user must tweak a variety of input parameters. Often a person will not quite understanding the impact a change will have on the final layout. As a result, people experience little control over the design process when using such black box software.

Software support for the design of graphics should provide people with mechanisms to create, refine and generate new designs. The design process itself is inherently interactive. As a result, software designers must take account of how easily and practically to incorporate a user and the underlying computer application into an integrated system.[1] Although systems fall on a spectrum from manual to automatic, little attention has been given to the middle of the range. The source of problems at the endpoints (i.e. automatic and manual applications) can be attributed to the failure of the system to delineate the roles of the user and the computer in an effective and useful manner. In the traditional master-slave framework, the software and its user are separate and independent components. As depicted by the diagram in Figure 1.1, data moves through such a system in series, with only one component working on the data at any given time. This pipe-lined approach enforces rigid boundaries between a person and the computer. An alternative approach is to integrate the two components, permitting them to work in parallel on the data. This approach is shown in Figure 1.2; both the person and the computer are able to modify the data at the same time, and both will see the results of the other's actions. This makes for a more collaborative and cooperative approach to design.

1.3 Collaborative Framework

The collaborative framework we propose views the interface as a means for people and computers to collaborate on solving problems, rather than as a mechanism for people to control computers. Its goal is to keep (or reintroduce) the user into the graphics design process; both the computer and the user participate in the design process. The system should be designed to exploit the strengths of each collaborator. People draw upon a broad range of experience and often rely upon intuition when making decisions. Computers, on the other hand, have a narrow area of expertise and greater computational power. Identifying these competencies is a key component of the collaborative paradigm.

The collaborative paradigm we propose relies on two key components:

[1] Throughout this thesis we will use the term *user* to denote people, the terms *computer*, *software*, and *application* to denote the machine, and *system* to designate the combination of user and computer together.

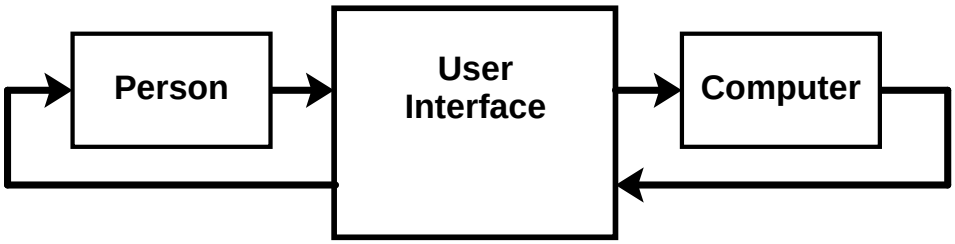


Figure 1.1: Traditional Master-Slave Framework.

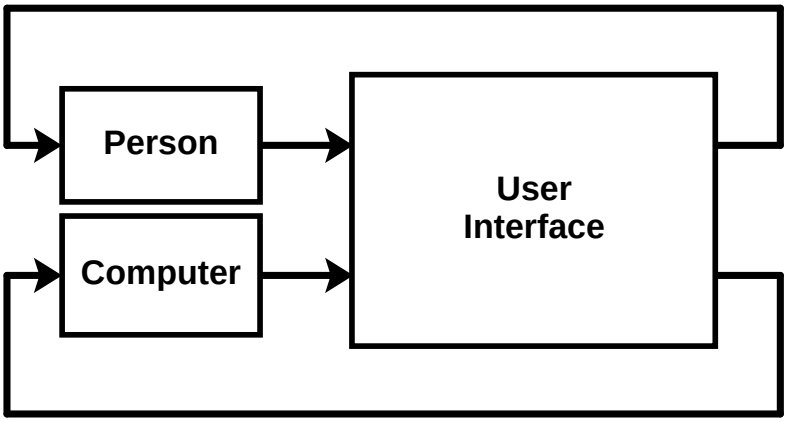


Figure 1.2: New Collaborative Framework.

- *Division of labor*: Defining a representation of the problem to be solved that naturally decomposes into a portion that a computer is good at and a portion the user is better able to perform.

- *Communication*: Designing languages and mechanisms to employ in transferring information between the person and the computer, specifying how much information to exchange.

As a concrete example, we will consider the graphic design problem of network diagram layout [2] which is one of the applications we tested hypotheses on. For instance, organizational charts are a common class of network diagrams. Given a set of nodes and edges, a person attempts to construct a diagram to communicate some information.

In designing a collaborative system, we must first find an appropriate representation of the layout task that can be shared by the computer and its user. Based on the representation, we must then determine the division of labor between the computer and its user for solving the problem. Computers are quite good at fine-grained placement of graphical objects whereas a user may find pixel-accurate placement, requiring fine-grained mouse gestures, extremely tedious. The user's role is more appropriately the coarse-grained layout, which will depend upon aesthetic and semantic judgments that may be difficult to specify in a discrete form. This division of labor with computer working at the local level and user more globally is typically a good match for the collaborators' abilities.

The second question to be addressed is communication. The user needs to communicate the global layout of the diagram to the computer. This goal might be achieved through a direct manipulation interface in which the user creates and places the nodes and edges of the diagram; most drawing and graphics packages utilize standard techniques for this task. Note that the layout provided by the user is only an approximation of the desired layout — the user need not take the time and effort to place the objects precisely. The computer's job is to convert coarse-grained gestures into fine-grained placement, and to indicate to people how this will be done. Many applications include a snap-to-grid feature which may be used for this function — it relieves users from the burden of working at the pixel level but enables them to understand (and predict) the actions taken by the application. The computer may utilize graphical means to aid a user in understanding its behavior; the grid used by the application may be visible, indicating to a user the closest point to the object being manipulated. The mathematical calculations performed by the application are not, however, revealed to the user.

The approach described in this thesis for utilizing the collaborative framework relies on the notion of optimization; the problem to be solved is framed as an optimization task. The original problem then becomes one of exploring a search space, determining local minima, and evaluating alternative solutions. One challenge is determining the appropriate roles for the person and the computer in the optimization task. We assign responsibility attempting to leverage the strengths of both participants. The user is responsible for global aspects of the problem: placing the computer into different areas of the search space, and determining when an acceptable solution has been reached. The computer works at a more local level, computing the local minima, displaying results to the user. In addition to establishing

[2] This task is also referred to as graph layout. We will use the terms interchangeably throughout this thesis.

the division of labor, we must also provide simple interface mechanisms to facilitate the interaction between user and computer.

Although others have used optimization in the design of graphics (Witkin and Kass, 1988; Poulin and Fournier, 1992; van de Panne and Fiume, 1993; Kawai, Painter, and Cohen, 1993; Schoeneman et al., 1993; Liu, Gortler, and Cohen, 1994; Sims, 1994; Tang, Ngo, and Marks, 1995; Christensen, Marks, and Shieber, 1995; He et al., 1996; Edmondson et al., 1997), they utilize a different division of labor between the computer and its user — the computer traditionally attempts to calculate globally optimum solutions. In our approach, however, the system is responsible for local optimization only. The user is responsible for the global portion of the optimization, as well as evaluating different local minima to determine an acceptable solution. The traditional approach to optimization in design is discussed in more detail in Chapter 4.

1.4 Problem Domains

To explore this new methodology, we have examined three problems from the domain of graphics design in detail: network diagram layout, parameter specification for representational graphics, and floor plan segmentation. Together these graphical design tasks provide good coverage of the design of both informational and representational graphics. The three systems we have developed illustrate the different strengths that a person and computer may contribute to the collaborative effort.

Network diagrams are a common form of informational graphic, and constitute an area in which it may be difficult for a person to define the aesthetic qualities desired in the final design. Most research in network diagram layout has focused on automatic layout. We have created an interactive constraint-based editor for network diagram layout, based on the collaborative framework described above. Our system, GLIDE, improves upon general constraint-based editors by providing a small but powerful set of specialized constraints specifically designed for drawing network diagrams. GLIDE is the first system to support people in specifying the visual organization of a diagram. The user provides an initial layout and constraint set, which characterize the desired visual organization of the diagram. The system then attempts to solve a constraint-satisfaction problem, by calculating a local minimum using a mass-spring physical simulation. The process is both interactive and iterative. GLIDE provides simple interface mechanisms for the user to create, view, manipulate and remove constraints. The user may intervene as the system attempts to provide a satisfactory layout, or once the system has finished. By using an intuitive local constraint-satisfaction scheme whose behavior is predictable and easily visualized through animation, GLIDE assists the user in understanding how to achieve a desired layout, and exploring design alternatives.

Parameter specification is an especially challenging problem for many computer graphic algorithms. Many techniques for generating representational graphics, such as volume rendering and physically-based animation, rely on numerous parameters in order to generate compelling images. The solution space of possible parameter settings and the corresponding space of resulting images are both large and complex. It is difficult for people to understand the impact and interaction of these parameters. In addition, many of these algorithms do not run in real-time, making it difficult to incorporate them into an interactive system in which a user is able to experiment with different settings. Our goal is to aid the user in the task of parameter specification. Under our collaborative methodology, known as De-

sign Galleries, the system pre-processes the design space, but enables a person to explore representative samples from the space in real-time. One interface approach uses a multidimensional scaling technique to layout representative thumbnail images in a two dimensional space. The user can then explore areas of interest within the layout in greater detail. This user-directed search exploits people's expertise at incorporating aesthetic criteria in evaluating design alternatives, and utilizes a computer's computational power in generating and providing representative candidates distributed throughout the design space.

Floor plan segmentation addresses the problem of identifying partially and fully bounded regions in a scanned bitmap image depicting a floor plan. The task of region segmentation, like many graphic design tasks, can depend in the end on arbitrary semantic information about the material depicted in the bitmap to which no purely syntactic method can be sensitive. Because no fully automatic method is to be expected, we believe that it is crucial to think of the task of region segmentation as being solvable only semi-automatically. Thus, any system for region segmentation needs to be evaluated not only for how well it works in an absolute sense, but also for how well any remaining problems (for there will be some) can be easily handled by simple human interventions. Other systems for identifying regions in a bitmap floor plan image place too much burden on the user; they require a person to trace out regions of interest with the mouse. Our collaborative system automatically extracts partially and fully bounded region definitions from a bitmap image. It suggests these candidate regions to the user, who can then further subdivide a single region, or join several regions into one region, using only gross mouse gestures. As our system requires minimal input from a user, it is better suited for generating this type of representational graphic.

Chapter 2

Background

In this chapter, we discuss three topics that serve as necessary background for this thesis. First we define the term *computer-user interface*. We then expand upon the idea introduced in Chapter 1 of design as an iterative process, and categorize graphics into two categories: informational and representational. Finally, we provide brief discussions of existing software packages for each category of graphic that incorporate some notion of collaboration into their computer-user interface.

2.1 What Is An Interface?

The notion of a "user interface" emerged in the 1970's during a time when computers were beginning to be used in a commercial setting by ordinary people. Prior to that time, computers were used primarily by specialists — scientists and engineers who received special training for using the computer. Computers were very costly; people's time was an inexpensive resource in comparison (Preece, 1994). Therefore, it was easier to adapt the people to the computer than vice versa. As non-specialists began to interact with computers, however, more attention needed to be given to the software and input/output devices that enabled people to work with a computer. From an engineering perspective, the computer already included a variety of interfaces, and so the user interface was simply another component of the application. It was typically defined "to be all user and machine behavior that is observable by an external observer" (Chi, 1985). In practice the interface was that which was presented *by* the computer *to* a user, rather than a user's interface to the computer. Note that this relationship is not symmetric, and the interface from the user to the computer is not included in standard uses of the term "user interface". Grudin (1993) provides a detailed discussion on what is meant by the term "user interface" and why this term is inappropriate in today's computing environment.

In this thesis, we will use the term "computer-user interface" as it provides a more accurate indication of our view of system design. We view the interface as the common ground shared by the two collaborators in solving a particular problem. We define it to be the mechanisms and procedures available to both the computer and its user. Although we attempt to exploit human expertise (e.g. global problem solving, perceptual acuity), we do not examine issues typically termed as human factors. Moran (1981) suggested a broader use of the term "user interface" in which the perceptual and cognitive processes of the user were included. Although such processes are important considerations, a formal analysis of them is beyond the scope of this thesis. We do, however, informally incorporate some

psychological aspects into the determination of the appropriate roles of the person and the computer in the collaborative framework.

The term "human-computer interaction" was adopted in the 1980's to characterize the interdisciplinary research that addresses the design of interactive systems. Dix et al. (1993) note "Human-Computer Interaction (HCI) is, put simply, the study of people, computer technology and the ways these influence each other. We study HCI to determine how we can make this computer technology more usable by people." From this perspective, the computer-user interface is extended to include both a person's and a computer's inner processes; it is a medium that enables people and computers to collaborate:

> "The computer is a tool, a complex artifact that can extend our reach. The design discipline of human-computer interaction systematically applies knowledge about human purposes, human capabilities and limitations, and machine capabilities and limitations in order to enable us to do things we could not do before. Another goal of HCI, as suggested in the definitions given above, is to enhance the quality of the interaction between people and computers. We strive, for example, to make technology easier for people to learn and easier for them to use." (Baecker et al., 1995)

The goal of computer-user interface design is to make software easier to use. Although this may appear to be a simple goal, many of today's applications are still incredibly difficult to use and understand; they hinder rather than aid a user in doing some task. Two key problems contribute to the difficulty: expectations and communication. The first problem is the user's expectations about a computer. Too often, the user just wants the computer to do the right thing; the computer should figure out "the answer". But what if there is more than one answer? How should the computer choose between them? What if the computer cannot find any answer, or it will take some unacceptably long time to find an answer? Should the computer give up, or should it get some help from the user? Inappropriate expectations may also contribute to the second problem — that of communication. Human-computer interaction is a form of communication that should be facilitated by the interface. Neither participant, however, is doing a particularly good job. In many applications, it is very difficult for users to convey to a computer what they want it to do. Computers, in turn, do not give understandable feedback to their users. Part of the communication problem arises from the existence of two "views" of the world. For the computer, every detail, both data and procedure, is specified formally and explicitly. For the user, the environment is not as clearly defined — it includes both implicit and explicit information, and people are often unaware of what guides their intuitions. The computer-user interface needs to provide a mapping between the two views, making the transformation transparent to the user. Often, a user is forced to work within the system view, using arcane commands, and setting various parameters, without understanding their meanings.

Our work on collaborative computer-user interfaces addresses these two problems. First, it helps to define a framework that better shapes the expectation of the user, namely that the computer is a collaborator with whom the user can work to accomplish a particular task. The computer is not a black box, or a magician that magically comes up with "the right answer". Rather it works with the user to reach an acceptable solution. Second, the computer-user interface provides simple interaction mechanisms that facilitate the communication between the two participants in the graphics design process. It makes a clear distinction between the user's and computer's view; it hides implementation details that

should be transparent to people by automatically translating between these two views. Thus the computer-user interface enables people to work in a more comfortable and appropriate environment.

2.2 The Design of Graphics

In this section, we categorize graphics into two categories based on the level of abstraction within the graphic. For each category of graphic, we describe related work for a sample of design tasks.

Informational graphics, such as bar charts and network diagrams, are graphic elements in which a conventional format is used to present a data set. Although the data for these types of graphics are often derived from the physical world, they are presented by people (and to people) at a more abstract level. There is not necessarily a literal correspondence between properties of some real world object and those of the corresponding graphic elements. Typical graphic design problems involving informational graphics include tasks such as page layout, hypermedia design, network diagram layout, and designing bar graphs and charts.

Representational graphics are graphic elements used to model real world objects; people have an existing model or idea from which they attempt to generate an electronic counterpart.[1] The graphic represents an actual (or imagined) object, rather than abstract data; graphical properties (e.g. distance, color, brightness) correspond literally (though perhaps approximately) to their real world counterparts. Typical applications for this category of graphic include CAD/CAM packages, animation, medical imaging, and virtual reality environments.

2.2.1 Informational Graphics

Business or presentational graphics are a typical form of informational graphic; they present abstract data using conventional formats such as charts or graphs, and are a useful medium for both analyzing and presenting data. Systems need to support and assist the user in both tasks of the design process: conceptualization and articulation. Ideally a person's efforts should be directed towards conceptualization, the more creative part of the design process and a task better suited for people. We rely on the computer for articulation. In practice, however, producing useful graphics is difficult, in large part due to the lack of support in available software for these two component activities.

Current packages for creating informational graphics are of limited use for one of two reasons. Many systems restrict users to a small set of static designs — a user selects a style, and the system automatically displays the data using a predefined algorithm. This inflexibility limits the user's participation in conceptualizing a design, and can frustrate people as they are unable to refine a design or generate variations of a given graphic. Other applications provide a user with more freedom, serving merely as recorders of every graphical aspect of the design, each one specified by the user at the level of primitive graphic

[1] In some applications, the object may not actually exist or even be part of the physical world. A CAD package, for example, can be used to design a car that will be produced, but does not yet exist. An animation may include mythical creatures, such as unicorns, or take place on an alien world that no human has actually experienced. We include such cases under the broad category of representational graphics.

operation. So much control is often a burden. The user is responsible for both phases of the design, specifying the data, the graphical properties and relationships, and the placement of all the graphical objects through tedious fine-grained mouse gestures.

Some previous research on "intelligent interfaces" has focused on fully automating the design of presentational graphics (Mackinlay, 1986; Roth and Mattis, 1990; Seligmann and Feiner, 1991) as a means for the computer to communicate information to its users. Although such automated methods are useful in enabling computer applications to communicate information graphically to people, they are not optimal for human users who are themselves designing graphics. People often incorporate aesthetic components into informational graphic design tasks. Such aesthetic considerations are often not taken into account in automated systems. In the domain of network diagram layout, for example, automated systems have focused on minimizing edge crossings and minimizing the total area used by a layout. The visual qualities of a layout are not included in the computer's calculations. A computer is not well equipped to evaluate such aesthetic criteria as part of its calculations, making it difficult or impossible for an automatic system to choose appropriately among several acceptable alternatives. Thus, any system for creating, for instance, a network diagram layout, needs to support a user in refining and generating new solutions. Ideally, it should incorporate the user into the design process itself. This is illustrated by the GLIDE system described in Chapter 3.

Recent research on the interactive design of presentational graphics takes this approach, and makes implicit use of collaboration between a user and the computer. The Gold System (Myers, Goldstein, and Goldberg, 1994) supports users in designing business graphics. It allows users to sketch a few instances of data elements using a standard drawing editor interface. The system then automatically instantiates the rest of the data, based on the users initial sketch. Because it allows the user to sketch an example of the desired graphic (rather than precisely drawing all objects with exactly the right size and location), Gold enables a user to generate graphics quickly and easily. In addition to producing standard spreadsheet graphics, users are able create hybrid charts, impossible under other existing packages. Gold also enables users to edit and refine an image, simply by redrawing desired portions. The system updates the graphic accordingly.

Sage, SageBook and SageBrush (Roth et al., 1994) are another family of tools used to create charts and graphs. Using SageBrush, users can assemble a graphic from scratch by selecting various graphics, and assigning data to their various properties. SageBook enables a user to browse existing designs, which they may use for inspiration, or as an actual template for their own design, which they may then edit. Sage is a knowledge-based tool for automatically designing graphics, and incorporates information from the other two systems, providing users with a design environment suitable for novice and expert designers alike. Both Gold and the Sage family exemplify the collaborative style interface we propose; they minimize required user input, reduce user tedium, and provide powerful systems for the design of graphs and charts. In Chapter 3 we examine network diagrams, a common form of informational graphic for which most previous research has focused on automated layout.

2.2.2 Representational Graphics

Research in the field of computer graphics has traditionally focused on the algorithms needed to produce representational images and animations. Image rendering techniques, such as

volume rendering and radiosity, focus on shadows and reflections for scene generation. Animations strive to be physically realistic, incorporating real-world physics and interactions between objects; less emphasis has been placed on integrating these algorithms into interactive tools that would aid users in creating and designing such graphics. The master-slave model of interface design dominates; people retain primary control of the design process, with computers delegated to a secondary status.

Consider for example the domain of animation, a common form of representational graphic. Almost twenty years ago Catmull (1978) described the early experiences of transferring cel animation to computers. Today computers are still delegated to the status of recording devices when used in motion capture systems. People wearing sensors generate desired motions, while the computer records the locations of the sensors over time. The computer uses the sensors to generate control-points for mapping the behavior onto a user-supplied model, generating the corresponding animation. Although the results are physically-realistic and visually plausible animations, the computer plays a passive role in the process. Key-framing, another popular technique for animation, employs computers as the in-betweeners, a role previously delegated to human apprentices. Computers interpolate between user-specified key-frames to generate the twenty-four frames per second needed for smooth animation. If computers generate an undesirable sequence of actions,[2] the animator will insert a new key-frame to further constrain the animation to be generated by the computer. Computer-aided cel animation, motion capture and key-frame systems are all implemented using a master-slave relationship. The computer's power is focused on computing and rendering specific animations which have been fully specified by users. People typically have an idea of the animation they would like to generate, and their energies are spent trying to convey the information to the computer.

This is not to say that computers have not had a major impact on animation; computers do play an important role in today's animations. Even limited automation offers a number of benefits. In the fall of 1995, Walt Disney released *Toy Story*, the first full-length, all-digital movie created entirely by artists using 3D computer graphics tools. The use of computers reduced the number of animators used from over six hundred used on previous films, to one hundred and ten (McWilliams, 1995). The point is that the role of the computer is that of tool.

> "Computer animation combines the skills of traditionally trained character animators with the most sophisticated 'pencils' in the world. Using computers as a tool, the filmmakers introduce a unique three-dimensional animation look, with qualities of texture, color, vibrant lighting and detail never seen before in traditional animated features." (Anonymous, 1996)

Although computers are increasing productivity within the traditional domain of animation, they could be used to explore the animation design space rather than just rendering a specified animation. Along with supporting traditional animators, such systems would extend the accessibility of animation beyond specialized artists by focusing on the creative process and moving beyond the production of the animation. The recruiting web page for Pixar includes a banner with the slogan "Computers Don't Animate, People Do!" (Pixar,

[2]The interpolation methods used in computer-aided key-framing are not without fault. Linear interpolation, the simplest method, often results in physically unrealistic animations, or causes articulated figures to behave in other visually unsuitable ways.

1997) Although we don't envision computers becoming animators in their own right, the division of labor could be more equally distributed. Design Galleries, a family of applications discussed in Chapter 4, incorporate the collaborative approach we propose and are appropriate for generating animations and other forms of representational graphics.

Not all graphics are ends unto themselves; a graphic may be an intermediary means (or an interface) to another representation to be used in solving another problem. Many CAD/CAM tools are used to create representational graphics which are then used to generate a specification to be used in the manufacturing process of the object. Architectural drawings are a typical example — the "graphic" represents a building to be constructed or modified. AutoCAD is a package that supports people (typically trained architects) in manipulating a floor plan: either designing one from scratch or modifying an existing floor plan that is already in an AutoCAD format. In many ways AutoCAD is comparable to standard drawing packages in that it follows the master-slave paradigm; the application is a drafting assistant that permits manual manipulation of the geometry within the design. There is minimal collaboration on the part of the computer in that it maintains constraints and provides an appropriate set of tools and objects for a person to design a building, but the person is responsible for both conceptualization and articulation of the design. The application provides little support for the creative aspects of design and exploring design alternatives.

Kochhar (1990) emphasized the importance of browsing within the design process. He developed FLATS, a system for automated floor plan layout (Kochhar, 1991) in which a person provided an initial partial design and a set of criteria to which the final design must adhere. The computer then generated a set of design alternatives for the user to browse through; any of the resulting designs could be refined by the user and then used to generate the next set of alternatives. This cycle is similar in nature to interactive evolution systems, as described in Chapter 4. An important aspect of Kochhar's work is its collaborative nature. FLATS supported people in exploring design alternatives, and refining a design through an iterative refinement process.

Finally we consider a system which employs a collaborative interaction mechanism to help users generate a desired graphic. Baudel (1994) introduced a novel spline editing technique aimed specifically at graphic designers. In traditional systems, splines are edited by specifying control points and tangents for a curve. Artists, however, do not think in these terms. The mark-based paradigm more closely matches an artists actions in the physical world; a series of marks are used to refine an existing curve. This form of communication is more natural to the target users, and illustrates the importance of communication within the collaborate framework we propose. The division of labor is such that the user (in this case an artist) is responsible for conceptualization (the general shape of the curve, and the computer, articulation (the precise location of each point in the curve). The user indicates the desired shape of a curve through a series of marks, rather than by manipulating one or more control points, which correspond to the computer's definition of the curve. The naturalness of the interaction enhances its usability. This particular system attempts to mimic the real world actions of a person. While such interfaces are useful in some situations, they are not required by the collaborative framework suggested in this thesis.

2.3 Summary

We view design as a two step iterative process involving conceptualization and articulation. The goal of conceptualization is to determine which objects to include in a graphic, and how to organize them, and under the collaborative framework is generally the responsibility of the user. Articulation determines the precise locations and graphical properties of the objects, a role given to the computer. In addition, the design process is often guided by two strategies. Instantiation occurs when a person uses the computer to generate a particular design. Browsing and exploration techniques are useful when a person has only a vague notion of the desired graphic. This type of search helps overcome the problem of unquantifiable output characteristics. People need not know or enumerate ahead of time the desired quality of a graphic — they need only recognize it when they see it. People often employ both strategies in the design of graphics. In the following chapters we present the three systems we have developed based on our collaborative framework.

Chapter 3

GLIDE

In this chapter we describe GLIDE (Ryall, Marks, and Shieber, 1996; 1997), an interactive editor for network diagram layout. It is the first system to support users in specifying the visual organization of a diagram. By using an intuitive local constraint-satisfaction scheme whose behavior is predictable and easily visualized through animation, GLIDE assists the user in understanding how to achieve a desired layout, and exploring design alternatives. Such a division of labor — having the user work globally, while the system works locally — exemplifies the collaborative approach proposed in this thesis.

3.1 Introduction

As described in Chapter 1, network diagrams are a form of informational graphic in which a set of nodes and edges are arranged to construct a diagram that will communicate some information. Entity-relationship diagrams, PERT charts, state transition diagrams, flow charts, and organizational charts are common classes of network diagrams. The task of network diagram layout (deciding which nodes and edges to use, and how to best arrange them) is a good representative problem for designing informational graphics.

There are three main considerations for designing such graphics. First, syntactic criteria define the well-formedness of a graph. For example, no two nodes should overlap. Next, layout aesthetics, such as minimizing edge crossings, aid the readability of the graph. Finally, perceptual organization assists people in the semantic interpretation of the diagram. This last component is crucial for a person's understanding of the information being presented in a diagram. Inappropriate or misleading perceptual organization of a diagram has been identified as a major cause of design flaws in informational graphics (Kosslyn, 1989; Marks and Reiter, 1990).

Most small to medium sized graphs (i.e. those with fewer than 50 nodes) that appear in publications or presentations are still drawn with the aid of fairly primitive commercial drawing tools like Microsoft's PowerPoint or Claris Draw. These manual tools provide minimal support for aesthetic graph layout; a person is responsible for both the conceptualization and the articulation of a layout, often working at a pixel level. Research in the graph drawing community has neglected the issue of perceptual organization and has instead focused on optimizing layout aesthetics in automated systems. Thus, most graph-drawing algorithms cannot support the exquisite symmetries, spacings, and alignments that graphic designers utilize in professional-grade work. This kind of layout detail can be achieved in

some constraint-based drawing systems, but the very general capabilities of such systems tend to make them cumbersome for the specific task of graph drawing.

Our focus has been on an interactive system that supports users in specifying perceptual organization of network diagrams based on the collaborative paradigm proposed in this thesis. We have built a system, called "GLIDE" (Graph Layout Interactive Diagram Editor), for interactive graph layout that organizes the interaction in a more collaborative manner than previous systems. We take advantage of user's expertise at globally designing the layout, and the computer's computational superiority; the user is responsible for an approximate layout of the nodes, and for specifying any desired visual organization. Under this paradigm, the user has the flexibility to create interesting designs, without the burden (and tedium) of having to precisely place every object in the layout. Using a direct manipulation interface, the user can easily modify a layout, guiding the system to produce a suitable graphic. GLIDE improves on general constraint-based systems by providing a specialized set of constraints, simple mechanisms for a user to add and delete constraints, and an intuitive method for solving the constraints. In addition, by incorporating animation into the process, GLIDE assists a person in understanding how to achieve a desired layout. We describe our approach in the following section. We then walk through a sample interaction between a person and GLIDE in Section 3.3 and provide the system design and implementation details in Section 3.4. We compare GLIDE with existing approaches for network diagram layout in Section 3.5.

3.2 Our Approach

GLIDE is a constraint-based system designed specifically for drawing graphs. It incorporates a small set of *macro* constraints, or Visual Organization Features (VOFs), (Marks, 1991; Kosak, Marks, and Shieber, 1994), which are listed in Figure 3.1;[1] the application of each VOF is illustrated by *before* and *after* layouts. VOFs are one mechanism for specifying and incorporating the perceptual groupings of a graph, and VOFs have been incorporated previously in a few fully automated graph layout systems (Kosak, Marks, and Shieber, 1994; Dengler, Friedell, and Marks, 1993); GLIDE is the first to allow interactive specification and manipulation of high-level VOFs such as these. In GLIDE, a user may apply and remove any number of VOFs interactively.

GLIDE applies user-supplied VOFs by converting them into a set of spring forces affecting the nodes in a graph drawing. Additional spring forces are introduced automatically to preserve syntactic correctness of the drawing, such as preventing nodes overlapping other nodes and edges. The user may also apply force directly to a node by dragging it with the mouse; this is a convenient override mechanism. The nodes, which are modeled as point masses, are moved by physical simulation into minimum-energy configurations. The simulation of the mass-spring model is continuously animated, indicating to the user the influence of the VOFs.

Incorporating animation into interfaces is not a new concept (Baecker and Small, 1990). More recently, Thomas and Calder (1995) have demonstrated the usefulness of animation in direct manipulation interfaces. Using physical characteristics such as inertia and gravity, they provided substance to objects, creating better user feedback. Feedback is of course essential to the success of direct manipulation interfaces; the use of animation in an interface

[1] All figures in this chapter were drawn with the GLIDE system, except for Figure 3.9.

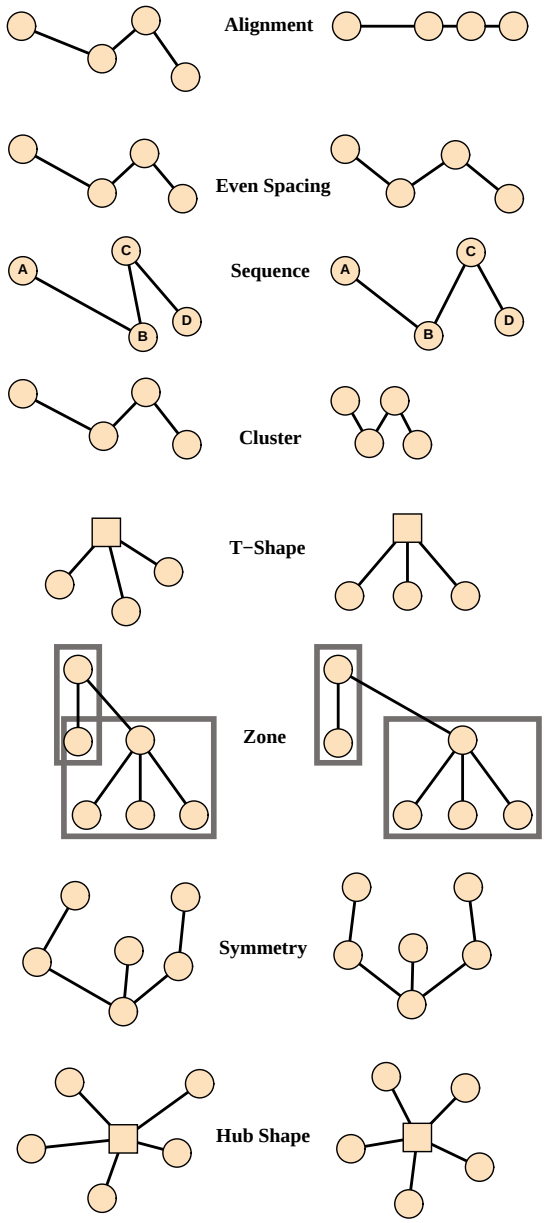


Figure 3.1: Visual Organization Features (after (Kosak, Marks, and Shieber, 1994))

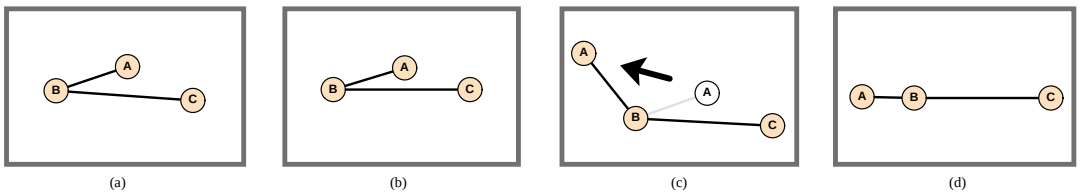


Figure 3.2: User intervention is required to find a globally optimal solution.

serves an important communication need. GLIDE exploits this approach in its implementation by continually animating the physical simulation of the mass-spring model. User interactions are also animated. As the user moves a node, for example, its position is updated in the physical simulation, causing the system to move other nodes as it attempts to satisfy existing constraints.

Using a direct manipulation interface, a person may move nodes and groups of nodes, along with adding and deleting any number of nodes, edges, or VOFs, to modify the diagram and its layout. Such manipulations are useful not only for exploring design alternatives but for providing "advice" to the system when it finds itself in a local optimum. Figure 3.2 is a simple example. In Figure 3.2(a), we see three nodes, connected by two edges. The user has added a single Alignment VOF to the set of nodes. The system attempts to satisfy this constraint by moving the three nodes toward an implicit horizontal line running through the vertical centroid of the three nodes; meanwhile, the syntactic constraint prohibiting overlap provides a repulsive force between nodes and edges. In Figure 3.2(b), GLIDE has moved node A down, and nodes B and C upwards. The three nodes cannot be aligned, however, due to the edge between B and C. By manually moving A anywhere to the left of B, as illustrated in Figure 3.2(c), the user obtains an optimized layout, such as the one in Figure 3.2(d).

These interactions occur as the simulation is running, and allow the computer and system together to collaborate in finding better global solutions to the implicit constraint-satisfaction problem. Such advice is especially useful in cases of over- or under-constrained designs. If a set of VOFs generates an over-constrained layout, GLIDE will find the nearest stable configuration, which may satisfy different VOFs to varying degrees. The user, who controls the design process at a global level (the choice of VOFs and the gross placement of nodes in the diagram), can easily guide the computer to find more satisfactory solutions and acceptable layouts by moving nodes or adjusting VOFs. In case of under-constrained layouts with multiple solutions, again the user can provide advice to explore alternatives.

Although a weak mechanism for satisfying constraints, energy minimization through physical simulation handles over-constrained systems gracefully, and provides an easily understood metaphor for the user. Thus, the use of a constraint-satisfaction scheme (mass-spring simulation) that is intuitive and predictable, rather than one better at finding global solutions, is deliberate. GLIDE is not intended to be good at globally satisfying the VOFs by itself. Rather, it is intended to provide an interface that allows a useful collaboration between user and computer in solving the layout problem. For this purpose predictability, simplicity, and the compelling nature of the animation are far more important than global

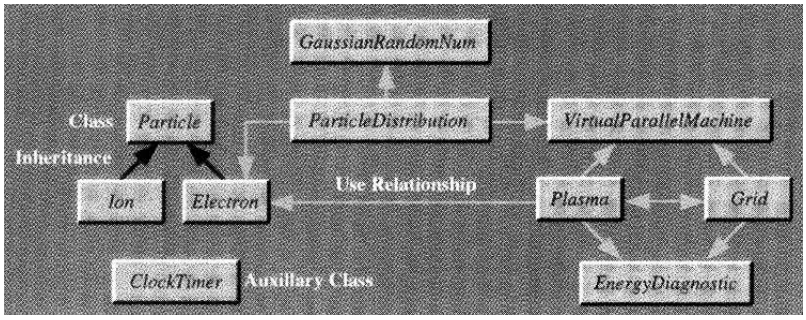


Figure 3.3: Scanned diagram on which the example was based.

optimality; these characteristics are an integral part of GLIDE's collaborative nature. Participant (computer and human) are given responsibility for what they do best, relying on simple techniques to communicate with and guide each other.

3.3 Example Interaction

Unlike traditional master-slave interfaces in which the user makes a request and waits for a response, GLIDE is an integrated system in which both the user and the computer affect the state of the design at the same time. The system does not pause while the user adds or moves nodes and edges, and applies or deletes VOFs. Likewise, the user can intervene and continue to interact with the layout as the system adjusts nodes' positions, attempting to satisfy the various constraints specified by the user. It is exactly this animation and simultaneity of action that makes GLIDE a compelling collaborative system, but that is difficult to express (as we have below) through a series of static snapshots. Figures 3.4-3.8 show snapshots of various intermediary stages in the process of drawing a given graph. Figure 3.4 depicts the entire system interface; other figures show only the canvas area.

For simplicity of exposition, the example interaction depicted here is based on a task in which the user attempts to create a particular layout already envisioned, rather than exploring alternative layouts. Our goal is to replicate the drawing in Figure 3.3 taken from a paper by Norton, Szymanski, and Decyk (1995). In Appendix A, we use GLIDE to explore design alternatives based on this same graph and to explore a data set previously unseen by the user.

The first step in the drawing process is to place the desired number of nodes in approximately the desired layout. To create a node, the user clicks the left mouse button on the canvas area. As is standard in drawing tools, the user has control over a variety of graphical properties of the nodes, including shape, font, background color, foreground color, border color, and dimensions. These are derived from system defaults, which can be set by a user, and modified at any time using an edit dialog window. Each node is automatically sized to accommodate its label; if a node's label is changed, the computer will automatically enlarge a node to accommodate its new label. These capabilities are not depicted in the figures. As the user adds more nodes, the computer is providing collaborative aid by automatically enforcing prohibitions of overlapping nodes; nodes too close together will be repelled from each other.

Edges, directed or undirected, linear or orthogonal, can be added between nodes. The

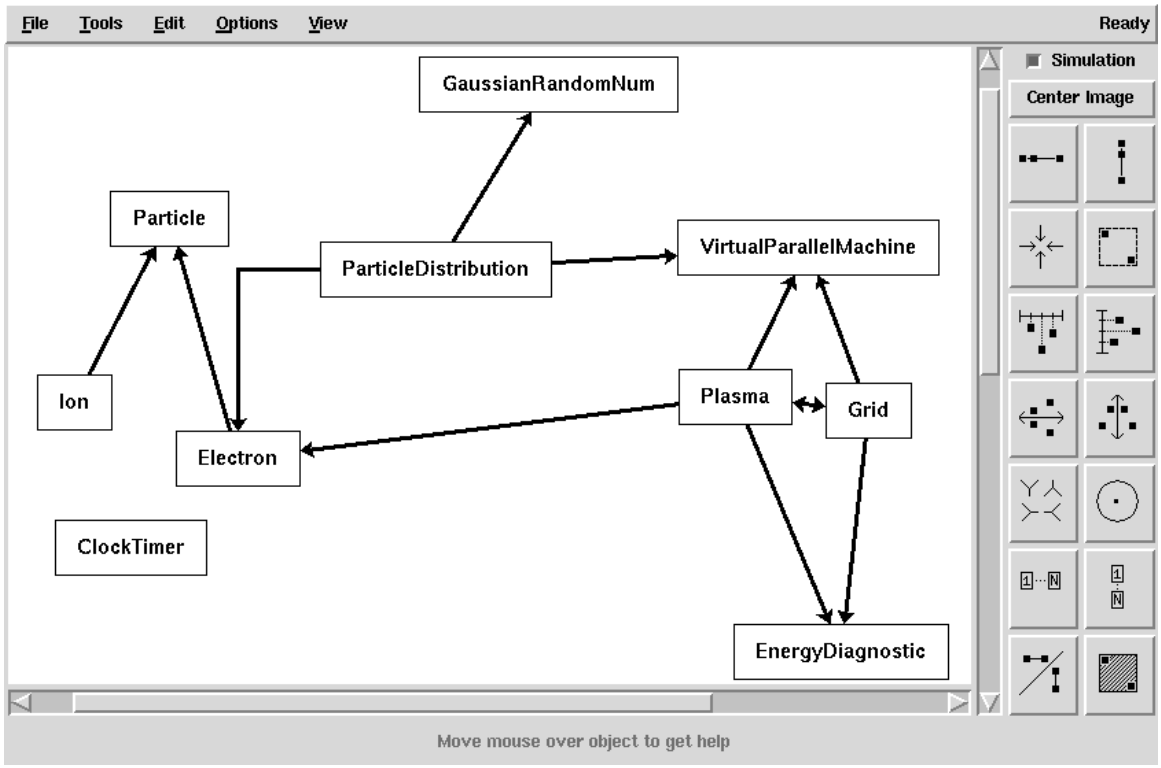


Figure 3.4: Initial layout, with labeled nodes and edges, shown in the context of the GLIDE screen layout.

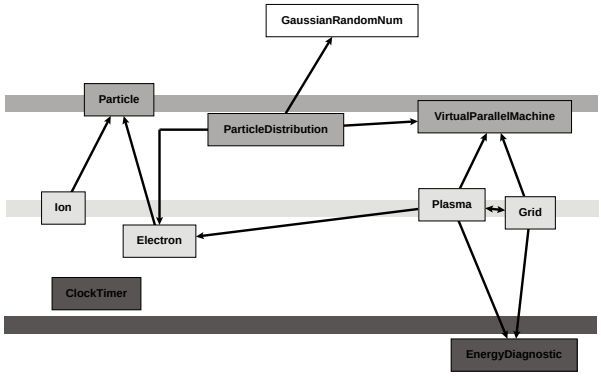


Figure 3.5: User adds an *Alignment* VOF to each row.

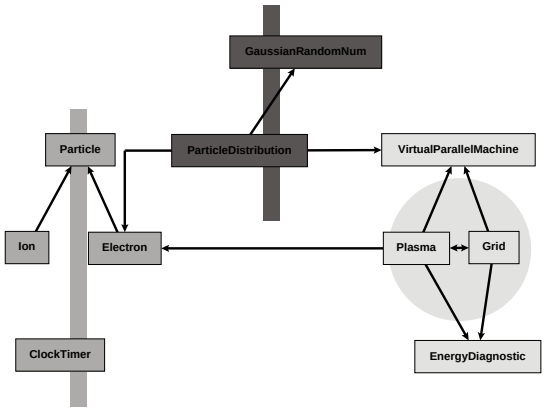


Figure 3.6: User adds three more VOFs: *Symmetry, Alignment* and *Hub Shape*.

user first clicks the left mouse button on the origin node. As the user drags the mouse towards the destination node, GLIDE displays an edge (anchored at the origin node, and whose head follows the mouse). Releasing the mouse button over the destination node causes an edge to be added; releasing it elsewhere aborts the edge connection process. The computer enforces prohibitions of node-edge overlaps; intersecting edges and nodes will be repelled from each other. Figure 3.4 shows the layout after an initial node- and edge-placement phase. Note the orthogonal edge between the "ParticleDistribution" and "Electron" nodes. GLIDE will maintain the orthogonality constraint (comprising two alignment constraints) throughout the design process.

To add a VOF to the layout, the user first selects a set of nodes using standard mouse techniques such as clicking or region-dragging. The user may then apply one or more VOFs to the set by pressing the appropriate push buttons, located on the right of the window in Figure 3.4. In Figure 3.5, the user has applied a horizontal Alignment VOF to the second, third, and fourth rows of nodes. As VOFs are applied to the nascent diagram, graphical indicators of the constraints are added, as described in Section 3.4.3. In GLIDE, the graphical VOF indicators are shown visually by a user-responsive highlighting mechanism that cannot be replicated in static images. We therefore use node coloring and static icons to indicate different VOFs in the figure; for instance, the grey rectangles in Figure 3.5 serve as a graphical indicator of the Alignment VOFs.

The system satisfies these constraints using the mass-spring simulation described in the next section. Indeed, movement of the nodes to satisfy the constraints would typically proceed while the user is adding more VOFs. Figure 3.6 shows the stable configuration that ensues after the three Alignment VOFs have been satisfied, along with three more VOFs the user has added. On the right, the user has added a Hub Shape VOF, indicated as a light gray circle. The center two nodes (dark gray) are to be vertically aligned. Finally, the four nodes on the left have had a Symmetry VOF applied to them.

Once again, the system attempts to satisfy all constraints, both old and new, in determining each node's placement. Figure 3.7 shows the updated node positions. Each row is still aligned, and the new VOFs have been satisfied as well. In addition, the user adds three text labels to the layout. Text labels are a specialized node type described in further detail

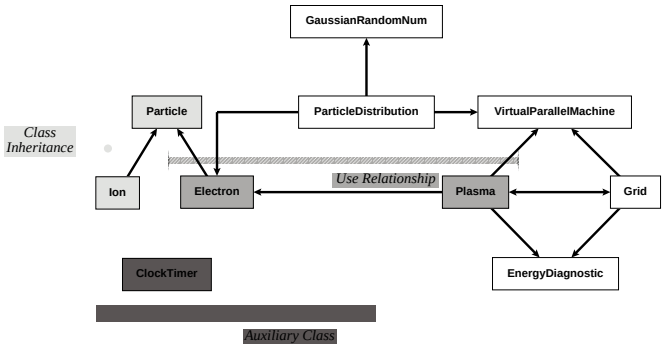


Figure 3.7: User adds three text labels, and three VOFs: *Clustering, Even Spacing* and *Alignment*.

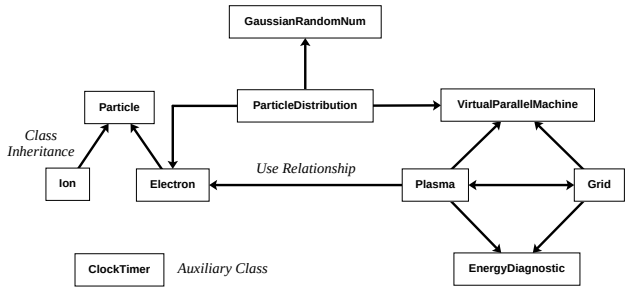


Figure 3.8: The finished layout.

below. The user has applied three more VOFs, which will better position each of the text labels. The light-gray nodes on the left are subject to a Cluster VOF. The dark-gray nodes on the bottom are to be horizontally aligned. The middle three nodes, shaded medium gray, are to be evenly spaced. Figure 3.8 shows the final layout generated by the full set of nine VOFs.

3.4 System Design

From a user's perspective, GLIDE is a simple high-level interface for adding and deleting nodes and edges, and for applying and removing various VOFs, thereby inducing new drawings. This facade is maintained by the underlying system, which is continually translating user actions into low-level constraints that it then tries to satisfy. In this section, we describe the relationship between the high-level VOFs and the low-level constraint mechanisms. In addition, we also describe how the constraints and constraint-satisfaction process are made apparent to the user.

3.4.1 Constraint Formulation

Constraints on graphs fall into two main classes, syntactic and semantic. As we have seen in Section 3.3, syntactic constraints are introduced automatically by the system, while semantic constraints (in the form of VOFs) are added interactively by the user. The system transforms constraint instances from both categories into a mass-spring model as described here.

Syntactic constraints are universal requirements necessary for a diagram to be well-formed. The GLIDE system respects two such constraints (see Figure 3.9):

- *Node-node overlap:* Two nodes should not overlap. This constraint is enforced by placing a spring between each pair of nodes. The spring's rest length is the required minimum distance between nodes, but it also has the property that its spring constant reduces to zero when stretched beyond its rest length. The spring therefore only applies force when the two nodes overlap, compressing the spring. Overlapping is thus prevented, but movement apart is not penalized. This is one of several ways in which the simulation is rendered nonphysical by generalizing the notion of a spring.

- *Node-edge overlap:* A similar spring is placed between each node-edge pair. Nodes are the only objects to which force can be applied, so the goal of applying a force to an edge is actually accomplished by applying half the force to each of the two nodes at the edges' endpoints. This disassociation between the spring's conceptual endpoints and the points of application of the spring's force is another example of how our model differs from more physically faithful mass-spring systems.

Applying these two syntactic constraints alone, the system enforces the well-formedness of diagrams. Semantic constraints, expressed as VOFs, enhance the visual form of the drawing. GLIDE supports the following VOFs, implemented with sets of springs as described (see Figure 3.9):

- *Alignment (horizontal, vertical, either):* The set of nodes should be collinear and axis-aligned. A spring with rest length zero is attached between each node and a virtual axis-aligned line through the centroid of the nodes. Note that forces are applied only to the nodes, and not to the virtual axis. In the case of a horizontal or vertical Alignment VOF, the axis-alignment is to the respective axis. The third case uses the axis to which the nodes are already most closely aligned.

- *Equal Spacing (horizontal, vertical):* The nodes should be spaced evenly along the given axis. Spacing along the orthogonal axis is unconstrained. Adjacent pairs of nodes are connected with springs whose rest length is the computed average distance between adjacent nodes.

- *Sequence (horizontal, vertical):* The nodes should be ordered in the current sequence along the given axis. Springs with a very short rest length and with asymmetric spring constant (zero if nodes are in proper sequence, positive otherwise) are placed between adjacent nodes in the sequence to keep them in order.

- *Cluster:* The set of nodes should be clustered together. Springs with a short rest length are placed pairwise among the nodes.

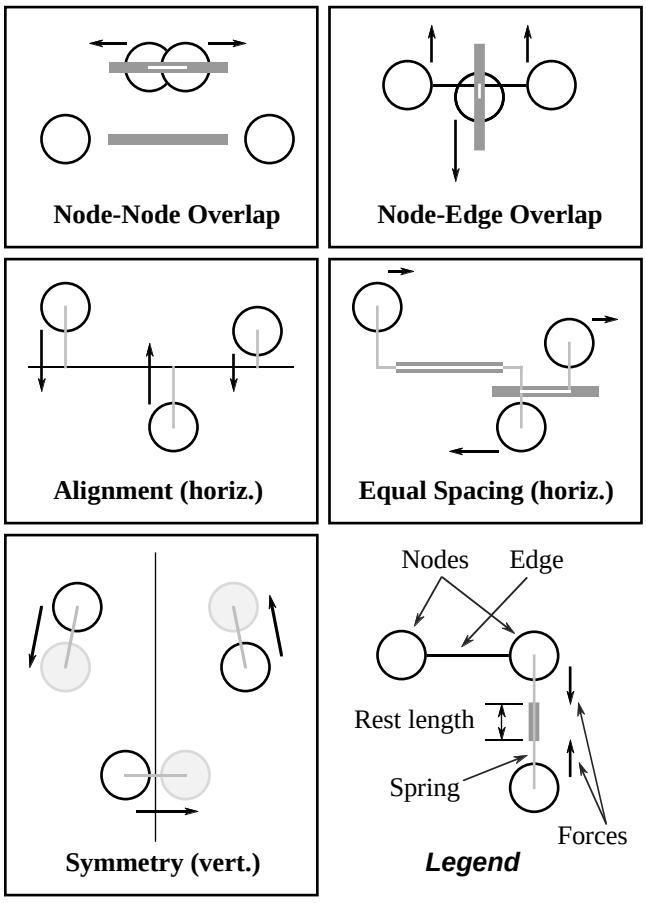


Figure 3.9: Reduction of some sample VOFs to systems of generalized springs.

- *Zone:* The bounding box of the nodes should contain no other nodes. The bounding box is treated as a "super-node," the node-node overlap method is applied, and the resulting forces are applied equally to the nodes comprising the zone.

- *Symmetry (horizontal, vertical):* The nodes should be symmetric about the given axis. Each node is paired with the node closest to its reflection about the horizontal or vertical line through the centroid of all the nodes. (A node may be paired with itself if it is closest to its own reflection.) Equal and opposite forces are then applied to the nodes in each pair to make them symmetric about the line of reflection.

- *T-Shape:* The nodes should form a T-shape, as in a tree diagram. The user specifies which of the nodes is the parent. The T-shape VOF can be enforced as a combination of Alignment and Equal Spacing VOFs for the children and an Equal Spacing VOFs for the leftmost and rightmost children and parent.

- *Hub Shape:* The nodes should be placed radially equidistant on a circle. A central node may optionally be specified by the user; if none is specified, a phantom node is added at the center. Springs are placed between neighbors on the perimeter and between the center node and each perimeter node with rest length equal to the calculated average radius.

In addition, GLIDE introduces a new VOF to aid the user in interacting with the system. The following VOF can be used to gain absolute control over the fine-grained position of nodes.

- *Anchor:* A node should be located at the current position regardless of what other forces in the physical simulation may be acting on it. Although this may be thought of as infinitely increasing the mass of this node, the Anchor VOF is implemented by calculating all forces as if under normal conditions (so that forces are still appropriately placed on other nodes), and then ignoring anchored nodes when node positions are updated.

Although the computer cannot move anchored nodes, a person may still move them using the mouse. A useful technique is for a user to anchor two nodes to further constrain a second VOF. Under a Hub Shape VOF, for example, the radius of the hub is determined by calculating the average distance between the center node and each node along the periphery. To obtain a specific radius, the user could anchor the center node along with one node on the periphery. The user can then resize the hub by moving either of the anchored nodes.

Finally, GLIDE also provides a single *diacritical* VOF. A diacritical VOF does not provide any constraint on the diagram, but merely augments the diagram with additional graphic elements tied to aspects of the diagram layout. The diacritical VOF implemented at present is the Frame VOF:

- *Frame:* The bounding box of the set of nodes is demarcated with a drawn frame. As the nodes participating in a Frame VOF move, the frame repositions and resizes itself accordingly. The user controls such properties of the frame as its color, padding (distance added to the bounding box before being drawn), and whether the frame is drawn as an outline or filled rectangle. The Zone VOF example in Figure 3.1 is illustrated using a Frame VOF.

The Frame VOF can be used in conjunction with a Zone VOF, but is distinct from it. The topology of a graph often includes enclosures; a Frame VOF can be used to instantiate this component.

3.4.2 Constraint Satisfaction

The fundamental low-level constraint mechanism is a spring that obeys Hooke's Law;[2] graph nodes move according to the forces acting on them, which result from the springs attached to them. A mass-spring model for graph drawing was first proposed by Eades (1984), but in his and most subsequent systems, the spring forces correspond to topological or geometric properties of the graph. In the GLIDE system (and in the system of Dengler, Friedell, and Marks (1993)), a more general notion of spring force is used. VOFs and syntactic constraints are converted to spring forces as described in the previous section.

Our system uses a physical simulation as a constraint-satisfaction method. In addition to spring forces, all nodes are subject to a global friction force for stability. This acts as a form of damping, which prevents the system from oscillating wildly. Direct user manipulation is a second mechanism for positioning nodes, although it is not treated as a force by the physical simulation. The result of user movement is to directly reposition the node. As this occurs *during* the physical simulation, however, the system will immediately reflect the updated position. The spring forces are recalculated by the computer, giving the appearance that the nodes have had new forces applied to them.

When GLIDE is first started, the physical simulation is at rest. The addition or deletion of VOFs, nodes or edges, as well as direct object manipulation by the user all cause the simulation to begin running. It runs until the system reaches a stable configuration, as determined by a combination of total elapsed simulation time, and kinetic energy present in the system; the simulation is guaranteed to run for some minimum time in order to ensure that the forces in the system have had time to affect the masses to which they are attached. The size of the time step in the simulation controls the animation presented to the user, as well as the frequency with which the user can interact with the layout. For each time step, the system calculates the forces on each nodes, which is used to update each node's momentum, which in turn is used to determine the new position for each node. The friction applied at each iteration is determined by the maximum spring constant value for active springs in that iteration — it is intended as an approximation of critical damping.

Our first attempt at constraint satisfaction based on optimization was not as successful. The approach was similar to that of Kosak, Marks, and Shieber (1994), utilizing an objective function which included a component for each VOF type. The system calculated the local minimum for this function using Powell's method (Press, 1988), a traditional numeric optimization technique. The behavior of the system was not intuitive to people; they were unable to predict the impact their actions would have. As this intuition is an integral part of a collaborative system we turned to the mass-spring and physical simulation approach, which is currently implemented in GLIDE.

[2] Hooke's Law states that strain, the ratio of the change in length to the original length, is proportional to the stress that produces it. (*"Ut tensio, sic vis."* — *"As the elongation, so is the force."*)

3.4.3 Visual Presentational Issues

A tremendous amount of flexibility is achievable using the VOFs above, in tandem with adjustments to the graphical properties of nodes. For instance, *text labels* in diagrams can be implemented without additional infrastructure. A text label is merely a node with a transparent background and border. The label can be attached to other graphical objects using a Clustering VOF for instance, as in Figure 3.8. Nonetheless, in keeping with the spirit of the system providing high-level access to low-level infrastructure, and in contradistinction to traditional drawing and constraint-based editing programs that provide uniform but low-level access to their primitives, a notion of text label is provided in the GLIDE interface directly to allow generation of such nodes easily.

Similarly, a node with neither border, background, nor text is a kind of *phantom node* that can be useful as a control point for other objects (edges between nodes or the bounding box of Frame VOFs, for instance). It can serve as a way station between two other nodes. By adding Alignment VOFs between the phantom node and each node to which it is connected, the appearance of an orthogonal edge that turns at right angles is effected. Again, phantom nodes and orthogonal edges are made explicit in the interface, though they require no additional infrastructure for implementation. The use of hidden objects to control constraint-based layouts has been previously proposed. See Gleicher and Witkin (1994), for example, for a discussion of alignment objects.

In addition to the graphical components of the diagram itself — the nodes, edges, and frames — the interface uses visual means to present to the user the current set of VOFs that are being applied to the diagram elements. Each VOF instance is indicated by a graphical *indicator* in the display; the shape of the VOF indicator is similar to the icon on the corresponding push button for adding that VOF. As a person moves the mouse over a particular VOF indicator on the canvas, GLIDE highlights all participating nodes. Conversely, placing the mouse over a node will highlight the VOF indicators for VOF instances in which that node participates. Finally, placing the mouse over a VOF push button will highlight all VOF indicators of the particular VOF type. These mechanisms enable a user to easily determine the extent and impact of a particular set of layout constraints. The VOF graphics are intended to provide visual feedback to people; they are not, of course, included in the final output diagram.

GLIDE's animation is also an integral part of its visual feedback. Both user and computer actions are animated as the layout is displayed in the canvas area of the system. The physical simulation continually updates the position of the nodes in the layout. Although they are not graphically represented, the forces on a node become apparent to users as they try to move various nodes. As a user moves one node towards another, the two nodes repel, causing the node under user-control to seem heavier, moving more slowly, while causing the second node to move away. As the user adds VOFs to the layout, other forces are created. Giving nodes and edges "substance" through animation aids the user in understanding how to manipulate the layout to obtain desired results.

Finally, the use of a local constraint satisfaction mechanism that animates its actions provides users with a means for understanding the development of the layout. Dynamic stability is concerned with minimizing the difference between successive layouts of the graph (Tamassia, Battista, and Batini, 1989). Unlike many automatic graph layout algorithms in which new layouts are generated without taking current node positions into account, the mass-spring model used in GLIDE includes current node positions in determining new

positions. Furthermore, changes in node position are made gradually, permitting users to maintain their orientation with respect to the graph. Any large changes in successive layouts are primarily due to user intervention. Thus, the collaborative nature of GLIDE aids in the dynamic stability of the system as a whole.

In evaluating GLIDE as a collaborative system, we must examine the communication mechanisms between it and its user. GLIDE's animation, graphical indicators, and highlighting mechanism are at the heart of its visual feedback. User interaction utilizes simple interaction mechanisms based on a direct manipulation paradigm; the user can manipulate constraints, nodes and edges directly, guiding the system in its search for a locally optimum layout. The effects of their actions are immediately apparent through the animation of the simulation. Furthermore, although GLIDE is based on a physical simulation, it does not necessarily expect the user to understand the underlying physics; users can have strong intuitions about how the interface will behave based on analogy and experience with the physical world.

3.4.4 Implementation

The interface for GLIDE is implemented in Tcl/Tk (Ousterhout, 1994), extended by a C module to run the physical simulation for the mass-spring model. In order to support user interaction the application must poll for user actions such as the addition or deletion of nodes, edges or VOFs, or the repositioning of any object. Most user interaction causes a change in the mass-spring configuration, and in turn a change in state for the simulator. In theory the simulator should run until it reaches a stable configuration, restarting automatically when a change in state occurs. In practice, however, such an implementation is not easily achieved due to the Tcl/Tk internals. Therefore, in the current implementation the simulator runs continually; while in an equilibrium state it continues to calculate forces even though no change will be made in node positions.

Another important part of the physical simulator implementation are the spring constants. These variables control the stiffness of the springs, and in turn the amount of forces placed on the nodes. All of the VOFs except Cluster use fixed and equal spring constants. The Cluster VOF is given a lower spring constant; the result is that Clustering constraints in effect take lower priority than the others types of VOFs. This behavior matches most users' intuition about the system behavior. The result of applying both a Collinearity and Clustering VOF should be to have a set of nodes positioned so that they are aligned and close together. With an equal spring constant, the nodes would be unable to align themselves. In developing this system, we allowed users (i.e. the developers) to interactively set spring strengths. While this functionality provides the user with more flexibility, it also complicates the dialogue between the system and the user. As a result, GLIDE currently prohibits users from varying spring strengths.

The two syntactic constraints use a spring constant that is higher than that of the semantic VOFs. The stronger springs used to prevent overlaps induce greater forces, ensuring the syntactic validity of the diagram. As noted, when two objects are not overlapping, the spring constant goes to 0, in effect making the spring infinitely stretchable. In the implementation, we use a gridding system to track node and edge position. Rather than doing the n^2 comparison of testing nodes pairwise for overlap, the system only tests nodes that fall in the same region. A similar method is used to test for node-edge overlap. If (for whatever reason) users want overlaps to occur, they may achieve this effect using anchors.

GLIDE supports various modes of input and output. Users may save the structure and layout of a particular graph into an ASCII text file, which contains the Tcl script necessary to generate the layout. This file may be edited manually to make small changes in the graph. The GLIDE system can read one or more of these files as input, enabling users to modify an existing layout, or combine several graphs into one layout. Users may save an image of the layout, which is generated in a PostScript or GIF format.

As previously discussed, GLIDE makes use of graphical indicators for representing VOF instances, and phantom nodes as control points for other objects. These graphics are not necessarily intended to be included in the final drawing. In fact, some users may prefer not to see them during the layout and editing process. To accommodate both of these issues, GLIDE offers multiple views to users, enabling them to turn off graphical indicators, phantom nodes, or anchors on nodes in any combination. The highlighting mechanism described above is still available to users; mousing over a node will highlight the graphical indicators for any VOF instance that node participates in. Phantom nodes can be manipulated even when they are not visible.

We have included standard drawing package functionality in GLIDE. Users can change object characteristics (color, shape, fonts, etc), for entire classes of objects (by setting the default value) or for individual objects via a pop-up menu; each object instance has a specialized menu associated with it, providing easy access for users. GLIDE also supports methods for cutting and pasting objects in the layout, including VOFs. The semantics are such that all selected objects are copied if appropriate. If an edge is selected, for example, without both its parents being selected as well, it is not included in the copy. For a VOF instance, only a subset of nodes need be selected for the VOF to be copied. In the case of a T-Shape VOF, however, if the root is not included, the instance is not copied. Cutting and pasting cannot be used to apply a VOF to an existing set of nodes; adding a VOF is accomplished by selecting the nodes and the appropriate pushbutton, as described in Section 3.3.

3.5 Comparison to Existing Approaches

This section provides a comparison of GLIDE to drawing packages and systems from the graph drawing and constraint-based editing communities.

3.5.1 Drawing Packages

At first glance, commercial tools such as Claris Draw, MacDraw, or Microsoft's PowerPoint seem an attractive alternative for drawing small to medium sized diagrams. Most provide various polygonal shapes, text objects, and lines, and fixed templates such as trees. They support people in creating and moving various objects, and often employ snap grids, which let users place objects without the annoyance of having to work at a pixel level. In many cases, they also provide mechanisms to enable users to align objects in the diagram. These are typically one-time actions that can easily become "undone" by future user action. In general, such drawing editors are primitive, and place too much onus on the user. Although they enable a person to establish various relationships among objects in a drawing, they do not maintain these relationships during the evolution of the drawing. In addition to global control over the design, the user is responsible for *all* aspects of the layout process, which is obviously undesirable. There is only minimal collaboration between the user and

the computer in such systems — the snap gridding feature described in Section 1.3 is one simple example.

3.5.2 Graph Drawing Algorithms

Battista et al. (1994) compiled an annotated bibliography of over three hundred papers describing systems and algorithm for drawing graphs. Most of these papers cover algorithms for fully automatic graph layout. There is a considerable additional literature on graph-drawing algorithms, but it also mostly concerns non-interactive techniques for automatic graph layout (Tamassia and Tollis, 1994; Brandenburg, 1995; North, 1996). Various systems have been built that also use such automatic layout algorithms. DAG (Ganser et al., 1993), for example, is a system that draws directed graphs. It is available to users via an e-mail server, and returns output in either PIC or PostScript form; as a result, people cannot fine-tune the results to better meet their needs. Automatic algorithms and systems are inappropriate for use in a network-diagram design tool; they allow no user input in solving a problem whose aesthetic and combinatorial difficulty make it crucial that people be allowed to participate.

Several tools for the interactive editing of graphs have evolved from the graph drawing community. These include Edge (Newbery, 1988), daVinci (Frohlich and Werner, 1994), GraphEd (Himsolt, 1994), its successor Graphlet (University of Passau, 1997), and commercially available software from Tom Saywer (Tom Sawyer Software Corporation, 1991). Recently Bridgeman, Garg, and Tamassia (1996) have provided a web-based service to serve as a test bed for comparing different automatic layout algorithms. In general, these graph layout and editing systems include a limited graph editor, which support the user in modifying the syntactic structure of the diagram (adding and deleting nodes and edges), in fine-tuning node placement, and in editing various attributes of the graphical objects (color, shape, size, fonts, etc). Most of these systems then use automatic layout routines to determine a final layout. Such systems are useful for drawing larger graphs; people may not know the structure of a diagram and an automatic layout may help them make sense of the data, enabling them to determine a desirable organization for the diagram. In such cases, automatic layout often provides a starting point for exploring design alternatives. These editors, however, restrict node movement and are too inflexible for small and medium sized networks; they do not support users in generating aesthetically pleasing layouts. In some sense they are comparable to typical spreadsheet graphics packages in that they enable users to make sense of large data sets; they share the drawback that the computer limits a user to pre-defined styles, which may or may not be what the user had in mind. Most importantly, they do not support the specification of perceptual organization.

Marks (1991) addressed the importance of perceptual organization in his thesis in which he described ANDD, an implemented system for automatic layout of network diagrams, and introduced the notion of *visual organization features* (VOFs) which can be used to characterize the perceptual organization of graphs. Subsequently, Kosak, Marks, and Shieber (1994) incorporated VOFs into another system for network diagram layout that relied on a massively parallel genetic algorithm in which an objective function was specified to evaluate the quality of the layout. VOFs have also been incorporated into another fully automated graph layout system (Dengler, Friedell, and Marks, 1993) which utilized a generalized spring algorithm to solve them. In all cases, the VOF set to be applied was provided as part of the specification of the graph. In contrast, GLIDE enables users to interactively specify desired

VOFs; users may also interact with the *GLIDE* system to aid it in solving the constraints.

Bohringer and Paulisch (1990) suggested the use of constraints to enable users to provide semantic information to guide automatic layout algorithms. They extended the Edge system (Newbery, 1988) to provide a small set of low-level constraints, which users could apply interactively by filling in a form. In this system, constraints were specified in one step and then applied (as a result of the user pushing an apply button) as a second step. Likewise, as users removed constraints (by pushing a button for each constraint), they would again need to activate the constraint solver manually to generate a new layout. The three basic constraints types were for absolute positioning, relative positioning, and clusters; these are only a subset of the constraints supported by GLIDE. Each constraint instance could be assigned a priority by the user. The system utilized a global constraint satisfaction mechanism. In the case of inconsistent constraints, the system deactivated constraints based on their priority, attempting to keep higher priority constraints over those with lower priority; among inconsistent constraints with equal priority, the system would select randomly which constraint to deactivate. Although the authors note that deactivated constraints were ignored during the evaluation of the constraint network, there is no discussion of how or if a deactivated constraint becomes reactivated. To support users in debugging the constraints, the system provided a query button, which would indicate to users if a particular constraint were being met. The system integrated the use of constraints into the Sugiyama layout algorithm[3] (Sugiyami, Tagawa, and Toda, 1981). GLIDE improves upon this system by providing a larger set of high-level constraints to users, a more intuitive satisfaction mechanism, and a computer-user interface with easier interaction mechanisms.

3.5.3 Constraint-Based Editors

Constraint-based drawing editors are perhaps one of the oldest ideas in computer graphics. Sutherland's Sketchpad (1963) introduced the notion of interactive graphics and incorporated the ideas of direct manipulation, and constraint-satisfaction. Subsequent research led to systems such as ThingLab (Borning, 1979) and Juno (Nelson, 1985); these early systems provided a limited set of graphic primitives and constraints. Juno2 (Heydon and Nelson, 1994) expanded upon this work. This system is a double-view drawing editor, which presents the user with a WYSIWYG graphical representation, along with a text-based version of the program used to generate the image. Modifying either view updates the other accordingly. This approach is appealing because it provides an intuitive interface for both programmers and non-specialized users. Converge (Sistare, 1990; Sistare, 1991) is a system in which three-dimensional constraints and the geometry to which they are applied are presented together in a single graphical framework; it presents constraints to the user by superimposing graphical icons onto the geometry. Many other systems have employed constraint-based techniques. The work of Hower and Graf (1995) provides a comprehensive bibliographic survey of constraint-based techniques and their application to a variety of tasks such as computer-aided design, graphics, layouts and user interface design.

Because they maintain relationships by enforcing various constraints throughout the editing process, constraint-based editors have the potential for a more balanced collaboration than traditional drawing packages that typically only provide one-time actions.

[3] This algorithm is used for laying out directed graphs by introducing dummy nodes to split edges so that nodes can be assigned to levels in a hierarchy.

Although constraint-based techniques have been utilized in numerous systems, they have enjoyed only limited success. This is in part due to the difficulty in creating, solving and presenting constraints to the user (Gleicher and Witkin, 1994). We consider three key questions surrounding the design of constraint-based system:

- Which constraints will be supported by the system?
- How will the constraints be established?
- What happens in the case of conflicting constraints?

Constraint-based systems are often overly general and complex, making them cumbersome for specific tasks, such as drawing network diagrams. In most systems, constraint selection (i.e. the set of constraints the system will support) is based on orthogonality and coverage, as opposed to convenience for the particular higher-level task at hand. GLIDE improves upon these systems by providing a small but powerful set of constraints specialized to support the drawing of graphs, and simple interface mechanisms for the user to create, view, manipulate, and remove constraints. Note that although GLIDE provides specialized constraints for network diagram layout, it does not provide guidance to the user in applying them appropriately. Such considerations typically fall in the domain of graphic design. The division of labor in GLIDE relies upon humans for this particular expertise. An interesting extension might be to incorporate design advice into the system. This modification would, however, change the very nature of the collaboration between the computer and its user.

Although three previous systems (Marks, 1991; Kosak, Marks, and Shieber, 1994; Dengler, Friedell, and Marks, 1993) incorporated VOFs into their diagrams, these were automatic systems; the constraints were not specified interactively by users. GLIDE is the first system to support users in specifying the visual organization of a diagram. The system provides a natural and powerful vocabulary whereby users can easily express the desired perceptual organization of graph layout. The VOFs provide for proximity relationships, alignment, axial and radial symmetry, sequential ordering of a layout, and other commonly used patterns found in graph layout. In addition, GLIDE introduces the use of a diacritical VOF, which provides additional graphic support to users, along with a meta-VOF, which enables users to better guide the system in articulating the layout of the graph.

People often have a difficult time understanding how a constraint-based system works. Because the mathematical methods used for constraint satisfaction are not necessarily easily understood, and due to the complex interaction of some constraints, it is often unclear to a user how and why the system moves from one configuration to another. Furthermore, some sets of constraints are unsatisfiable. Many methods do not degrade gracefully under such conditions, and are unable to show a user how to remedy the situation. On the other hand, when a design is under-constrained, multiple layouts will satisfy the given set of constraints. Without user guidance, the system would need to guess which one a user wanted.

Nonetheless, constraint-based systems, if extended to allow collaboration on the setting up and solution of the constraints are a viable alternative for graphic design tasks such as network-diagram layout. Indeed, the paradigm that we propose in this thesis is just such an alternative.

3.6 Summary

Network diagrams are a common form of informational graphic and constitute an area in which it may be difficult for a user to define the aesthetic qualities desired in the final design. In particular, perceptual organization is a key for human understanding of such diagrams. Previous work on graph drawing has focused on syntactic criteria and layout aesthetics in automatically laying out a graph. Because most interactive graph drawing tools rely primarily on these automatic algorithms, they do not support users in specifying perceptual organization information, which is an integral part of providing semantics to the diagram.

GLIDE is unique in its approach to constraint-satisfaction. Its use of a generalized mass-spring simulation which emphasizes local constraint satisfaction and whose behavior is intuitive and predictable to users, rather than one better at finding global solutions, is an integral part of its collaborative nature. GLIDE is not intended to be good at globally satisfying constraints by itself. Rather, it is intended to provide an interface that allows a useful and efficient collaboration between user and computer in solving layout problems. For this purpose, predictability, simplicity, and compelling animation are far more important than global optimality.

The basic concept underlying the GLIDE interface — tight collaborative interaction between user and computer to solve an optimization problem, with the computer performing local optimization and the user responsible for global control — has resulted in a graph editor that enables users to draw small- and medium-sized graphs easily. The mass-spring simulation approach may be applicable to other layout, drawing, and design tasks. In implementing such a system for other domains, the challenge would be in identifying the perceptual organizations that are relevant to the particular task at hand.

Chapter 4

Design Galleries

Viewed abstractly, all algorithms in computer graphics map input parameters to output values. For example, image rendering maps scene parameters to output pixel values; animation maps motion control parameters to trajectory values. The problem of parameter specification — finding a good set of input parameters to generate desirable outputs (i.e. realistic images or compelling animations) – is an important but challenging problem in incorporating computer graphics algorithms into interactive systems. It is difficult for people to understand the impact and interaction of these parameters. Finding input parameters that yield a desirable output is difficult and tedious for many rendering, modeling, and motion control processes. This is not surprising; these mapping functions are usually multidimensional, nonlinear, and discontinuous. A system that solves this problem would help expert and novice users alike.

In this chapter we describe Design Galleries[1](Marks et al., 1997), a novel approach to parameter specification that embodies in its computer-user interface the collaborative framework proposed in this thesis. It exploits a computer's superior computational power to explore the space of design alternatives, and a user's ability to incorporate aesthetic criteria into evaluating and organizing competing designs. The Design Gallery methodology outperforms other approaches to parameter specification by handling the problems of *high computational costs* and *unquantifiable output characteristics*. It supports users in both *exploratory* and *instantiative* graphical design tasks. We focus on a novel method for arrangement of large data sets of graphics and interaction techniques that support users in browsing design spaces.

4.1 Introduction

Manual parameter tweaking has long been a bane for computer graphics. Searching for a specific image can be time-consuming and tedious, and is often a case of trial and error; in many ways it is like searching for a needle in a haystack. A person supplies the input parameters, leaving the computer to generate the corresponding graphic. A typical scenario

[1]The material presented in this chapter is based upon a research project comprising thirteen people geographically distributed over five locations, and is included as a good illustrative example of the collaborative framework proposed in this thesis. An overview of the approach is given, with emphasis placed on this author's contribution — the arrangement and interaction techniques as they are incorporated into a user interface for a variety of application areas.

would be for a person to use a simple editor to set the various parameters, each represented by a control such as a slider or knob. By way of analogy, consider a television set that has four parameters to control image quality: hue, color, brightness and sharpness. Most people have a hard time adjusting just these few parameters to get the desired picture; it is unclear to most people how changing one parameter will impact the others. Now imagine having to wait one minute every time one of the knobs is tweaked before seeing its effect instead of the continuous, real-time feedback that TVs provide. Parameter specification would be horribly slow and frustrating. Such is the case with many computer graphics applications. Time delays of minutes, hours, or even days are not uncommon. Finally, a TV has only four parameters — computer graphics algorithms can have many more. As the number of parameters grows it can become increasingly difficult for a user to predict the effects of adjusting particular parameters and combinations of parameters. The notion of getting the computer to assist actively in setting parameters is therefore appealing.

Inverse design and interactive evolution are the primary previous approaches to computer-assisted parameter specification. Inverse design is an automated approach in which a person provides an objective function that encapsulates the desired characteristics of a graphic; the computer then optimizes this function, producing the corresponding graphic. In many cases, however, a person does not always know what qualities will make one graphic preferable to another. This problem of *unquantifiable output characteristics* makes inverse design impossible for many design tasks. Interactive evolution, a semi-automatic approach to parameter specification, solves this problem by employing the user as a dynamic function; a person picks and chooses among a set of graphics, indicating which ones are preferred. Many computer graphics algorithms, however, do not run in real time — they may require minutes or hours to generate a graphic. Because of these *high computational costs*, it is impractical to use interactive evolution to assist in parameter specification.

Our approach to computer-assisted parameter setting, which we call Design Galleries (Marks et al., 1997), presents the user with the broadest selection, automatically generated and organized, of perceptually different graphics or animations that can be produced by varying a given input-parameter vector. The general approach is to have the computer generate a diverse selection of graphics in batch mode, as a preprocessing phase, which the user can then browse through in real-time, viewing and selecting graphics interactively. Such a division of labor overcomes the problems of unquantifiable output characteristics and high computational costs that are associated with existing approaches to computer-assisted parameter selection; a comparison of Design Galleries to existing approaches is given in Section 4.5.

Central to the implementation of a Design Gallery is the presentation of design alternatives. A primary goal of the Design Galleries is to provide access to a large set of graphics so that a person can get an understanding the overall design space. The division between computer and user in presenting and exploring the design space is an integral part of the collaborative paradigm used to design the computer-user interface. However, the greater the size of the set, the more difficult it will be for someone to browse effectively. We call the problem of organizing and presenting a set of output graphics for easy and intuitive browsing by the user *arrangement*; it is discussed in more detail in Section 4.4.

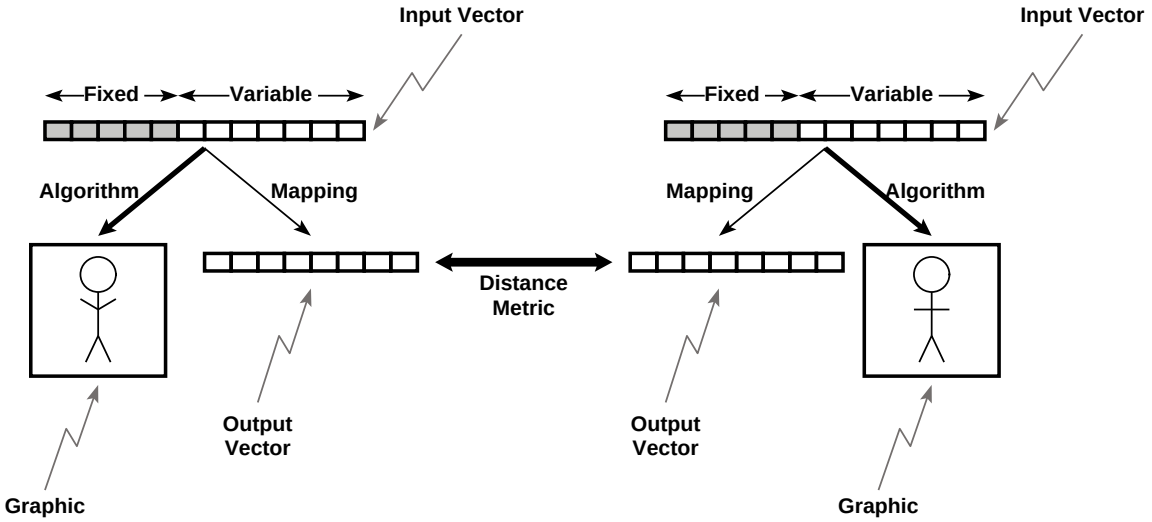


Figure 4.1: Components and interactions of a Design Gallery Application.

4.2 Design Gallery Methodology

Generally speaking, the Design Gallery paradigm can be applied to any graphics problem that involves setting parameters that make up an *input vector* of variables whose specification gives rise by an *algorithm* (typically involving heavy computation) to an *output graphic*, and where the judgment of output graphic quality is subjective, informal, or otherwise difficult to define formally. The paradigm requires a method of characterizing the output graphics with an easily computed *output vector*; the output vector is generated using a *mapping* function (which will involve less computation than the algorithm used to generate the full graphic) from the input vector to the output vector. The output vectors are such that a *distance metric* on the space of output vectors approximates the perceptual similarity of the corresponding output graphics. A *dispersion method* is used to efficiently find a sample of input vectors that well cover the space of output vectors (hence output graphics). The selected input vectors are mapped to output graphics during a pre-processing phase before the interactive session with the user begins. The selected graphics are presented to the user through a perceptually reasonable *arrangement method* that makes use of the distance metric. The user can then effectively and intuitively browse through the range of final graphics.

Figure 4.1 provides a schematic representation of six of these eight basic building blocks (input vector, output graphic, output vector, algorithm, mapping, and distance metric — dispersion and arrangement are discussed below) and their interactions for the Design Gallery approach. The set of input vectors delimits the space of possible (output) graphics. A single input vector specifies the initial data set and a particular set of parameters — some portion of the input vector may be held constant. Each input vector may be used to generate an output graphic (e.g. an animation or an image) using a (potentially computationally expensive) algorithm, or an output vector using a less expensive mapping function. The output vectors act as compact representations of the output graphics; they capture the salient features of the graphics, and are used to evaluate and compare alternatives during

the dispersion phase. The mapping function used to transform input vector to output is typically a light-weight computation in comparison to the algorithm used to generate the full output graphic. In Figure 4.1 arrow width is used to indicate computational expense. The distance metric is used to compare two output vectors and approximates user perceived similarity in the output graphics; smaller distances correspond to a higher similarity in the output vectors, and in turn, a greater perceptual similarity between corresponding output graphics.

Given building blocks, the dispersion component searches through different parameter values for the input vector, typically on the order of millions of combinations, using the mapping function to calculate the corresponding output vectors. Using the distance metric, it can compare output vectors, expanding the working set to get good coverage of the design space. The result is a maximally distributed set of output vectors (which correspond to a set of output graphics). The arrangement component takes as input this set of well distributed output graphics and attempts to organize and present the results to the user. The focus of this research has been on the arrangement and interaction techniques as they are incorporated into a user interface for such a system. This collaborative interface and its application in a variety of domains is discussed in more detail in the following sections.

4.3 Sample Application: Medical Imaging

We have applied the Design Gallery approach to several common parameter-setting problems: light selection and placement for image rendering, both model-based and image-based; opacity and color transfer function specification for volume rendering; and motion control for particle-system and articulated-figure animation. For illustrative purposes, we will briefly describe a sample Design Gallery application for medical imaging. All the applications are described in more detail in separate papers (Marks et al., 1997; Kang et al., 1995), and additional example interfaces given in Appendix B.

Volume rendering is a means for visualizing large data sets. It is one of the techniques used in the field of scientific visualization; as a tool, it is useful for both image understanding and generation. As used in medical imaging applications, volume rendering is typically applied to data sets that define attributes of a model not only at the surface, but inside as well. Such data sets can be generated from CAT-scan (computerized tomography), PET-scan (positron emission tomography), MRI (magnetic resonance imaging), and ultrasound procedures. Volume rendering enables viewing such data as a three-dimensional field, rather than as individual planes (which is how the data is gathered). In addition, a single data set may be rendered into hundreds of different images, revealing different components of the structures represented by the data. This is to be distinguished from the different view points that might be used to render the image.

One algorithm for volume rendering is based on ray tracing; it makes use of two transfer functions to determine the color and opacity for the voxels (three-dimensional pixels) in the data set. For our medical imaging application the data values are pre-segmented into four disjoint subranges: one each for air, fat, muscle and bone. Standard colors are used to represent the different tissue types, and so the color transfer function is held constant. Changes in the opacity transfer function will result in images that reveal the underlying structural elements to varying degrees. Figure 4.2, for example, shows the same data set rendered using different opacity transfer functions.

The application we consider here is a volume rendering experiment using a data set

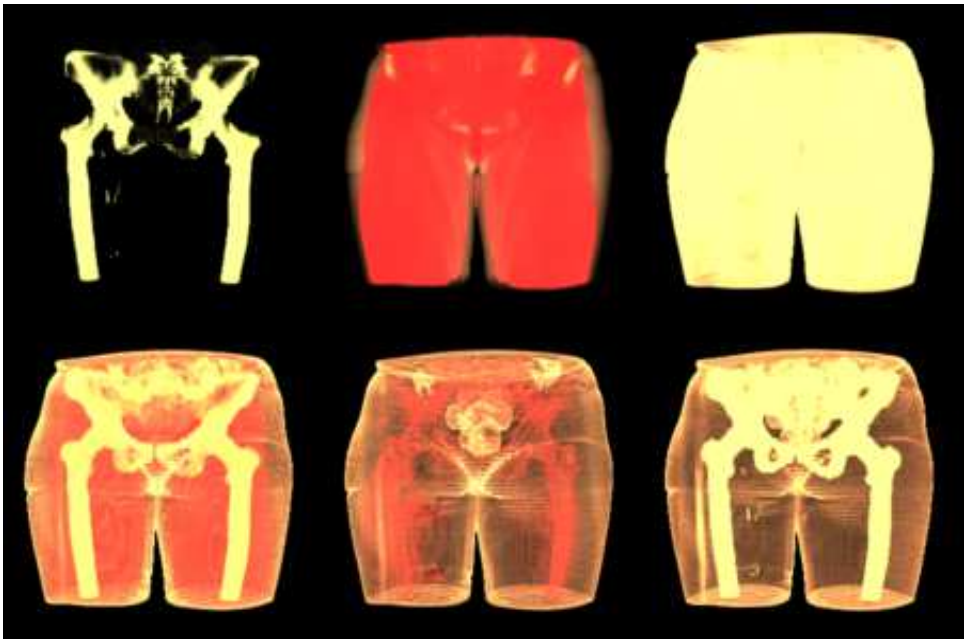


Figure 4.2: Human hip data set rendered using different opacity transfer functions.

for a human pelvis. The fixed portion of the input vector includes the 3D CAT-scan data of a human hip and the color transfer function. The variable component of the input vector is for the opacity transfer function, which is parameterized by a polyline, containing the y-coordinates for twelve control points; the x-coordinates are held fixed. Varying the opacity transfer function will cause the different tissue types to be rendered using different transparencies. Because changes to the transfer function will generally affect many pixels throughout a volume-rendered image, we need only include a handful of pixels in the volume-rendered image in the output vector. After some experimentation (as discussed by Marks et al. (1997)), we settled on using eight pixels, selected manually. Dispersion on the basis of eight well-chosen pixels seems to produce excellent dispersion of complete images at a much reduced computational cost. Representing all of their YUV color space values results in twenty-four values in the output vector. Mapping is done by volume rendering of the eight sample pixels. Standard Euclidean distance is used as the distance metric for comparing vectors in the output space.

The dispersion heuristic uses an evolutionary strategy that adapts its sampling over time in response to what it implicitly learns. It starts with an initial set of random input vectors. These vectors are then perturbed randomly. Perturbed vectors are substituted for existing vectors in the set if the substitution improves dispersion. The measure of dispersion used is nearest-neighbor Euclidean distance in the space of output vectors. Improvement is rapid at first. However, the rate of improvement drops quickly. After considering millions of candidate images, the dispersion procedure returns 256 input and output vectors, distributed throughout the design space. The system renders the full-sized image ($m \times n$ pixels) corresponding to each input vector using a volume rendering algorithm.

Arrangement is discussed in detail in the following section.

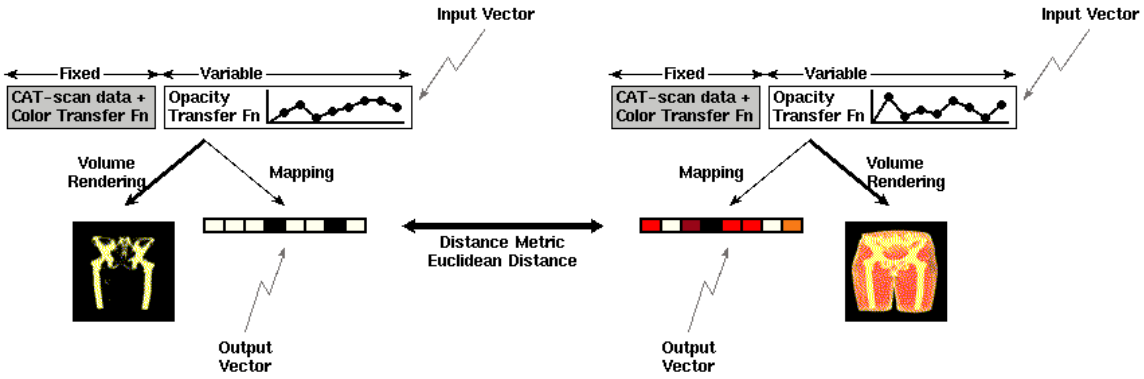


Figure 4.3: Instantiated components for a medical imaging Design Gallery application.

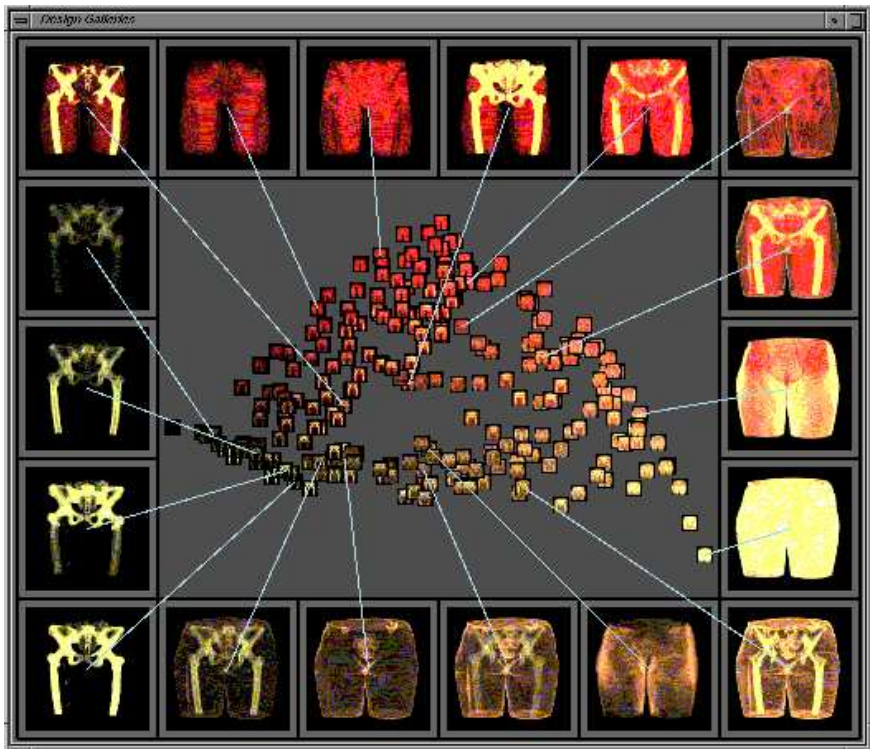


Figure 4.4: A Design Gallery for medical imaging with different opacity transfer functions.

4.4 Arrangement and System Design

Presenting data in an organized manner is an important part of information understanding. The primary goal of the arrangement component is to make it easy for people to navigate through the design space. We would like users to explore the display intuitively, relying on visual comparisons in evaluating and examining candidate graphics. A great deal of research has been done in the field of graphic design (Bowman, 1968; Dondis, 1973; Tufte, 1983; Tufte, 1990; Tufte, 1997); its goal is the design of effective visual communication for information presentation. Typical considerations include form, spatial organization, and composition. Within the field of user interfaces, Marcus (1983; 1995; 1992) has applied many of these ideas to the design of effective displays and usable interfaces. He presents basic principles which can be used in GUI design (Marcus, 1995). Based on his characterization, we present four considerations for the arrangement component of a Design Gallery application:

- *Icons*: What representation should be used to present the output graphics to users?
- *Layout*: Given a set of icons, how should they be arranged on the screen?
- *Navigation*: How should the user move through the design space?
- *Interaction*: What methods does the system provide to the user for manipulating the space and graphics within it?

4.4.1 Icons

The representation used to present output graphics to users will be called an *icon*. One possible icon choice would be to use the output vectors used by the dispersion heuristic. This mathematical representation of the graphic, however, would not provide users with much information, particularly as people are interested in the perceptual qualities of the graphic. Another option would be to use a number or a simple point to represent each graphic. Again, this would not aid people visually. An obvious choice would be to use the output graphic itself. Screen space, however, limits the number of full-sized graphics that the system can present to users at one time.

The icons used in the Design Gallery applications are *thumbnails* (e.g., 32×32 pixels and smaller) which are small, low-resolution copies of the full-sized output graphic. Although thumbnails provide less detail than the corresponding full-sized images, they still supply important information to the user; visually scanning the area enables people to identify patterns in the layout. Due to their small size, many more thumbnails may be displayed simultaneously than larger images. The thumbnails are arranged in the *icon display panel* (the inner region in Figure 4.5). The surrounding *gallery* (the outer region in Figure 4.5) is used to store full-sized images of interest to the user.

For the animation applications, static images (both thumbnail and full-sized) are also used for the icons and gallery graphics. Animating all the images simultaneously has two drawbacks. First, it is not clear that a user could make sense of that much information in that particular format. Second, due to computational and hardware restrictions, the animations would most likely not run at their nominal speed. In some cases, for example the particle system described in Appendix B.4, playback rate is an important component

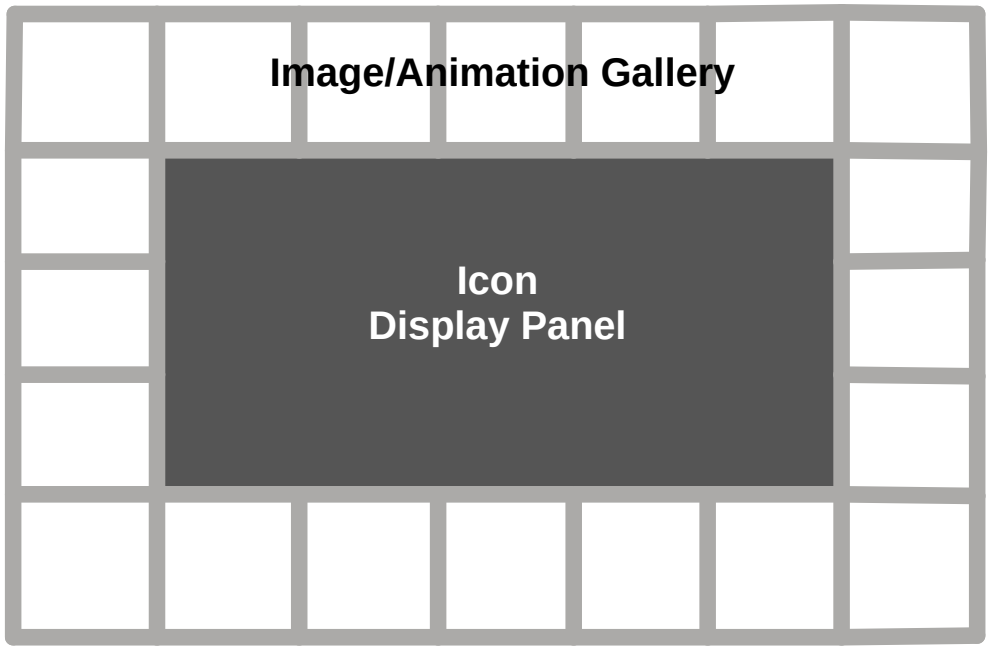


Figure 4.5: A schematic of the Design Gallery user interface.

of the animation. The static images are generated by compositing multiple frames from the animation; while the resulting image does not look like any single frame in the animation, it does provide users with a good intuition for the behavior of the animation over time.

4.4.2 Layout

The layout of the icons determines the position of the icons on the screen. Possible structures for the layout include grids, trees, graphs, tables, and lists, in either two or three dimensions. An important goal is to preserve the structure information from the multi-dimensional space. We would also like to enable users to make quick visual comparisons between graphics. Thus, the layout should be organized according to the perceptual similarity of the graphics.

Under the current Design Gallery framework, icons are arranged in a two-dimensional layout using a proximity metric. By generating a two-dimensional embedding of the space, the computer provides users with an overview of the design space. The layout of icons is accomplished with a form of multi-dimensional scaling. The original problem formulation due to Kruskal (1977) is as follows: Given a set of n objects, a dissimilarity matrix, and a target dimension d (in this case two) the problem is to find a mapping of the n objects to a set of n points in d-space, such that the distance between points in the new space is as close as possible to the original dissimilarities.

Our implementation is due to Torgerson (1958). Let d_{ij} be the distance between output vectors i and j, and let δ_{ij} be the distance between their corresponding icons in a two-dimensional embedding. We use Torgerson's method to compute the icon positions so that $\sqrt{\frac{\sum_{i,j}(\delta_{ij}-d_{ij})^2}{\sum_{i,j}\delta_{ij}^2}}$ is minimized. To make the best use of the available display space, we then rotate the computed embedding to align its first principal component with the horizontal axis of the icon-display window. The resulting layouts will not be without anomalies — as we are using it, multi-dimensional scaling is a projection from a high-dimensional space onto a two-dimensional space, and this cannot be done without loss of information. The use of an approximate layout is accommodated by the collaborative interface design; the layouts do reflect the underlying structure of the output vectors well enough to allow effective browsing. One important practical detail: since full-size versions of all the images returned by the dispersion procedure must be rendered anyway, it is convenient and better to compute distances on the basis of these full-size images in the arrangement phase, instead of the output vector used in the dispersion phase.

Prior to the multi-dimensional scaling layout method currently used in the Design Gallery interface, we explored the use of a dynamic spring model based on the physical simulator described in Chapter 3 for the arrangement of the icons in the interface. We defined springs between each pair of icons in the system using the dissimilarity matrix to determine the rest length of each spring. This technique was particularly sensitive to the initial placement of the icons in the two-dimensional space. Animating the layout process did not help users to understand how or why the layout was generated. Many anomalies were present in the resulting layouts; although the mass-spring method is good for local optimization, it is not useful in generating globally optimum layouts automatically. We attempted to increase user participation in the layout phase by permitting a user to indicate one or more icons of interest, which would increase the corresponding spring strengths. The resulting system did not make good use of the available screen space and was slow in generating layouts. The multi-dimensional scaling approach, combined with appropriate

navigation techniques, is more appropriate for icon layout. An alternative arrangement approach based on a hierarchical partitioning of the graphics is discussed by Kang et al. (1995).

4.4.3 Navigation

Navigation mechanisms permit people to explore the design space, and in the case of Design Galleries, move through the icon display area. It is important to provide users with a sense of perspective and context as they explore this space. People should have access to both local and global views, with varying levels of detail as needed.

In a Design Gallery application, the user is initially presented with the entire set of icons, arranged as described above such that visually similar icons are closer together, and dissimilar icons are further apart. The set of icons is automatically scaled to fill the entire space of the icon display area. In the resulting layout, however, there is no guarantee that icons will not overlap. Since perceptually similar icons are grouped together, clusters are similar, and overlaps result in only a minimal amount of information loss. The user may select an area of interest using the standard mouse technique of region-dragging. The system then utilizes two processes to refocus the display for the user:

- *Panning*: The system calculates the bounding box of the selected icons, and recenters the display on the new center.
- *Zooming*: The system rescales the bounding box to utilize as much of the center display as possible. The scaling is done uniformly in both dimensions, and is limited by the larger dimension.

As the user moves in to examine an area more closely, the system will first pan and them zoom. As fewer images are being displayed within the display panel, the number of overlaps decreases. Zooming in removes visual clutter and enables users to see icons that may have been previously obscured. In moving out, the system reverses the process. Zooming out condenses the icons, which is followed by panning to recenter the display. The system smoothly animates both phases of the navigation; this behavior enables users to keep track of where they are within the larger context of the design space, while examining particular regions of interest in more detail.

The zooming mechanism of current Design Gallery interfaces does not resize the icons as the magnification level changes; it only respaces the given icons. This is in large part due to implementation details. Although a scaling operator is available in the Tcl/Tk environment, it does not apply to imported images. One alternative would be to generate thumbnails on the fly (or to have a set pre-computed and interpolate amongst them as the scale changed); computation costs (and storage constraints) make this unrealistic.

The system currently supports two-dimensional panning. Extending the layout to three dimensions might prove useful for the arrangement component, and provide additional visual information to users. Rather than using a true three-dimensional environment, such as OpenGL, we could implement a three-dimensional effect by exploiting the illusion of depth that is possible on a computer screen; smaller thumbnails would appear to be further away. Similar problems to the magnification of thumbnails associated with zooming would need to be considered in order to support three-dimensional panning.

4.4.4 Interaction

Interaction mechanisms provide users with methods for manipulating the space and graphics within it. Such functionality is related to, but distinct from, navigation. For example, in a text document retrieval system, clicking with the mouse could retrieve title, abstract or full-text for a given document. Interaction techniques can also provide landmarks for users, which are helpful for determining their present location within the space.

Design Gallery applications rely on the mouse for user input. A user may move the mouse over any of the icons in the display area; clicking on a particular thumbnail will bring up a full-size image of the corresponding graphic. Releasing the mouse causes the graphic to disappear. Alternatively, images can be dragged to the gallery (where they will snap into the closest pane) and arranged at the user's discretion. Dragging and dropping a gallery image into the center region will remove it from sight. In Figure 4.4, a user-selected set of images is shown in the surrounding image galleries. The lines connecting images with their thumbnails are only included to give some indication of how images congregate in the thumbnail display; the association between thumbnails and images is done by dynamic highlighting in the actual user interface. Mousing on an image in the gallery highlights its associated icon, and vice versa. In this way, the user can group graphics of interest in the gallery, and determine their position in the global display with this interactive highlighting mechanism. If the icon to be highlighted is at the bottom of a stack of icons, the system will temporarily float the icon to the top. Note that the stacking order of the icons is arbitrary, and independent of their visual content.

4.4.5 Application-Specific Functionality

The computer-user interface is implemented using Tcl/Tk. This environment enables the Design Gallery interface to be customized to accommodate a person's preferences; users may specify colors for various portions of the interface, dimensions of the gallery, etc. More importantly, application designers can easily extend the interface, adapting it to various application areas. The volume rendering example given in Section 4.3, for example, includes functionality that permits users to view the opacity transfer functions using `gnuplot`. Animations can be shown in the gallery when appropriate; the two-dimensional pendulum described in Appendix B.2, for example, uses a C back-end to render the animations in real-time. The particle systems, see Appendix B.4, are computationally more expensive, and so MPEG movies are used in place of real-time animation. For each application, however, the basic interface is the same; additional components can be loaded to support application-specific functionality as needed.

4.5 Comparison to Existing Approaches

In this section we describe two computer-assisted approaches for the problem of parameter setting: inverse design and interactive evolution. We also survey related work on the problem of arrangement from the field of information visualization.

4.5.1 Parameter Specification

Inverse Design

One computer-assisted methodology for parameter setting is inverse design, which exploits the notion of design by optimization. This approach is well-suited to problems with a large number of candidate designs that can be enumerated and evaluated automatically; such evaluation, however, is often problematic. An objective function (a mathematical specification of the desired characteristics of a graphic) is supplied that the computer can use to rate the quality of a candidate graphic. The system searches for parameter settings that optimize the function, automatically evaluating and ranking candidate graphics. Inverse design has been successfully used in a variety of applications, including label placement (Christensen, Marks, and Shieber, 1995; Edmondson et al., 1997), motion-synthesis (Liu, Gortler, and Cohen, 1994; Sims, 1994; Tang, Ngo, and Marks, 1995; van de Panne and Fiume, 1993; Witkin and Kass, 1988), lighting specification (Kawai, Painter, and Cohen, 1993; Poulin and Fournier, 1992; Schoeneman et al., 1993), and volume rendering (He et al., 1996). However, this automated method of parameter specification provides little support to users for defining objective functions. There are two reasons that inverse design is problematic: deciding what characteristics to include in the objective function, and determining how to specify a set of characteristics as part of an objective function. We will examine each of these in more detail.

First, people may not know what characteristics they would like to include in a graphic (i.e. to include in the objective function). This problem of *unquantifiable output characteristics* results from the fact that even though desirable graphics may be readily identified by inspection, it may not be possible to specify a priori the qualities that make them desirable. This rules out the use of inverse design for parameter specification and indicates that the approach is not well-suited for browsing systems in general. A possible extension would be to allow users to easily view multiple candidates for a single objective function, or to permit users to specify multiple objective functions. This would not, however, remove the problem of how to specify the objective function in the first place or the problem of unquantifiable output characteristics. In addition, because current inverse design systems provide only a single solution, they would not be able to organize multiple candidate solutions in a manner that would make them easily accessible to users.

Second, once a person knows what qualities to include in the objective function, it is still difficult for a user to define the objective function. Inverse design replaces the parameter specification problem with an objective function specification problem. It is important to note that the choice of objective function may affect the aesthetics of the resulting graphics, the quality of the solution, and the efficiency of the search method for optimizing the objective function, and is therefore an important consideration. In general stating mathematically the desirable properties of an animation or abstract image is very difficult. Even for graphics tasks with clear principles and conventions that can potentially be converted into objective functions in a straight forward manner, developing a good objective function may still be hard. In his thesis, Christensen (1995) remarked, "Coming up with an appropriate objective function for a general label-placement problem (that is, one that includes point, line and area features) is a difficult task." Furthermore, aesthetic criteria are often not as easily formalized in a discrete manner. Christensen (1995) addressed the problems of label placement and motion-synthesis using inverse design. Specifying objective functions for animated mass-spring models was the more difficult and time-consuming task. Although

the resulting animations were visually compelling, they were seldom (if ever) achieved on the first specification of an objective function. As the generated animations behaved in undesired ways, the objective function was refined to further constrain the behavior of the character. This iterative refinement process was hampered by the high computational costs associated with calculating the animation trajectories. Thus, inverse design is inappropriate for novice users who lack the expertise necessary to define objective functions, and is often problematic for expert users.

Inverse design is only feasible when the user can articulate and quantify what is desired in a graphic. The Design Gallery approach handles the problem of unquantifiable output characteristics by removing these restrictions. By generating a representative set of graphics from the design space, a Design Gallery application enables users to browse a space, determining the available possible graphics without having to specify an objective function. In addition, because Design Gallery systems do not require users to have domain specific knowledge, they may be used by both expert and novice users.

Interactive Evolution

Interactive evolution falls into the middle ground between manual and automatic approaches. The computer is responsible for exploring and presenting candidates from the design space. The user acts as a dynamic objective function, indicating images of interest to be used in guiding the exploration. Based on the user's input, the computer generates a new set of candidates using a variety of genetic algorithm operators. Traditional genetic algorithms required the specification of survival fitness criteria to be evaluated by the computer — this is just another form of a pre-defined, user-specified objective function. Interactive evolution replaces the static fitness function with dynamic user interaction to evaluate alternatives. The process repeats until the user "evolves" a satisfactory result. The advantage of interactive evolution over inverse design, is that a person may apply an objective function that is understood only implicitly (in the form of subjective selection), and need not make explicit the objective function. As a result, the user's fitness function can incorporate poorly defined characteristics such as "interesting", "aesthetically beautiful," "good likeness," or "life like" (Baker and Seltzer, 1994). Such properties would be difficult to formalize in a mathematical sense. In addition, a person's criteria may change during the interactive session; this is often an integral part of iterative refinement. By alleviating the problem of requiring users to specify an objective function in advance, this collaborative approach supports a browsing capability that enables users to refine their queries through visual means, and supports the exploratory component of design.

Interactive evolution was first introduced by Dawkins (1987) who describes a system for evolving images of creatures called "biomorphs". Interactive evolution has subsequently been used in a variety of applications including generating facial images (Caldwell and Johnston, 1991; Baker and Seltzer, 1994), creating insect-like images (Smith, 1991), motion synthesis (Ventrella, 1995), volume rendering (He et al., 1996), and other computer graphics tasks (Kochhar, 1990; Sims, 1991; Todd and Latham, 1992). In most cases, interactive evolution is more powerful than traditional computer-aided graphics tools in that it enables people with no previous artistic training to generate interesting images and animations. It enables users to explore large design spaces, relying on only gross interaction techniques in communicating with the computer, typically pointing and clicking with a mouse to indicated images of interest. More recent work (Baker, 1997) has focused on implementing an

interactive evolution system for searching a database of existing facial images for a particular person (i.e., as a computer-assisted mug book). Thus, the interactive evolution approach seems promising for both instantiative or searching tasks (i.e., finding a particular image) for narrow areas within the design space.

Design Galleries and interactive evolution are appropriate to use for design tasks involving unquantifiable output characteristics. In both cases, the user is the source of output quality judgments. One distinction between the two approaches comes in their utility as browsing systems. A Design Gallery application uses dispersion to provide the user with a set of graphics that are representative of the design space, thus providing an overview of the entire space. Interactive evolution systems, on the other hand, typically seed their initial population with a random sample of graphics, with future populations being generated using genetic algorithms as guided by user selection. Depending on the application, the user can sometimes provide a sample image as a starting point. This is not, however, typically the case. The initial sample, whether randomly generated or user-supplied, is not guaranteed to cover the entire design space, particularly due to the limited size of the sample and the vastness of the design space. Thus, although interactive evolution systems have utility in exploring a particular area of the design space in detail, they are less appropriate for general browsing.

A second drawback to interactive evolution is the impact of the computational cost on system performance. As previously noted in Section 4.1, some computer graphics algorithms are too computationally costly to be incorporated into interactive evolution systems. These systems require computation to generate candidate graphics for their user at each iteration in generating a single generation. For many computer graphics algorithms, the high computational costs preclude graphics (images or animations) from being computed in real time. For such algorithms, interactive evolution becomes unusable. Although a user can make aesthetic judgments in real time, the system would be unable to generate new sets of candidate solutions in a time period that would be acceptable to users; the cycle of iterative refinement is broken by delays in generating graphics (or animations) for a user to evaluate. In contrast, the Design Gallery framework allows users to explore and interact with computationally expensive algorithms because the computation is done ahead of time. A Design Gallery application runs its dispersion heuristic as a pre-processing phase; users are not subjected to the time delays associated with computationally expensive algorithms. In addition, the use of a mapping function and output vector (rather than a full algorithm and output graphic) enable a Design Gallery interface to evaluate and explore many more graphics than other approaches.

4.5.2 Information Visualization and Navigation

Existing approaches to parameter setting do not typically include an arrangement component. Research on information visualization and visual information seeking, however, addresses many of the same issues, emphasizing the processes of navigating through large collections of information and interacting with users through visual means. Users need a means for exploring large information spaces — a design space is just a specialized information space in which each document is a single graphic. Browsing systems aid users in accessing these large data sets. The amount of information to be presented to users is continually growing and becoming more complicated. Typical computer screen sizes, however, have remained relatively small. Therefore, systems need to employ innovative methods to

enable users to better understand these large spaces. In this section, we survey several systems that provide innovative approaches to information visualization and navigation.

Ahlberg and Shneiderman (1994) defined design principles for visual information seeking, relying on people's capacity for visual information processing. They incorporated notions of proximity, color and size, along with animated presentations, and user-controlled selections in order to support users in exploring large information spaces rapidly and reliably. Their Starfield approach relied on scatter plots of points. The points represented various objects in a database, such as people, videos, papers, songs or photos. Meaningful two-dimensional displays were produced by having the user select two ordinal attributes of the items to be presented; additional features supported selection and zooming which could be used in refining a query. Text documents could be arranged by two of the following characteristics: author, year of publication, or word count. A database of people might include attributes such as age, number of siblings, number of years of education, salary, or other demographic variables. It is not clear that the two components of the Starfield approach are applicable to the domain of graphics-based information. First, using points as icons for a graphic denies the user easy access to important visual information. Second, "natural axes" may not be as easily identified for graphics-based documents.

A second application of relevance is that of navigating a database of color images (Rubner, Guibas, and Tomasi, 1997; Rubner, Tomasi, and Guibas, 1997). This work takes a similar approach to that of Design Galleries, although the work was done independently and in parallel with our project. It utilizes a notion of color signature (comparable to our output vectors) in order to evaluate the similarities of color images, and uses a multi-dimensional scaling technique to embed the images in a two or three-dimensional space. Rather than providing a listing of images resulting from a user query, it presents the user with a visual representation of the results in two-dimensions. In contrast to a one-dimensional list that reveals only the distance between the query and each element of the results, the two-dimension visualization used in this approach (and in Design Galleries) can be used to convey the distance between all the images in the set. In addition, the use of thumbnails provides important visual information to users.

The notion of multiscale viewing (Furnas and Bederson, 1995) in which objects and structures embedding them can be displayed at different scales is helpful for users in navigating large information spaces. Traditional flat views, such as a single window on a computer screen, provide users with access to a single, small, local piece of the structure at any given time. Although users can control the locality of this view, they have no access to the larger context or "big picture". The use of zooming enables user to change the magnification level of a particular view, but it still limits users to either local or global access, but not both simultaneously.

The Spatial Data Management System (Donelson, 1978) was one of the first systems developed to offer users both global and local views. It provided two windows, one as a panoramic view, and the other a close-up view of an information landscape. Users navigated by either panning in the local window, or clicking in the global level to move directly to an entirely new area.

Furnas (1981; 1986) suggested an approach to provide users with easy access to peripheral information. His *fisheye* lenses present a distorted or warped view of a space; things near the center of the lense are highly magnified, but the whole structure is shown, with decreasing magnification from the center of vision. The result is that information of current interest has the greatest detail, while surround information has a less-detailed view.

Peripheral information provides important context which aids users in orienting themselves as they navigate a space.

Pad (Perlin and Fox, 1993) uses a spatial metaphor for computer interface design. It is an infinite two-dimensional informational plane that can be shared across users. Pad provides views, called *portals*, that aid the user in the navigation of the infinite space. Portals may have varying magnification levels, providing panoramic overviews, or small-localized access. Interestingly, portals may be recursively applied to themselves, increasing the magnification and access of particular regions at the user's discretion. The apparent size of document determines the amount of detail provided to the user. Pad also introduces the notion of *semantic zooming*: an object can change appearance as the amount of space it is allotted changes. In traditional geometric zooming, objects change only their size, and not their shape, as the magnification level changes.

4.6 Summary

Design Gallery interfaces are a useful tool for many applications in computer graphics that require tuning parameters to achieve desired effects. Their basic strategy is to distill from the set of all possible graphics a subset with optimal coverage. The gallery is automatically constructed through a dispersion and arrangement phase, which is typically computationally intensive. The intent is that this process occurs off-line, for example, during an overnight run. After the gallery is constructed, the user is able to effectively browse through the space of output graphics. Previous approaches such as inverse design and interactive evolution are infeasible due to the problems of unquantifiable output characteristics and high computational costs associated with many computer graphics algorithms.

In examining the challenges associated with parameter specification approaches, it is important to note where the burden falls. Unquantifiable output characteristics and high computational costs are problematic issues for users. The technical challenges associated with Design Galleries — dispersion and arrangement — shift the burden onto the computer and the software designer. As discussed in Sections 4.3–4.4, we have met these challenges. The results of applying the same dispersion and arrangement techniques to different applications are given in Section B.

The Design Gallery methodology utilizes the collaborative framework in determining the division of labor between people (Design Gallery designer and users) and the computer system. In characterizing the roles of the user and the computer in an optimization-based framework, the Design Gallery approach follows the work of the previous chapters. The creator of a Design Gallery interface (who is in some sense part of the application) works at a local level, choosing the input and output vectors, along with the distance metric. The computer also works locally, performing the dispersion, the mapping of input vector to output vectors and output graphics, and the arrangement of final graphics in the gallery. The user operates at a global level and is responsible for selecting graphics of interest, and arranging them in the gallery in a meaningful fashion. The collaborative nature of the Design Gallery interfaces solves the problem of parameter specification, and supports people in exploring and understanding large design spaces.

Chapter 5

Floor Plan Segmentation

The problem of floor plan segmentation — identifying partially and fully bounded regions in a bitmap image — is a representational graphic design problem that emphasizes the importance of converting information in hard-copy form to its electronic equivalent. The bitmap image is typically a scanned floor plan; the user's goal is to generate an electronic representation of the building geography depicted in the floor plan. In particular, we would like to support the delineation of regions demarcated by *subjective contours*, making the process as easy and user-friendly as possible.

In this chapter, we describe work on an interactive system for floor plan segmentation. As our system extracts partially or fully bounded region definitions from a scanned bitmap image with minimal input from a user, it is better suited for generating representational graphics of this nature than other semi-automated techniques. Our method may be applicable in other domains, such as forms processing, cartoon coloring, or web image map generation.

5.1 Introduction

In general, geographic information for buildings is available only from hard-copy floor plans. Although geographic data for buildings is sometimes generated with architectural CAD systems, large quantities of paper-based information pre-date CAD tools. The issue of transforming hard-copy information into machine-readable form falls within the realm of document processing, a field that has traditionally focused on segmenting scanned documents into regions of text and images, and then on interpreting these regions. Our focus is on the latter part of this process: interpreting a scanned bitmap that depicts a building floor plan. We are interested in the topology and geometry of the different regions (e.g., offices, lobbies, or closets) of the building. We wish to extract such data from a scanned floor plan, by annotating the floor plan to delineate the relevant regions. For this reason, the graphic design task at hand is not an end in itself; it is an intermediary step of a larger process.

As with the problems addressed in the previous two chapters, approaches to floor plan segmentation range from manual to automatic. Manual methods, such as tracing (using a mouse or digitizing tablet), are tedious for people and are inherently inaccurate. Automated methods, though less laborious, do not always yield correct and accurate methods. The task of region delineation, like many graphic design tasks, can depend in the end on arbitrary semantic information about the material depicted in the bitmap to which no purely syntactic

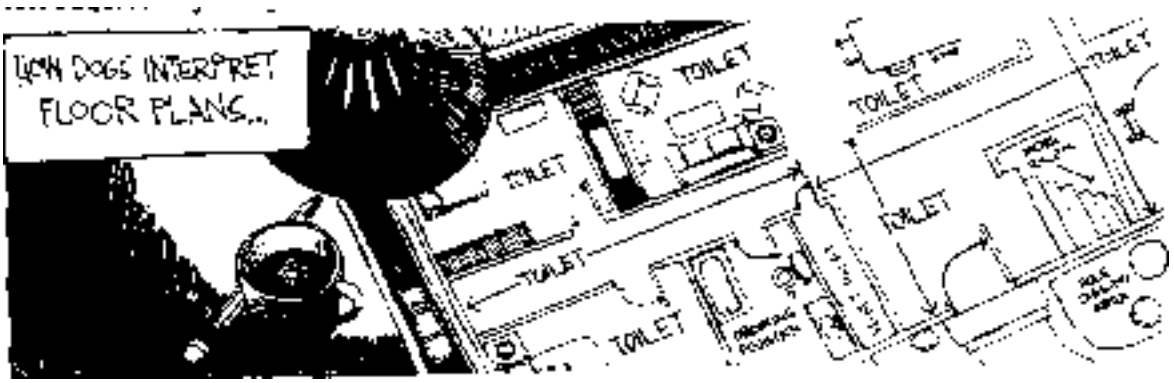


Figure 5.1: Arbitrary amounts of semantic information may be required to interpret a floor plan.

method can be sensitive. As illustrated by the *Non-Sequitur* cartoon (Wiley, 1994) in Figure 5.1, region boundaries (and their utility) are in the eye of the beholder. As another example, the distinction between a foyer to a room and a bay in it may be unmarked in a floor plan. If the former is considered a separate region and the latter part of the same region, only someone familiar with this semantic distinction would be able to delineate the regions correctly. Because no fully automatic method is to be expected, we believe that it is crucial to think of the task of region delineation as being solvable only semi-automatically.

We wish to explore collaborative methods that utilize a similar division of labor between user and computer but relax these assumptions. Our approach integrates a novel method for providing a syntactic model of subjective regions into a semi-automatic system for delineating fully and partially bounded regions in a scanned floor plan image. We encapsulate the notion of subjective region boundaries into a single function, and view the region-identification problem as optimizing this function. In the following section we describe our approach, focusing on the region delineation technique. Section 5.3 discusses the computer-user interface design, followed by an example interaction with the system in Section 5.4. In Section 5.5 we compare our novel region segmentation technique to previous approaches, and the collaborative system into which it is integrated to existing systems that might be used for the task of floor plan segmentation.

5.2 Our Approach

Our approach makes use of a proximity metric for delineating partially or fully bounded regions of a scanned bitmap that depicts a building floor plan. A proximity field is defined over the bitmap, which is used both to identify the centers of subjective regions in the image and assign pixels to regions by proximity.

Our approach has two main advantages over existing techniques. First, the region boundaries generated by the method tend to match well the subjective boundaries of regions in the image. As discussed in Section 5.5, other methods (such as filling) are not suitable for identifying partially bounded regions. Second, our technique is incorporated into a semi-automated interactive system for region identification in floor plans. Simple human

interventions requiring only gross information can be used to correct the results generated by the proximity-field method. This is in keeping with the collaborative framework proposed in this thesis. The division of labor is such that the computer is working at the local level, assigning pixels to regions, while the user works at a global level, guiding the system and correcting mistakes with gross gestures.

5.2.1 Proximity Field

A novel contribution of our approach is the method by which the subjective boundaries of regions are defined. As discussed above, the standard area-filling method has the problem that subjective boundaries that are not marked with explicit lines in the image are overrun so that neighboring regions are invaded. Our approach attempts to better characterize the notion of a subjective region by encapsulating the notion "subjective region boundary" in a single function and by viewing the region-identification problem as the problem of optimizing this function. The approach can be motivated by reconstructing the problem with the area-filling method. Suppose we are given a drawing of the two-room building given in Figure 5.2. Note the door between the rooms. Because of this door, a filling algorithm started from any point would find a single region (as in Figure 5.2b). Intuitively, the reason that a given pixel, say the one labeled r in Figure 5.2f, is taken to be associated with the right room rather than the left (where the filling was started) is that it is closer to the center of the right room than the left room. In order to use this intuition to actually delineate regions, we must find a way of characterizing these two notions of 'closer' and 'center'.

We do so with a proximity field. Imagine a surface defined so that the height of the surface at each black pixel in the image is zero and at each interior pixel the surface is as many units below zero as the pixel's distance to the nearest black pixel, so that the surface forms a series of valleys with the black pixels as ridges separating them. The surface just defined is the proximity field; a topographic map of a surface is given in Figure 5.2c and a cross-section is shown in Figure 5.2d. Local minima in this field provide a rough characterization of the notion center of a region.

We want to assign each pixel in the image to one of the field minima, in particular, the closest one. The appropriate notion of closeness is not mere geometric distance (as defined by, say, a "Manhattan distance" metric). Instead, the surface itself provides a definition of closeness. Objects on such a surface tend to move downhill so as to locally minimize their potential energy. The minimum reached from a given pixel by such a local minimization process is an appropriate notion of "closest center". We can imagine a ball placed at the given pixel and released; the local minimum that it settles in is the center of the room the pixel belongs to. Since, as depicted in Figure 5.3g, a ball at point p would roll to the upper minimum, whereas one at point r would roll to the lower minimum, the two points are taken to lie within the upper and lower region respectively.

This rolling ball analogy fails, unfortunately, when the pixel in question is on a plateau of the surface between two minima, say at point q in Figure 5.3f, in which case both minima are reachable by descent from the given pixel. Conceptually, this aberration can be eliminated by using a smoothed version of the surface as depicted in Figure 5.3e. In practice, the allocation of pixels to regions that would be engendered by using the smoothed surface can be calculated directly by the technique described in Section 5.2.2, without actually calculating the smoothed field.

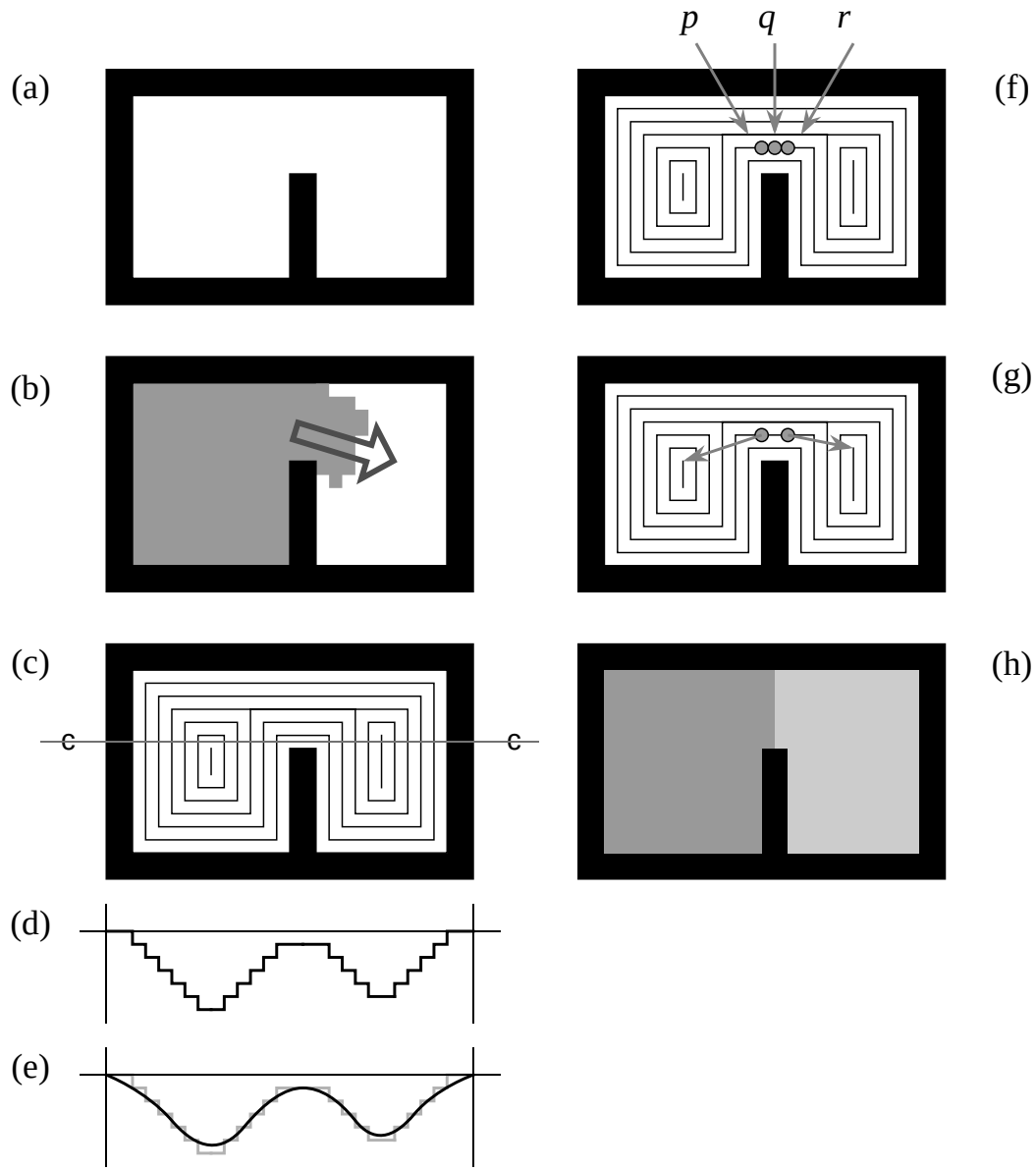


Figure 5.2: (a) A simple two-room drawing. (b) A filling algorithm started in the left room traverses the subjective boundary of the doorway. (c) A "topographic map" of the proximity field for the floor plan. (d-e) Unsmoothed and smoothed cross-sections of the proximity field along a horizontal line through the doorway. (f) Points to the left of the doorway (as point p) are nearest to the local minimum in the left region, whereas points to the right (as r) are closest to the local minimum in the right region. Points right in the middle (as q) may be considered part of either room; the algorithm presented would associate such a point with the deeper minimum, hence the larger left room. (g) The notion of closeness can be motivated by imagining a ball located on the surface, rolling downhill to the nearest local minimum. (h) Regions defined by the proximity-field technique shown as a two-coloring of the floor plan.

This discussion provides evidence that an approach based on energy minimization in the proximity field might be appropriate for characterizing subjective boundaries better than the simple filling technique. Our method is based on just this metaphor of energy minimization. We describe the method in more detail in the next section.

Another feature of the proximity field technique is that alternative definitions of the proximity field might be used to characterize regions of a quite different sort from boundaries perceived as physically contained regions. The simple field used here models regions unconstrained in all directions except by overt indicators of subjective contours. The sensing region of a motion sensor, by contrast, is constrained to subtend a certain angle outward from the site of the sensor. The regions defined by such a sensor can be defined by an alternative proximity field that is more constrained than the one used here but that still respects subjective regions. The generality of the proximity field approach to region delineation thus makes possible the modeling of many different kinds of regions. Similarly, an alternative proximity field definition might be used to find the subjective segmentation of a document image.

5.2.2 Region Definitions

A region, under the model we are describing, is characterized as the set of pixels for which a given minimum in the proximity field is closest. Thus, specifying a region follows from specifying the corresponding field minimum. This is easily done by local search. Given a set of pixels all of equal value, which we will call the working set, a new set is generated in the following manner. The neighboring pixels with the lowest field value are found. If the pixels in this neighboring set are higher than those in the current set, the current set constitutes (part of) a local minimum of the surface, and the search is done. If the neighboring set pixels are the same height, the working set is augmented with the pixels in the neighboring set, and the process is iterated. If the neighboring set pixels are lower, the neighboring set becomes the current set and again the process is iterated. Eventually, this iterative search terminates with a set of equal-value pixels that comprise a minimum. A graphical depiction of the iterative process is presented in Figure 5.3.

Under certain conditions, for instance, when started at point q in Figure 5.3f, the working set may become temporarily noncontiguous. In the area within the door, for instance, there is a plateau in the proximity field. The first few iterations of the algorithm above expand the working set to encompass larger areas of the plateau. Eventually, if the starting point is nearer one room or the other, the edge of the plateau nearer that side will be found, and the working set will move in that direction, eventually settling in the corresponding minimum. However, if the starting point is in the exact center of the plateau, both edges will be found on the same iteration, and the working set will comprise pixels from both sides of the plateau. The search will continue in this way until one of the discontinuous parts of the working set hits a minimum. If the other part is not yet at a minimum, the search will continue, using only the latter portion of the working set, eventually finding the lower of the two minima. In essence, the method as described places points midway between two rooms as part of the larger of the two rooms. (If both minima are reached at the same time, that is, both rooms are the same size, one or the other can be chosen arbitrarily.)

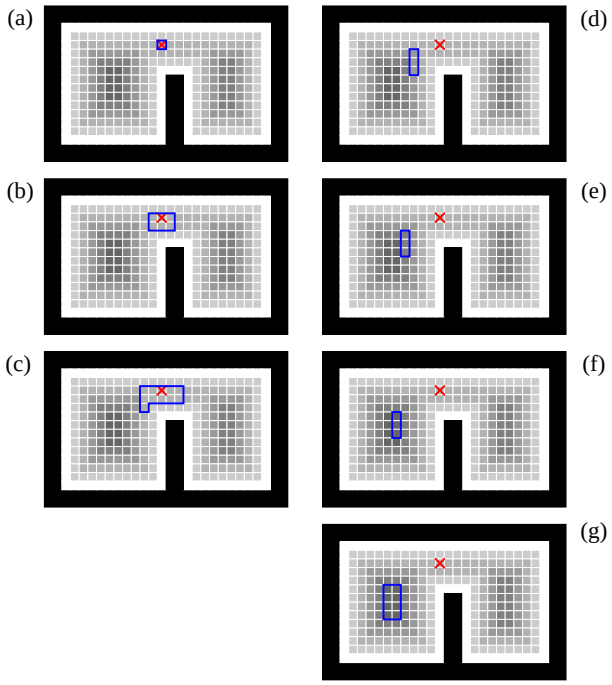


Figure 5.3: Stages in the computation of the nearest local minimum to point p of Figure 5.2f. The proximity field is given as a false grayscale coloring of the interior pixels, with darker colors being lower. The starting pixel is marked with an $\times$ in all drawings, and the pixels of the working set is bounded with a polygon. (a) The working set is initialized to the starting pixel. (b-c) The lowest-valued neighboring pixels are of the same value, so are added to the working set. (d-f) The lowest-valued neighboring pixels are of lower value, so they become the new working set. (f) Pixels within the local minimum have been found. (g) The working set grows to encompass the full local minimum, and the algorithm halts.

5.2.3 Weaknesses

The basic proximity field technique works quite well for finding subjective contours of regions in bitmaps. Section 5.4 and Appendix C provide examples of the method as applied to actual scanned floor plans. However, as previously argued, the articulation of any such method with some human intervention ability is crucial. The essential idea behind leveraging of human intervention is simply this: repairing of incorrect or inaccurate results of the automatic method should require only gross human interaction, rather than finely detailed work. This is the heart of the collaborative approach proposed in this thesis. Users should work at the global level, leaving the system responsible for the local level. The proximity field technique compares well with other techniques, such as area filling, not only in that it performs better ab initio at finding subjective contours, but also because it lends itself well to this leveraging the addition of simple human intervention.

The remaining frailties of the proximity field technique can be classified as follows:

- *Missing subjective contours:* Indicators of a subjective contour may be wholly or partly absent from the input image. For instance, what appears to be a single large room

in the floor plan may be thought of by the occupants of the room as two separate regions. Since no syntactic reflex of the distinction between the two rooms is found in the image, no method based purely on bitmap processing can be expected to observe this distinction.

- *Multiple minima:* A subjective region may encompass several minima, that is, it is the union of the regions defined by the multiple minima.

Both of these problems are easily handled by only simple human interventions, given an appropriate computer-user interface. Indicators of missing subjective contours can be manually added with simple line-drawing tools. Because the method does not require closure of the subjective contours, simple hints as to the missing subjective contour are all that is typically required to repair the region delineation. For example, in Figure 5.6 in the next section, a short line segment is all that is required to coerce the proximity field method to find the correct subjective contours. Multiple minima and their corresponding regions may be joined into a single region. Details are provided in the following section.

5.3 System Design

Using our collaborative interface approach, we frame the problem of floor plan region segmentation as an optimization problem. Our goal is to have the user work at the global level, and the system at the local level. The proximity field method described in Section 5.2 is well-suited for this approach. The user brings semantic information to the task — knowledge of the area represented by the floor plan. The system employs a local search strategy as it attempts to segment the image into regions. The search is guided by the user at a global level using two mechanisms as described below.

We have implemented the proximity field technique for region delineation in a prototype system using C and Motif. The interface for the system includes a viewing area for the bitmap image and a control panel for various parameters. A screen capture of the interface is shown in Figure 5.4.

Input

Our system takes as input a scanned bitmap image, such as the one in Figure 5.5. By starting from a scanned bitmap image, the system relieves the user from having to redraw the layout geometry from scratch. Although some CAD packages use the scanned blueprint as an underlay, the user is still required to trace the outline of the regions manually. Bitmaps for our system should be pre-processed to remove text. Such bitmaps will typically be generated by scanning the hard-copy of a floor plan. The resulting bitmap may contain noise or missing boundaries — our system is such that it can handle missing or corrupted data. Other floor-plan-like drawings may be generated by simple drawing packages. An example of this type is given in Figure C.1.

System Implementation

The system computes the proximity field over the bitmap using the method described in Section 5.2. The results can be be displayed to the user as a grayscale topography of the surface, as shown in Figure 5.6. The value at each white pixel is shown through false-coloring, with darker tones connoting lower values. This view can help shape the user's

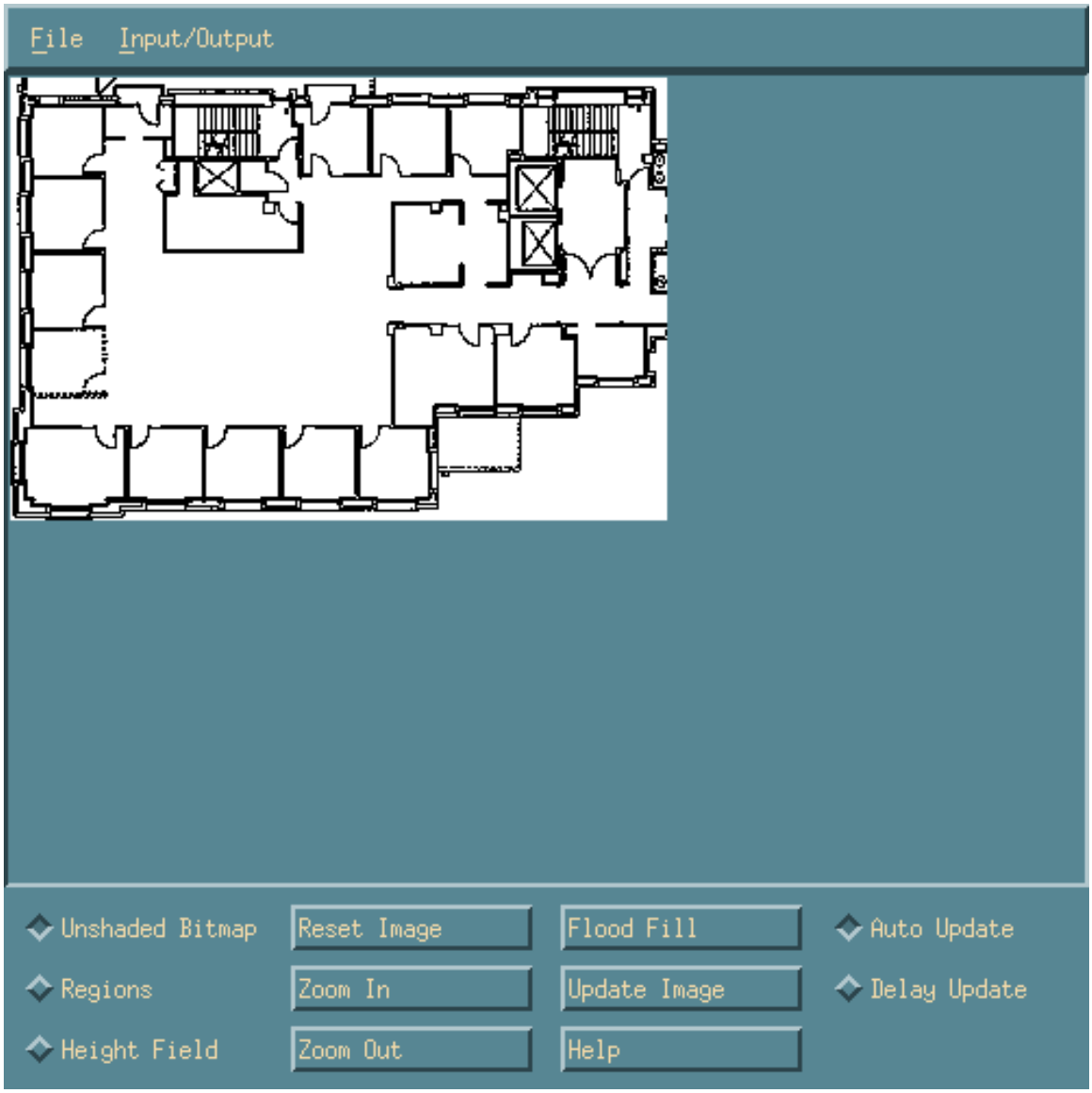


Figure 5.4: Computer-user interface for the system.

intuition about how the system is detecting and defining regions. The system then uses the proximity field to compute the corresponding regions, and displays these to the user as a false coloring of the regions, as shown in Figure 5.7. The colors are assigned to the regions randomly, and so adjacent regions (potentially with no physical boundary between them) may be assigned the same color. The user can change the color of a region using the mouse — the system will cycle through a pre-set listing of colors. The user may also specify different stipple patterns to be used in coloring the regions.

User Interaction

Our system reduces the amount of user input, and simplifies the *type* of interaction as well. The system provides for human intervention in the form of user-added subjective contour hints and region grouping. Subjective contour hints act as walls, which enable a user to subdivide a single region. The user can use the mouse to add sketch lines to bitmap image. These additional "walls" need not be accurate; they can be slightly skewed, and need not abut nearby physical walls. The user is not required to work down at the pixel level, but rather need only use gross mouse gestures to achieve the desired effect. The user can control whether or not these false-walls are visible in the display. The system also permits users to delete any of the false-walls they have previously introduced.

In other instances, a user may wish to join two or more regions into a single logical region, for example joining door swings to their corresponding rooms. This is accomplished by selecting the regions to join with the mouse. Depressing the left mouse button starts a chain of regions to be joined; clicking the right mouse button adds additional regions to that set. The user can delete these joins at a later point if desired. No pixel level postediting of the regions is supported; the need for post-editing at this level of detail is rare and would be better viewed as an indicator of a flaw in the method. Both joins and subjective-contour hints can be saved to an output file to be used in later editing sessions.

All human intervention is achieved through standard graphical drawing and selection techniques that can be used at several magnifications of the image. Users can zoom in and out on the bitmap images using the buttons on the control panel. In addition, the system provides a choice of three views of the bitmap image: the black and white original, the grayscale proximity field, and the false colored regions. As the user adds (or deletes) subjective hints (in the form of false walls) to the image, the system will need to update the proximity field, and in turn the region definitions. Users also have the option to turn off the automatic update mechanism, enabling them to make several changes at once, before having the system run an update. In the case of joins, although the region definitions are modified, no changes are made to the proximity field; no additional calculations are needed.

5.4 Example Interaction

In order to provide a feel for the quality of region delineation achievable using the proximity field on actual scanned bitmaps, we apply it to a sample scanned floor plan, a map of the Digital Equipment Corporation Western Research Laboratory. The original bitmap is shown in Figure 5.5. Note that the boundaries of some of the rooms are discontinuous as a result of scanning errors; the lowest of the four rooms on the left wall is an example. Other rooms, for instance the room in the lower right corner, have open doors.

Based on this initial bitmap image and using the technique described in Section 5.2.1,

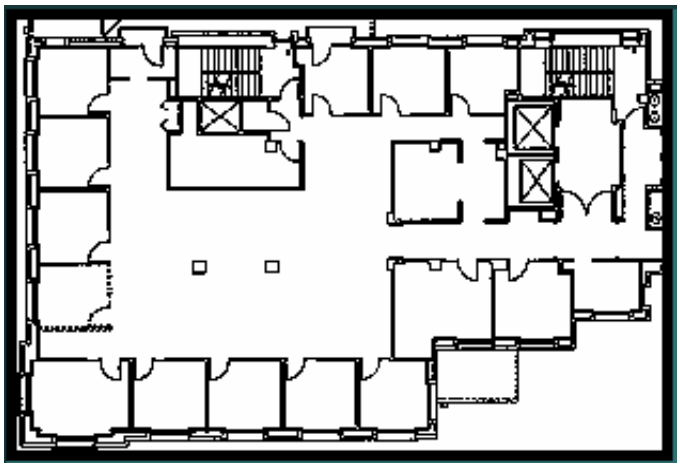


Figure 5.5: The original scanned bitmap.

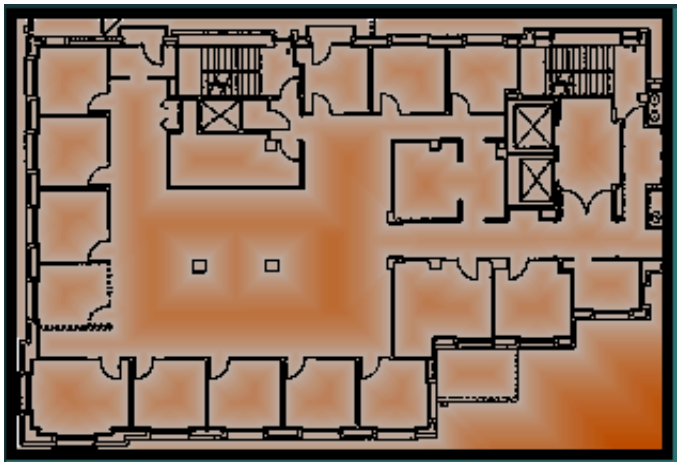


Figure 5.6: The false-colored proximity field generated for the bitmap.

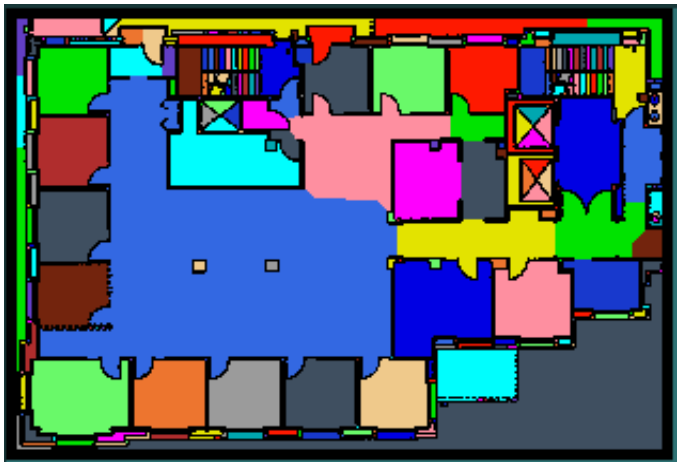


Figure 5.7: The regions delineated according to the proximity map in Figure 5.6.

the computer calculates the proximity field which is depicted using a grayscale coloring in Figure 5.6. The darker colors correspond to lower values in the field. When regions are defined on the basis of this proximity field, the results, shown in Figure 5.7, are a good first approximation to the desired region boundaries. In particular, the rooms mentioned above with discontinuous borders are correctly delineated. Nonetheless, several problems remain. These are easily remedied through simple user intervention.

The user begins by introducing a single subjective contour "hint" at the upper right of the main lobby using gross mouse gestures. A second one is introduced to mark off the small corridor in the lower left corner. As illustrated in Figure 5.8, neither false wall is exact; they do not abut any of the nearby walls. Nevertheless, our technique is robust enough to calculate acceptable region definitions.

Next the join operation is used to merge several regions into a single region. The user first clicks the mouse above the upper-right lobby area, and then in each of the two door swings adjacent to it. Although this has no direct effect on the proximity field, the three regions are joined creating a single region as desired. The final region delineation (determined to be acceptable by the user) is shown in Figure 5.10.

5.5 Previous Work

In this section, we survey previous work on topics related to floor plan segmentation. We begin with an overview of several existing software systems that might be used for the task of region segmentation. Next we examine work in the areas of vision and image processing, and document analysis. We conclude with a discussion of techniques used in drawing tools and a comparison of the various techniques.

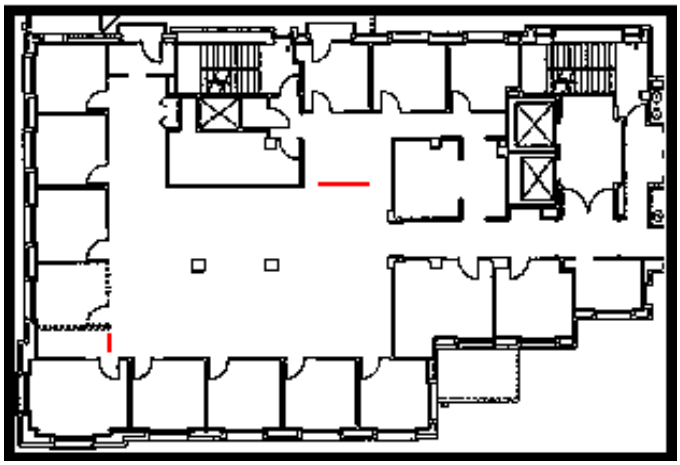


Figure 5.8: The user adds two false walls.

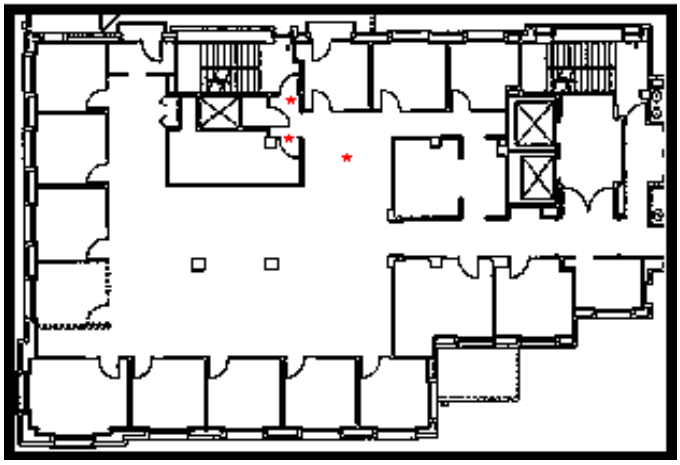


Figure 5.9: The user joins the door swings to the upper hall region.

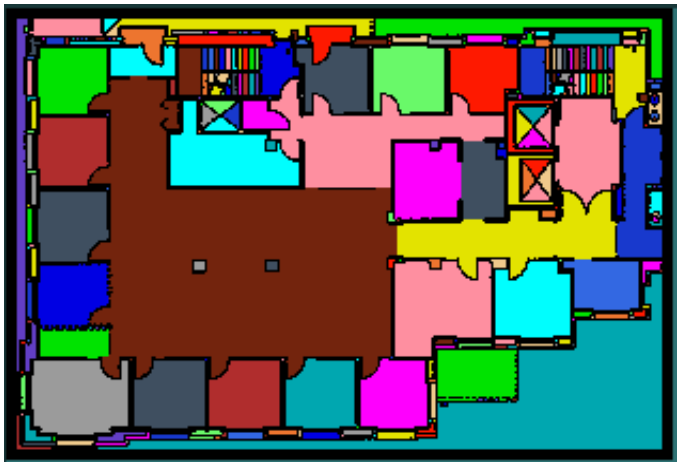


Figure 5.10: The final region definitions.

5.5.1 Systems

MARIS

The MARIS (Map Recognition Input System) system (Suzuki and Yamada, 1990) is designed to digitize large-reduced-scale maps into a layered representation. Large-reduced-scale maps contain more detailed geographic information than small-reduced-scale maps. Most techniques to deal with the former are pixel-based, and as such require increased computation time. Because automatic techniques are not completely accurate, semi-automatic methods are required to make corrections and refine derived representations. MARIS combines several techniques: vectorization, border tracing, a vector-based map recognition method and an interactive input and correction method which result in an efficient system for digitizing maps. The authors believe that their methodology can easily be applied to other types of drawings, such as architectural drawings. The only required change would be to modify the recognition method used by the system.

ROCKI

The ROCKI project (Nardelli, Fossa, and Proietti, 1993) takes a different approach. Using a three-step knowledge-based approach, the system starts with primitive interpretation, through which it identifies basic graphical objects. These objects are then interpreted and grouped to obtain structured objects. This leads to the final step in which the document is interpreted, and semantics for the document are defined. The authors exploit contextual information in order to do their analysis. The ROCKI framework is intended to be a toolkit which can be used for interpreting maps and office documents. One implementation, as described in (Nardelli and Proietti, 1993a; Nardelli and Proietti, 1993b), has a similar goal as our work: to produce a semantic representation of "a low-level image representing a floor plan map". The input to this system, however, is not a scanned bitmap image but an AutoCAD file containing a vectorized representation of all objects within the image. In addition, AutoCAD presents a structured layer representation of the data; thus the input to this system is considerably more detailed and structured than a scanned bitmap image.

Moreover, this system makes several other assumptions:

- objects (walls, doors, windows, pillars) are represented by rectilinear segments
- the rooms of a building are represented by a set of closed polylines
- polylines are made of vertical and horizontal segments

These assumptions simplify the problem a great deal. Specifically, they disallow regions with subjective contours. It would seem that a corridor, which is not usually defined by a set of closed polylines, would be unidentifiable in this system. Yet the authors claim that passageways (including corridors, entrances and landings) are in fact identifiable. Although the goal of this work is the same as ours, namely to determine a meaningful representation of a floor plan, their approach to the problem differs drastically from our own.[1]

Mapedit

Mapedit is a specialized WYSIWYG editor for creating World Wide Web image maps, available for Microsoft Windows and the X Window System. Image maps turn a GIF image into a clickable map, by designating regions of interest within the image, and specifying a URL link for each region. Creating image maps by hand is often difficult and tedious. Mapedit employs several of the techniques found in a variety of drawing tools. It allows users to load an image into an editing window. Users then draw regions on top of the background image, defining areas of interest. Users may draw polygons by tracing various regions, or by using standard templates (rectangles, circles, etc) for delineating regions. Once an acceptable region has been defined, the user inputs the corresponding URL, comments, and other Web properties. Mapedit is better suited than generic drawing packages in helping a user generate image maps in that it provides the original image as a backdrop to guide the user in defining regions. Given that floor plans contain far more regions of interest than a typical image map found on the Web, Mapedit still places too much burden on the user for our domain.

Deformable Polygons

Our first attempt at designing an interactive system for floor plan segmentation was one in which candidate regions were identified semi-automatically through the use of deformable polygons. Deformable polygons are similar in nature to snakes (Kass, Witkin, and Terzopoulos, 1988) and deformable templates (Yuille, Hallinan, and Cohen, 1992), but are specialized to the particularities of the task at hand. The system supported the user in the task of interactively specifying important areas of the floor, such as rooms, lobbies, closets, etc. The user specified a location at which to place a deformable template; the template then deformed automatically to conform to the contours in the scanned image; when the template achieved a static conformation, it was proposed by the system as a candidate region. The user repeated this process for each area of interest.

To determine how well the edges of the deformable polygon conform to the contours in an image, we defined a potential surface over the image. This is comparable to the

[1] To date, this is the only documented work specifically addressing floor plan recognition that we have found.

proximity field metric previously discussed in Section 5.2.1. The polygon deformed so as to minimize its potential "energy," which was combined terms for both boundary and shape potential. The initial polygon was a regular octagon, but dynamic vertex insertion enabled the polygon to better fit more complexly-shaped regions. In addition, the use of a dynamic height field caused the polygon deformation to proceed in two phases: expansion and contraction. Details on the deformable polygon method can be found in (Ryall et al., 1993).

The use of deformable polygons to delineate regions in floor plans was still too user intensive. It required a user to seed each region of interest with an initial polygon. There were also a variety of parameters the user could tweak in order to gain better performance from the system. Although the deformable polygons lessened the type of user input — the user need only click the mouse (roughly in the center) in a desired region, it seemed that more responsibility could be given to the computer. In fact, we discovered the system could derive regions directly from the height field, without the use of deformable polygons. The resulting method (which is described in Section 5.2) is a good illustrative example of the collaborative framework proposed in this thesis.

5.5.2 Vision and Image Processing

Computer vision and image processing use a variety of techniques for edge detection and object reconstruction in gray scale images. One area of research is most relevant to our work. Deformable templates (Yuille, Hallinan, and Cohen, 1992) have been used for automatic facial feature extraction, and snakes (Kass, Witkin, and Terzopoulos, 1988) have been used to interactively identify regions of interest in medical imaging data. The basic approach is to define parameterized templates which interact with an image in a dynamic manner. The computer minimizes an energy function, which contains terms for various aspects characterizing the desired region. The minimum of the function corresponds to the best fit of the template to the contours in the image. In the case of deformable templates, a priori knowledge about the shape of the regions guides the detection process. Snakes, on the other hand, make no assumptions about the regions being identified. They rely on the user to position the initial template, and include more terms in the energy function to prevent the structure from deforming too much. We explored the use of a comparable technique, deformable polygons, which is discussed below in Section 5.5.1

Another approach to image analysis is to use morphological operators, originating from the field of Mathematical Morphology. Morphological segmentation approaches are typically either edge-based or region-based. As we have seen, edge-based techniques are inappropriate for segmenting floor plans because subjective contours are not fully-bounded. Region-based approaches are more promising. In particular, the technique known as the *watershed transformation*, first introduced by Beucher and Lantuejoul (1979) and more recently improved by Vincent and Soile (1991), is similar to our method of region segmentation. This method borrows its terminology from topography. It makes use of a watershed line to separate two regions, known as catch basins, and relies on the assumption that the regions to be identified are locally homogeneous. The technique is typically applied to gray scale images that are viewed as topographic reliefs, with each pixel being assigned a value corresponding to its intensity.

There are two main difference between the watershed and our region segmentation technique. The first difference is that we calculate the field over the image based on a proximity

metric whereas the watershed method derives its values from the content of the image. The second difference is our incorporation of the segmentation technique into an interactive system. The classic watershed technique tends to oversegment images due to the numerous local minima present in the image. To correct this problem, it relies on other automated methods to weed out background and unimportant regions. In our experience oversegmentation has not been a problem. This is primarily due to the nature of the floor plan segmentation task. More importantly, we rely on a person's judgement to determine when a region has been divided into too many regions and provide them with simple mechanisms to remedy the problem.

5.5.3 Document Analysis

A great deal of work has been done on general document analysis (Casey and Nagy, 1991; Kasturi et al., 1990; O'Gorman and Kasturi, 1992; Wong, Casey, and Wahl, 1982), maps (Antoine, 1991; Boatto, Consorti, and Buono, 1992; Kasturi and Alemany, 1988; Nardelli and Proietti, 1993a; Suzuki and Yamada, 1990), and technical drawings (Clement, 1981; Joseph, 1989; Pao, Li, and Jayakumar, 1991). General document analysis techniques provide a framework for understanding a variety of complex document types. These systems will often rely on (as yet unimplemented) specialized modules to interpret images or line drawings. As a result, although our work could be incorporated into a larger document processing system, it does not appear that this body of research has direct application in our domain. Line drawings are typically classified into three categories: maps, technical drawings (engineering and mechanical drawings), and circuit/logic diagrams[2].

Maps appear to be very similar in nature to floor plans. Most research has been done on *cadastral* maps, that is, maps describing the geometry of land properties, including buildings, in a geographical context. The image is divided into polygonal regions; regions are fully bounded. In the case of floor plans, however, there are regions that are only defined in terms of the boundaries of other regions. A corridor, for example, is not fully bounded, but tends to be defined as the area outside of a series of offices. Again, although some of the techniques used to interpret cadastral maps may be relevant, they will be unable to support subjective contours, which is one of our goals.

Finally, technical drawings, such as engineering designs, are related to floor plans in that they use a set of standard symbols, and incorporate dimensioning text into the image. An important difference when compared to architectural drawings is that technical drawings are most often depicting 3D objects whereas floor plans are 2D in nature. In addition, in a floor plan it is usually permissible to approximate a curved line through a series of straight line segments; this is not the case in mechanical drawings, for example. Depending on the domain, standard symbols can be used to denote a variety of objects. In floor plans, for example, windows, pillars and doors will usually be marked by conventional symbols, although the conventions may change depending on when the floor plan was drawn, and in what country, for example. In addition to pre-defined symbols, shading or pattern information can also be used. In the case of cadastral maps, for example, cross-hatching is used to indicate buildings within the image (Boatto, Consorti, and Buono, 1992). It is important to recognize buildings prior to vectorization, as only the perimeter of a building need be vectorized. Many systems use a feature-based approach to recognizing symbols.

[2]We have not explored the latter category sufficiently to include it in the current discussion.

This method is attractive because it is robust enough to handle symbols that have been arbitrarily rotated or scaled.

The removal of dimensioning text in engineering drawings is a current research topic. Standard segmentation techniques separate an image into text and graphics. In the case of dimensioning notation, however, the symbols often overlap graphics. Dimensioning lines and associated text are known as dimension sets. Recent work (Chai and Dori, 1992; Dori, 1989; Fletcher and Kasturi, 1988; Joseph, 1991; Lai and Kasturi, 1993), has examined how to extract text strings and dimension sets from mixed text/graphics images. Our concern is not, however, how to extract the dimensioning text, but rather how to incorporate it into the representation of an image. As our system assumes a bitmap free of text, we have ignored dimensioning information to date, and have not addressed the question of how to "clean up" scanned images to make them acceptable input into our system. We believe that segmentation techniques and existing software tools will suffice. Dori (1991) makes a clear distinction between the geometry of the image, and the annotation within the image. He has examined the use of a flat matrix grammar (an extension of context-free grammars) in order to create a graphical knowledge base. An expert system then uses this knowledge base to automatically understand engineering drawings. Within our system, this technique may be useful in generating the semantic portion of the representation of the scanned image.

5.5.4 Drawing Tool Techniques

Semi-automatic techniques vary in the amount of input and guidance required by the user. In the case of freehand drawing and tracing, the user is doing most of the work. Under area filling and template filling, the user chooses the point from which to start the process, which is then performed automatically by the system. Several software tools exist today that can facilitate the delineation of areal regions on a floor plan. They all utilize one or more of the following four techniques:

1. *Freehand drawing:* The user redraws the floor plan from scratch with a mouse or digitizing tablet, delineating regions as part of the drawing process.

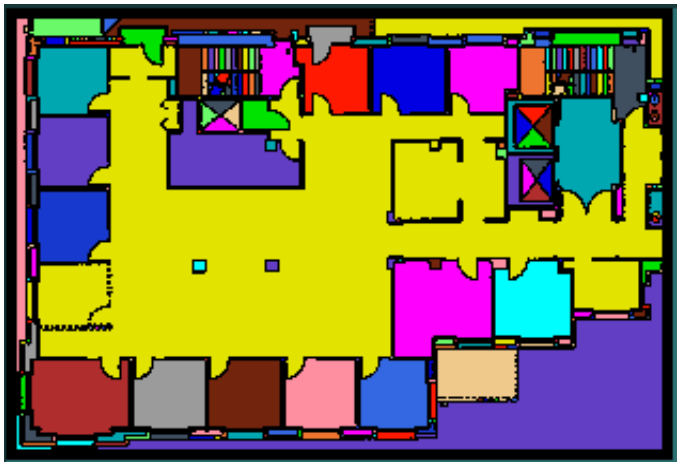
2. *Tracing:* The user traces out regions on the floor plan by hand, using a mouse or digitizing tablet.

3. *Area filling:* The user selects points on the floor plan image, and the computer fills in the fully enclosed areas surrounding the points. (The method is often called "flood filling".)

4. *Template fitting:* The user positions and sizes predefined templates (usually rectangles) to cover the desired regions approximately.

Template fitting, free hand drawing, and tracing are inherently manual techniques, require varying degrees of user input. Unlike the three manual methods, the area-filling and deformable templates approaches have the attractive property of being more automatic. Both, however, require user input to select the initial positions to start the algorithms from. In addition, as we have seen, the area-filling technique is brittle, as it relies on the unrealistic assumption that subjective boundaries in an image (the boundaries of image areas that are subjectively perceived as forming distinct regions) coincide with actual closed pixel contours in the image. Area-filling performs poorly on our sample floor plan, as shown in Figure 5.11(a).

(a)



(b)

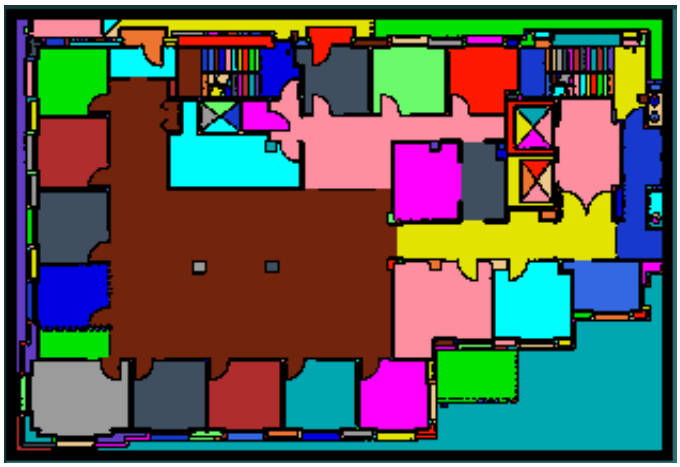


Figure 5.11: Regions delineated by two methods. (a) Regions generated by area-filling. (b) Regions generated by the proximity-field method, with minor post-editing. Note the introduction of the subjective-contour "hint" at the upper right of the main lobby and the joining of regions in the room above it.

It might be thought that the ability to add lines or other features to the image as a preprocessing step would vitiate the problems with area filling. It does not. In the case of area filling, quite fine-grained manual preprocessing of the image will tend to be required, because the preprocessing would need to be used not only to add missing large-scale subjective contours but also any missing bit of subjective contour down to the pixel level. The improved automatic performance of the proximity field technique means that only the grossest of missing subjective contours need be filled in. Because the proximity field technique is relatively insensitive to perfect abutting of lines and the like, the line drawing can be done quite rapidly, and on an as-needed basis. As the user of a system sees that a certain region is not being appropriately subdivided, the user merely needs to provide a hint to the system by quickly drawing a line segment where there is a missing contour.

Subjective regions encompassing multiple proximity field regions can similarly be handled without fine-grained editing of the image. A user can specify to the system that several regions should be combined to form a single subjective region merely by indicating the corresponding centers. Because the local search procedure described in Section 5.2.2 can find region centers starting from a large area around the minimum, the process of selecting the regions can be done manually with only gross human actions, such as mouse-clicking in the general vicinity.

Thus, the proximity field technique not only works better as an automated method for region delineation, but it fits well in a semi-automated system in that (i) the better performance means that less repair of results need be carried out manually, and (ii) the types of human intervention needed to correct the behavior of the technique require only large-scale gross actions rather than fine-grained editing. Figure 5.11 shows the results of delineating regions under two methods.

5.6 Summary

Region segmentation, like many design problems, cannot be solved by purely syntactic methods. Consequently, any method for identifying regions in a bitmap must allow for human intervention at some point in order to correct errors detectable only with semantic information. Techniques must also permit people to specify a region is partially bounded — the boundaries of the region might have been corrupted during the scanning process (a frequent occurrence), or the region might be bounded in part by subjective contours not depicted in the image. Existing approaches and systems are not optimal for the task of region segmentation as they rely on manual techniques (i.e., tracing) and often make unrealistic assumptions (i.e., regions are fully bounded) about the regions being defined or the input to the system.

Because it provides a better syntactic model of subjective regions, the novel proximity field method introduced in this chapter is better-suited for region delineation. The collaborative system into which it is integrated allows for simple interactive postprocessing. Subjective contour hints, in the form of false walls, enable users to subdivide regions. A joining operation permits users to combine multiple regions into a single area. Thus the resulting system is superior to both manual and automatic methods previously proposed.

The system described in this chapter exploits the collaborative framework proposed in this thesis. The computer works locally, calculating region definitions based on a local search technique. The process is guided by the user at a global level; simple human interventions requiring only gross information can be used to correct automatically generated region definitions. In addition, the user has the control to determine when the region definitions are correct (or good enough). Such a division of labor enables the system to work on a variety of floor plan "styles". See Appendix C for additional example floor plans.

Chapter 6

Conclusion

We have presented a new paradigm for computer-user interface design which helps determine the division of labor between the computer and its user. This paradigm is based on the notion of collaboration between the user and the computer; it exploits the strengths of each participant by giving the user the global portion of the problem and leaving the computer with the local portion. In contrast to the traditional master-slave paradigm in which the interface acts as a means for the user to control the computer, the collaborative paradigm leads to interfaces that act as media through which people and computers can work together to solve a particular problem. It encourages the development of semi-automatic systems through which users can explore a large number of candidate solutions, while evaluating and comparing various alternatives.

We have applied the collaborative approach to problems from the domain of graphic design, a domain in which aesthetic criteria and a user's personal preference play important roles in arriving at acceptable solutions. We have designed and implemented three novel systems for graphics design tasks. GLIDE is an interactive constraint-based editor for network diagram layout that supports users in interactively specifying the visual organization of a diagram. Design Galleries is a family of applications for parameter specification for a variety of computer graphic algorithms, including volume rendering, animation, and particle systems. Finally, our floor plan segmentation system enables users to easily and accurately identify regions of interest in a scanned floor plan image.

Currently, GLIDE can be used by the user to explore design alternatives. In some cases, however, the user may not know how to begin laying out a particular graph. Future work on the GLIDE system could include using several existing automatic layout routines to generate example layouts for a user to browse, providing them with a starting point for the layout they wish to generate. Output from another system could be used as input to GLIDE. Alternatively, the automatic layouts might be used only to help users generate ideas. An interesting question would be how to arrange these candidates to best aid users in their search.

Another area of further investigation would be to incorporate some notion of scaling and abstraction into the GLIDE system. At the present, the system provides users with a single view of the entire graph; if the graph is larger than the viewing area of the canvas, it may be accessed via scroll bars. For larger graphs, this may prevent users from getting an overview of the entire graph. By enabling users to view the diagram at varying levels of magnification, GLIDE could support users in understanding the layout as a whole. The use of abstraction techniques, such as grouping a set of nodes together and replacing them

with a single larger node (thus hiding the details of a particular component) might also be beneficial.

Design Gallery interfaces are a useful tool for many applications in computer graphics that require tuning parameters to achieve desired effects. The system interface enables the user to effectively browse through the space of output graphics. The arrangement component of the interface provides a two-dimensional space with minimal navigation and interaction techniques. The ability to annotate the design space would be a useful extension to the Design Gallery interface. It would enable users to explore spaces containing more representative points without getting lost by establishing landmarks for themselves (or others); it would also aid users in mapping the design space for further exploration.

We have shown the utility of the collaborative approach for computer-user interface design within the domain of graphic design. We believe that this this general approach to computer-user interface design will be applicable to problems outside the domain of graphic design. Once we replace the idea of computer as *servitor* with computer as *collaborator*, we will be able to design interfaces that better leverage human abilities through simple yet sophisticated interfaces. In a recent article, Wegner (1997) describes how by allowing interactivity, an algorithm could in effect be made more powerful than a Turing Machine.

> "Interaction is a more powerful paradigm than rule-based algorithms for computer problem solving, overturning the prevailing view that all computing is expressible as algorithms." (Wegner, 1997)

The collaborative framework presented in this thesis illustrates the importance of interaction in computing systems and of striking a balance between user intervention and computer control.

Appendix A

GLIDE: Additional Example Diagrams

In Chapter 3 we presented an example interaction with the GLIDE system; the user's goal was to create a particular layout already envisioned, namely the drawing in Figure 3.3 (Norton, Szymanski, and Decyk, 1995). In Section A.1 we examine the use of GLIDE to explore the development of a design alternative for the same data set. GLIDE can also be used to help users organized and understand previously unseen data sets, as illustrated by the example given in Section A.2.

A.1 Exploring Design Alternatives

As an example of the flexibility of the system, Figures A.1–A.8 present successive snapshots of quiescent states of the interface as a user develops an alternative layout starting from the same initial layout as the example used for the walk through in Chapter 3.

In order to generate a new layout, the user must identify for novel visual organizational patterns. In the layout in Figure A.1, there are two tree-like sets of nodes with roots at the `Particle` and `ParticleDistribution` nodes. The user adds a one *T-Shape* VOF as shown in Figure A.2, and a second instance as shown in Figure A.3 Note that the `Electron` node

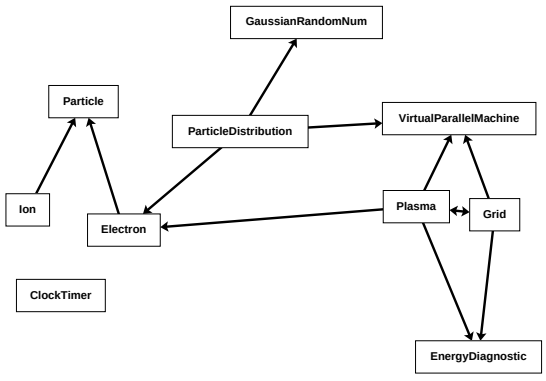


Figure A.1: Starting point for exploring a design alternative — same initial configuration as Figure 3.4.

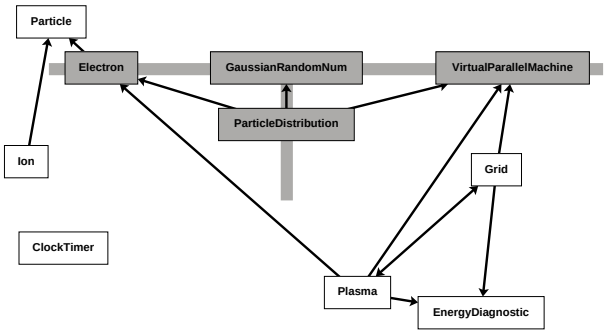


Figure A.2: User adds a *T-Shape* VOF to four nodes.

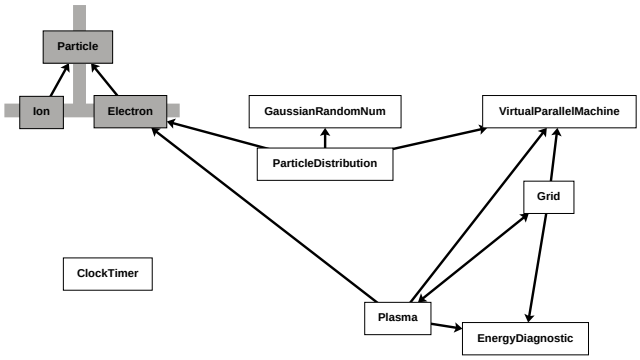


Figure A.3: User adds a second *T-Shape* VOF to the left three nodes.

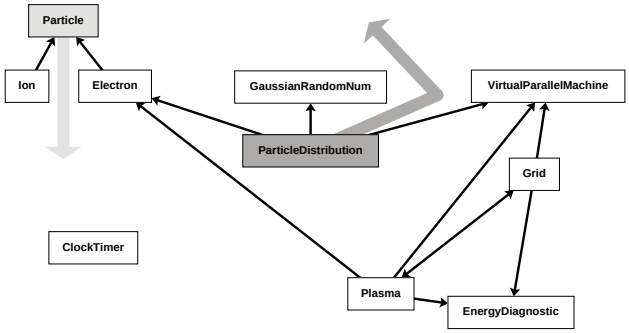


Figure A.4: User manipulates the root node of each *T-Shape* VOF.

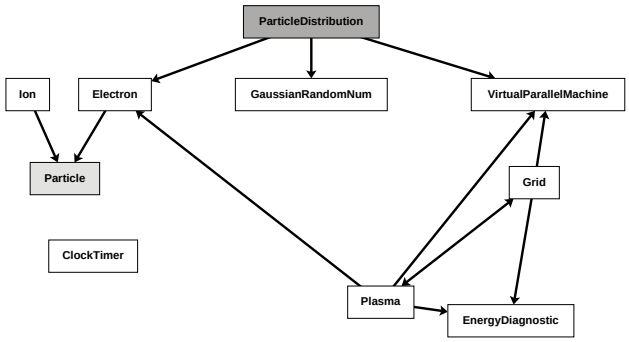


Figure A.5: The intermediary layout with the *T-Shape* VOFs in the right places.

participates in both VOFs. GLIDE satisfies these constraints as shown in Figure A.4.

Now the user would like the *T-Shape* VOFs to be oriented differently. By manually manipulating each of the root nodes, the user helps GLIDE find a more acceptable layout. This intervention is shown in Figure A.4, and results in both *T-Shape* VOFs being inverted; the system maintains the *T-Shape* constraints generating the layout in Figure A.5.

Next the user swaps two nodes by manually moving the `GaussianRandomNum` node to the right, placing it to the left of the node `VirtualParallelMachine`. GLIDE continues to satisfy the existing VOFs, and the nodes settle into the layout shown in Figure A.6.

Now that the diagram has the desired general layout, it only requires a few additional VOFs to clean up its appearance. In Figure A.7 the user adds *Vertical Alignment* and *Vertical Symmetry* VOFs. Two *Horizontal Alignment* VOFs, as shown in Figure A.8, bring the rows of the diagram into alignment. No alignment is needed on the first row due to the two previously established *T-Shape* VOFs.

All that remains is to space the rows vertically, which is accomplished by adding a *Vertical Even Spacing* VOF as indicated in Figure A.9. The final layout is given in Figure A.10.

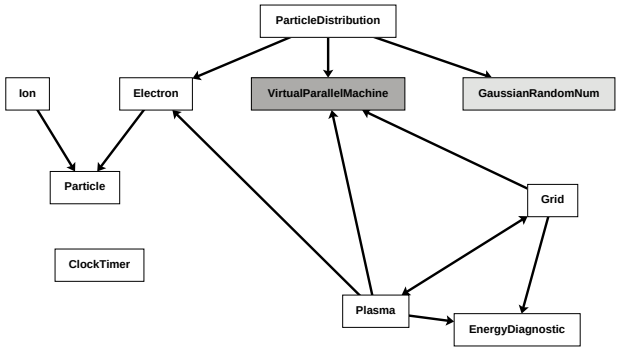


Figure A.6: User swaps the `VirtualParallelMachine` and `GaussianRandomNum` nodes.

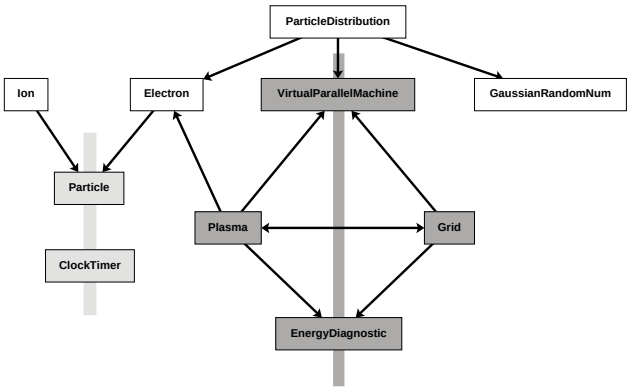


Figure A.7: User adds two more VOFS: *Vertical Alignment* and *Vertical Symmetry* .

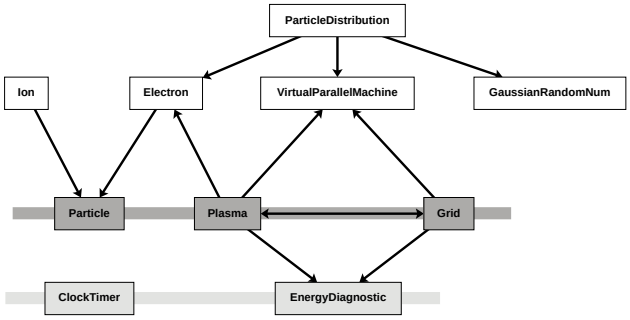


Figure A.8: User adds two *Horizontal Alignment* VOF.

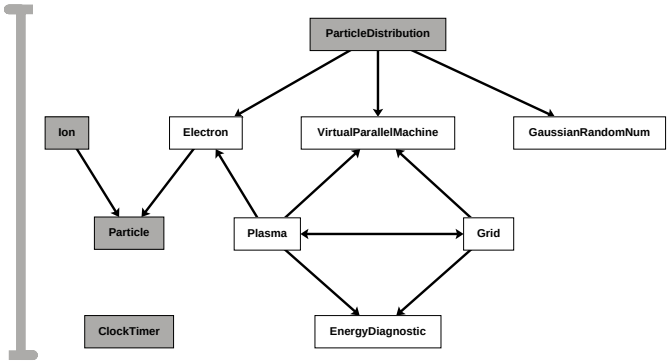


Figure A.9: User adds a *Vertical Even Spacing* VOF.

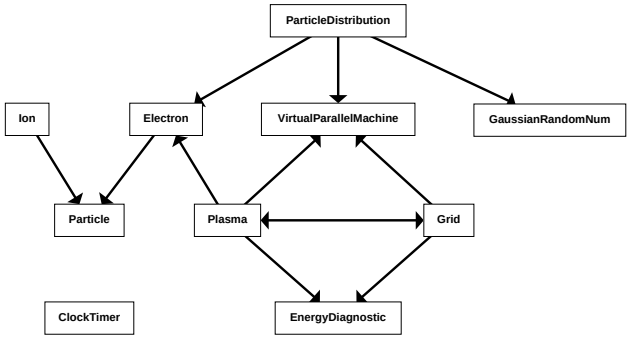


Figure A.10: The final layout.

A.2 Understanding and Organizing an Unknown Data Set

In this Section we again illustrate the use of the GLIDE system to explore design alternatives. In this case, however, the user's goal is to generate a layout for a set of data *without* a priori knowledge of what the visual organization of the diagram might be.

The data set was randomly generated and contains 29 uniformly-sized nodes, each with a unique label. The nodes were colored using 6 hues. Within a given set of same-colored nodes, each node has the same letter,[1] but a unique number in its label. The diagram contains 27 directed edges. The initial layout for the nodes was also randomly generated. Figure A.11 shows the initial configuration presented to the user.

The first two layouts, Figures A.12–A.13, were generated by a novice user. In the first layout (shown in Figure A.12), the user has grouped nodes by *color*. This diagram clearly illustrates the 6 sets of nodes, and the interconnections between them. Figure A.13 shows a second layout by the same (novice) user. This time the user's strategy was to minimize edge crossings. The red lines in both diagrams indicate the VOFs used.

The layouts in Figures A.14–A.15 were generated by an expert user. As a first strategy, the user attempted to generate a sink-to-source diagram in which all the edges within the diagram "flow" in the same direction, in this case up the page. As shown in Figure A.14, the diagram contains four disconnected components (which are not apparent in either of the novice-generated layouts). The user also noted a hub shape organization near the center of the diagram. Starting from this sink-to-source layout, the (expert) user generated the layout shown in Figure A.15. These two layouts provide a good illustration of the importance of iterative refinement component of the design process. The act of exploring one design alternative led the user to identify and generate a second layout.

The four layouts shown here are very different from each other, each conveying different structures of the underlying data set. It would be difficult to objectively evaluate and rank these diagrams. Such an ordering would depend upon what information the designer was trying to convey through the diagram. As the data set was randomly generated, there is no right answer to this question.

[1] One set of nodes contain no letter.

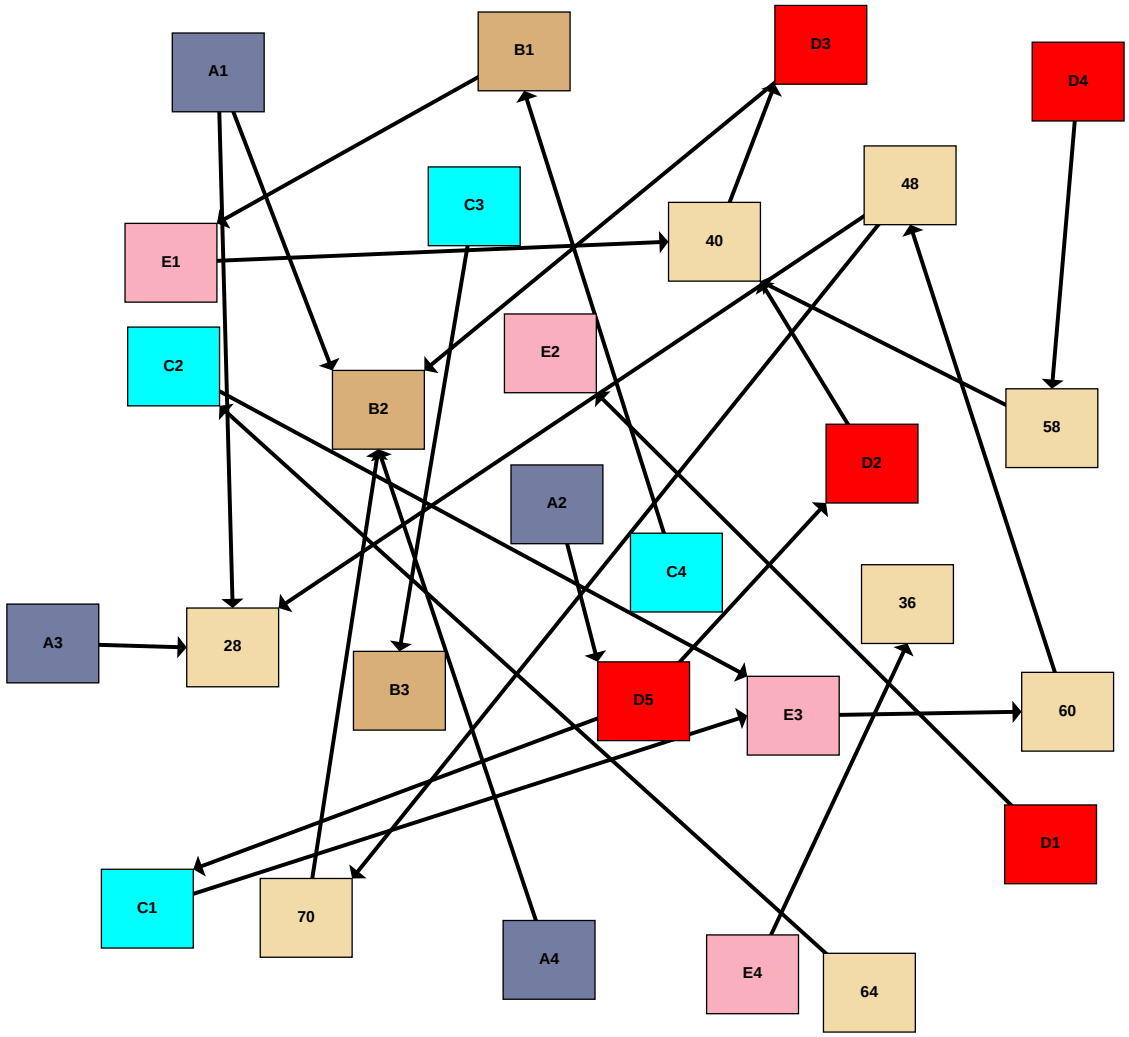


Figure A.11: A random data set with random initial layout.

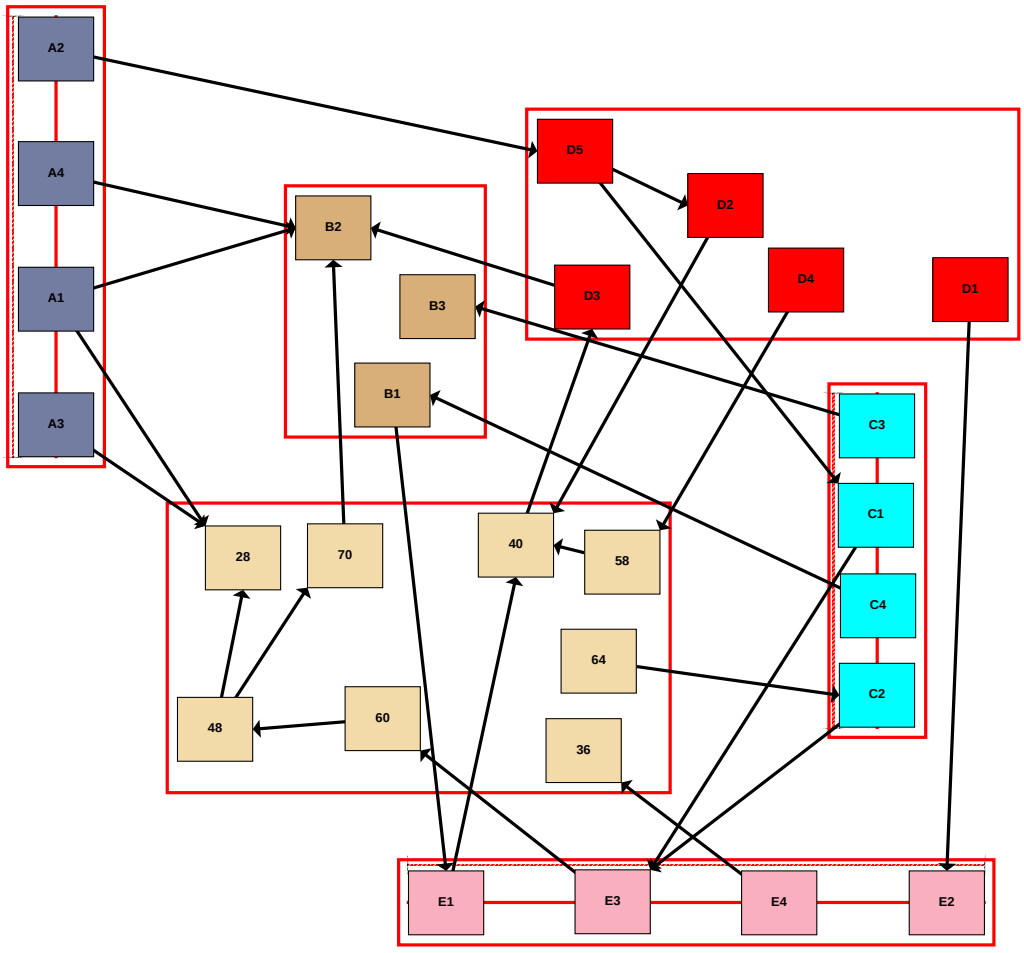


Figure A.12: A first attempt by a novice user.

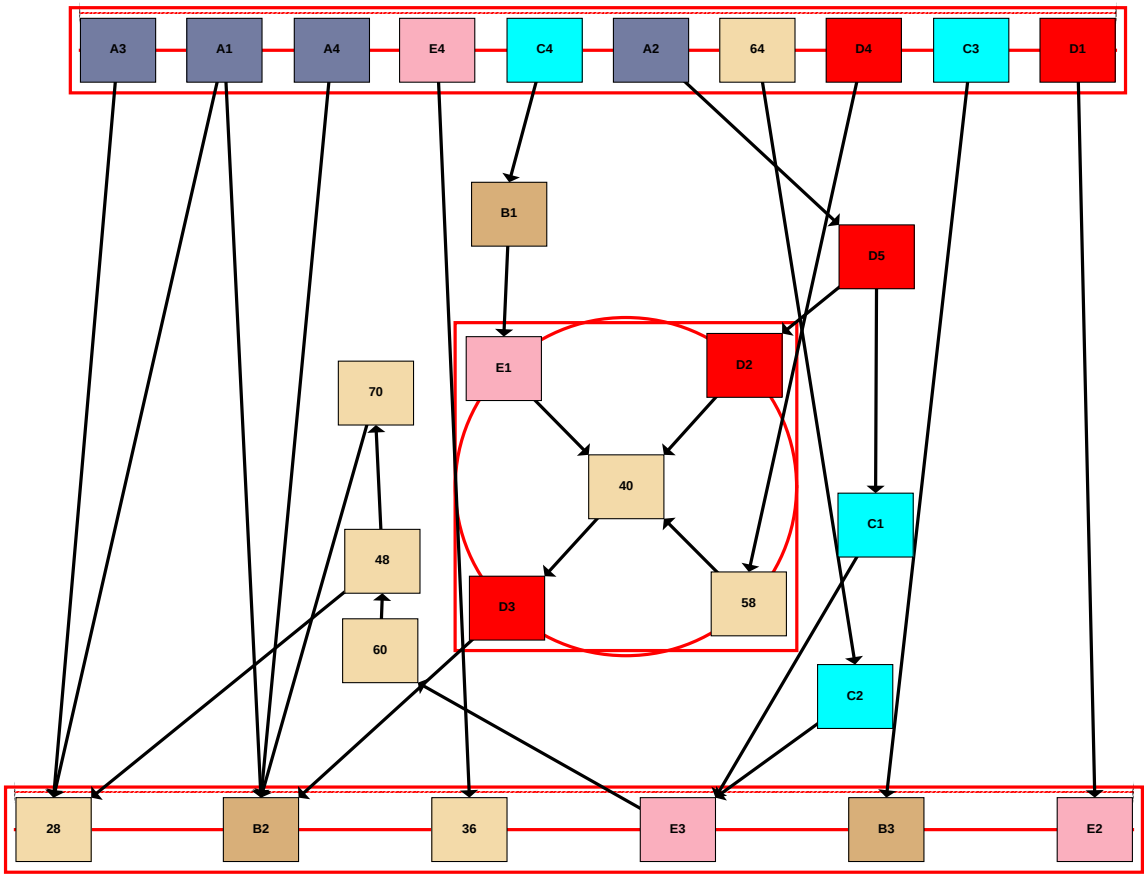


Figure A.13: A second layout by a novice user.

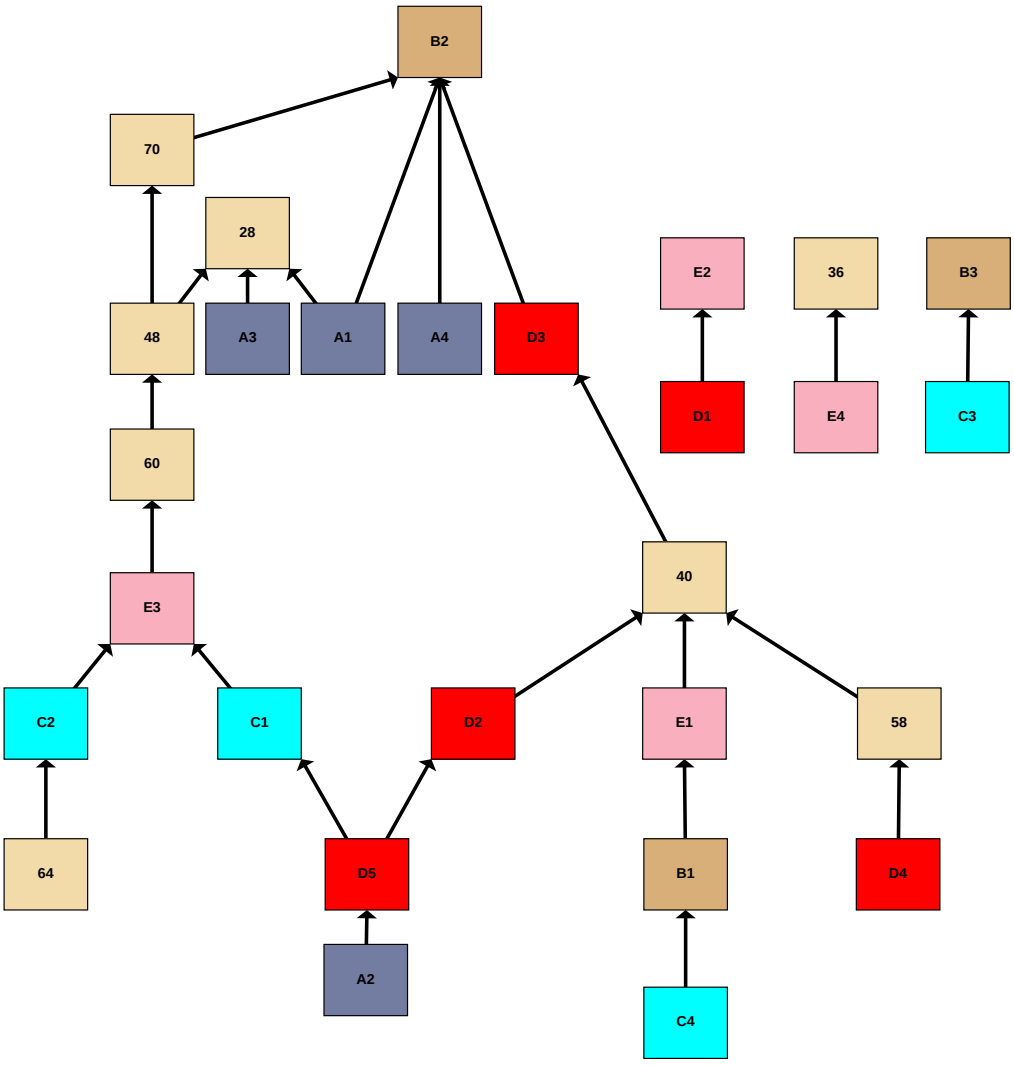


Figure A.14: A sink-to-source layout by an expert user.

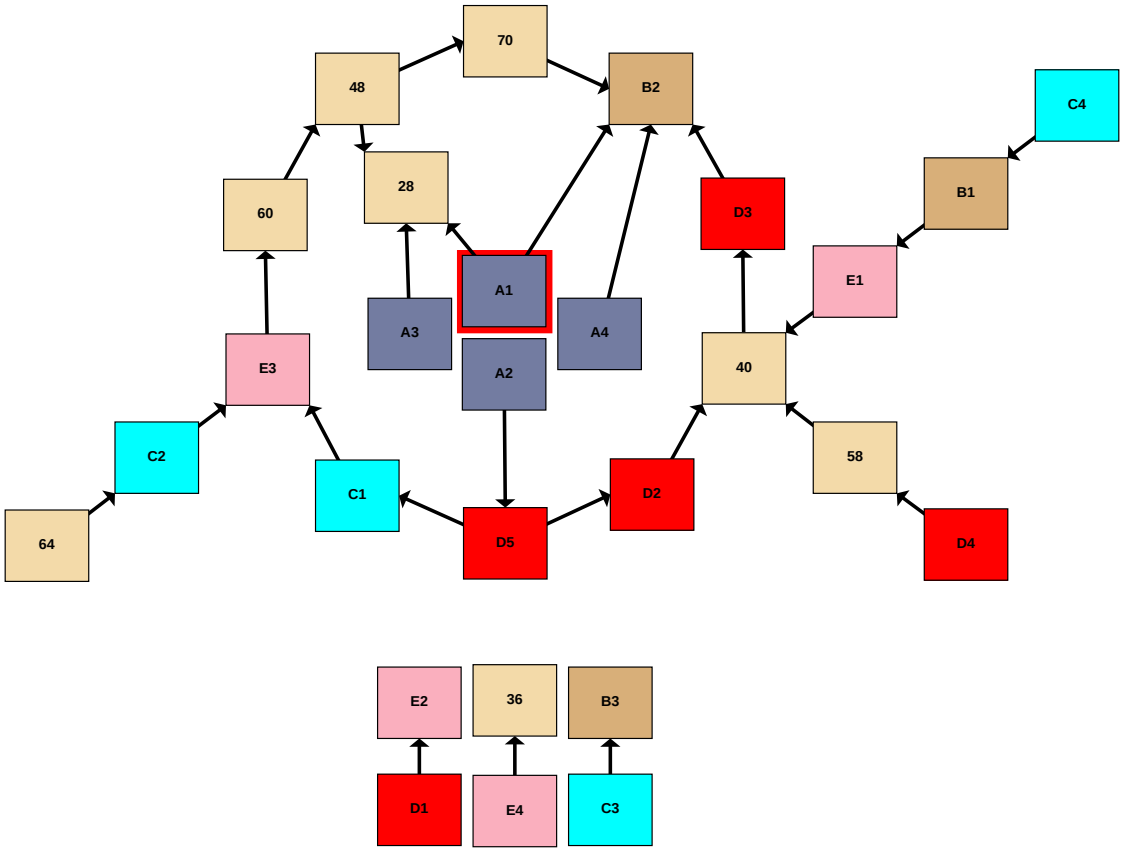


Figure A.15: A symmetry-based layout by an expert user.

Appendix B

Design Galleries: Example Application Areas

All examples in this appendix make use of the same dispersion and arrangement techniques that were used for the medical imaging example in Section 4.3 and utilize Euclidean distance as the distance metric on their output vectors. Lines have been added to each interface to show the correspondence between thumbnails and full-sized images in the gallery.

B.1 Scientific Visualization

This volume rendering Design Gallery uses the simulated electron density of a protein as its data set. Figure B.1 illustrates its interface. Both the color and opacity transfer functions were varied in the input vector, for a total of 23 input parameters. As with the example in Section 4.3, mapping is done using a volume rendering technique, and the output vector is composed of eight manually selected pixels.

Figure B.2 shows the result of clicking on one of the images in the image gallery: the corresponding opacity and color transfer functions are depicted in a pop-up window, allowing the user to see how image and data relate.

B.2 Two Dimensional Animation

The two-dimensional double pendulum is a simple dynamic system with rich behavior that makes it an ideal test case for parameter-setting methodologies. A double pendulum consists of an attachment point h, two bobs of masses m_1 and m_2, and two massless rods of lengths r_1 and r_2, connected as shown in Figure B.3. Our pendulum also includes motors at the joints at h and m_1 that can apply sinusoidal time-varying torques.[1] The input vector comprises the rod lengths, the bob masses, the initial angular positions and velocities of the rods, and the amplitude, frequency, and phase of both sinusoidal torques, for a total of 14 input parameters.

Choosing a suitable output vector proved to be the most difficult part of the Design Gallery process for the double pendulum, as well as for the other motion-control applica-

[1] Even without the application of external torques at its joints, the two-dimensional double pendulum is provably chaotic (Dullin, 1994).

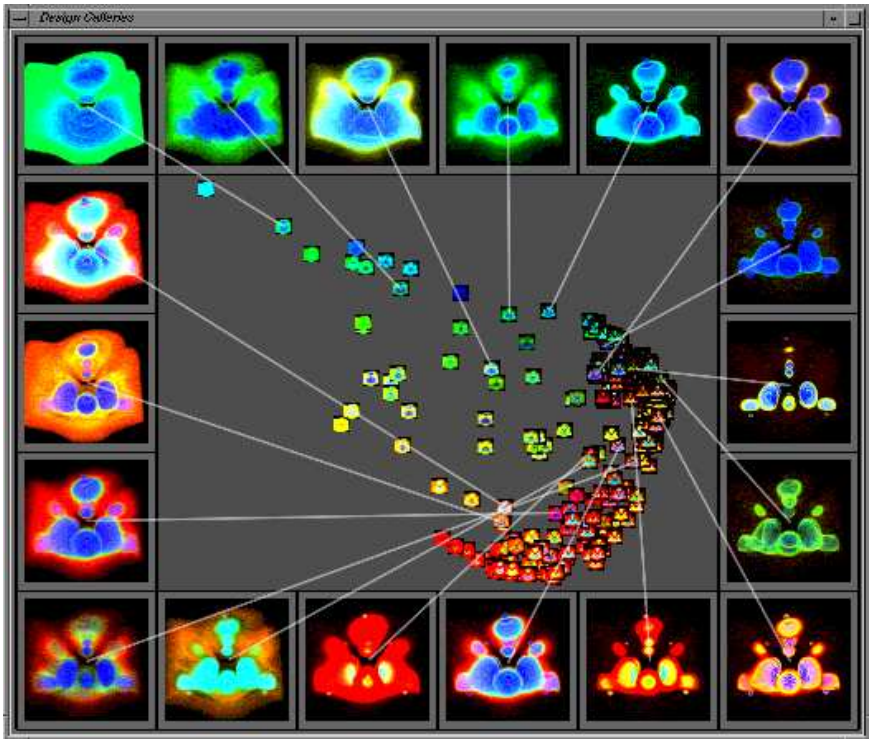


Figure B.1: A Design Gallery with different opacity and color transfer functions.

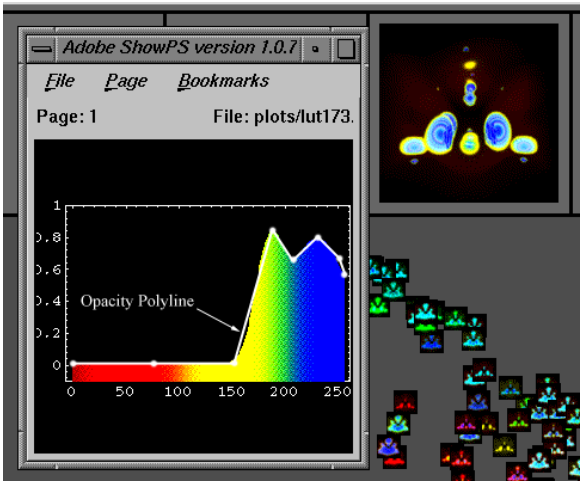


Figure B.2: Pop-up display depicting opacity and color transfer functions.

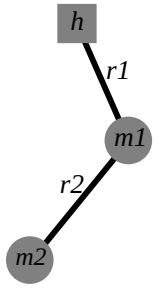


Figure B.3: A two-dimensional double pendulum.

tions; several rounds of experimentation were needed (see the paper by Marks et al. (1997) for more details). For the double pendulum, the output vector has 12 parameters: the differences in rod lengths and bob masses, the average Cartesian coordinates of each bob, and logarithms of the average angular velocity, the number of velocity reversals, and the number of revolutions for each rod. The mapping from input vector to output vector is accomplished by dynamically simulating 20 seconds of the pendulum's motion.

The Design Gallery interface shown in Figure B.4 is for the two-dimensional double pendulum. The displayed thumbnails are static images of the final state of a pendulum, along with a trail of the lower bob over the final few seconds. We found that these images give enough clues about the full animation to enable effective browsing. Dragging a thumbnail to a gallery slot places a full-sized static image corresponding to the thumbnail into the gallery. The gallery images can then be animated, permitting users to view multiple animations simultaneously.

B.3 Three Dimensional Animation

The previous Design Gallery is useful in finding and understanding the full range of motions possible for the pendulum under a given control regime. However, complete generality is not always a useful goal: the animator may have some preconceived idea of a motion that needs subtle refinement to add nuance and detail. The three-dimensional hopper dog, shown in Figure B.5, is an articulated linkage with rigid links connected by rotary joints. It has a head, ears, and tail, and moves by hopping on its single leg. It has 24 degrees of freedom (DOF). The hopper dog is actuated by a control system that tries to maintain a desired forward velocity and hopping height, as well as desired positions for joints in some of the appendages.

Seven quantities of the system are designated as quantities of interest, to be explored by the Design Gallery interface: the forward velocity, the hopping height, and the positions of 2-DOF ear joints, a 2-DOF tail joint, and a 1-DOF neck joint. For each quantity of interest, a time-varying sinusoid is chosen as the desired trajectory, with the minimum value, maximum value, and frequency specified by the input vector, giving a total of 21 values in the input vector. The hopper dog has other degrees of freedom that are not explicitly controlled by the input vector. The output vector measures the seven quantities of interest, using data captured over a 30 second physical simulation. Measures of the average and variance of each quantity of interest are recorded in the output vector, giving a total of 14

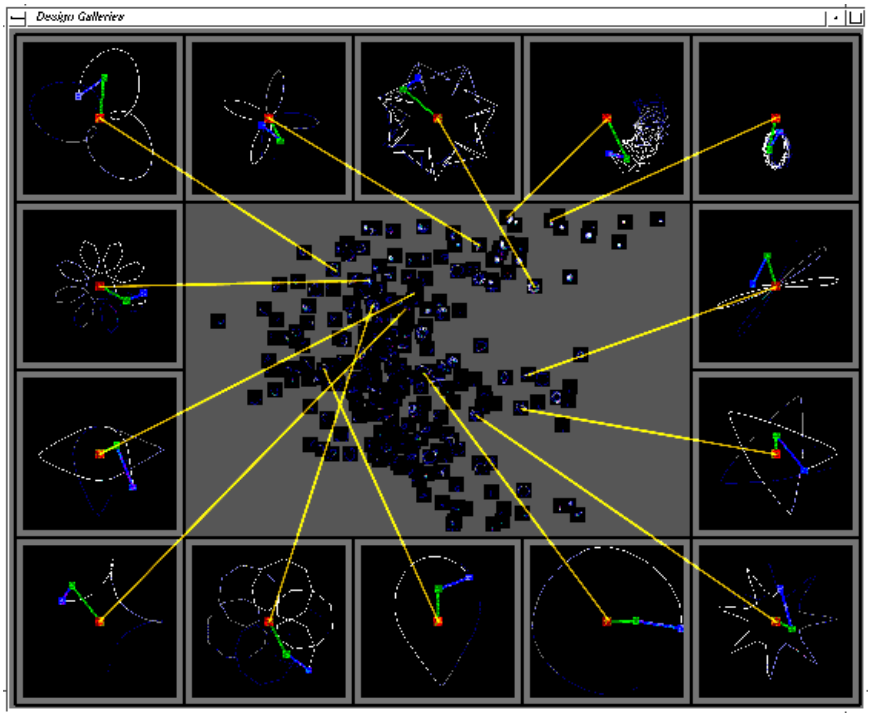


Figure B.4: A Design Gallery for an actuated two-dimensional double pendulum.

values. The equations of motion for the system are generated using a commercially available package (Rosenthal and Sherman, 1986); dynamic simulation is used to produce both the end animations and relevant mapping from input vector to output vector.

The Design Gallery interface for Hopper Dog is shown in Figure B.4. The displayed thumbnails are motion-lapse images of his ears and tail. As with the double-pendulum Design Gallery interface, images in the gallery can be animated.

B.4 Particle Systems

Particle systems are useful for modeling a variety of phenomena such as fire, clouds, water, and explosions (Reeves, 1983). A useful particle-system editor might have 40 or more parameters that the animator can set, so achieving desired effects can be tedious. As in the previous subsection, we use a Design Gallery interface to refine an animator's rough approximation to a desired animation.

The scenario for our experiment is a new beam weapon for Klingon starships. A first draft was produced by hand using a regular particle-system editor. The input vector contains the subset of particle-system controls that the animator wishes to be tweaked. The controls govern: the mean and variance of particle velocities, particle acceleration, rate of particle production, particle lifetime, resilience and friction coefficients of collision surfaces, and perturbation vectors for surface normals. Among the parameters that are held fixed in this example are the origin, average direction, and color of the beam.

The output vector comprises measures of the number of particles, their average distance

Figure B.5: HopperDog: A three-dimensional articulated dog.

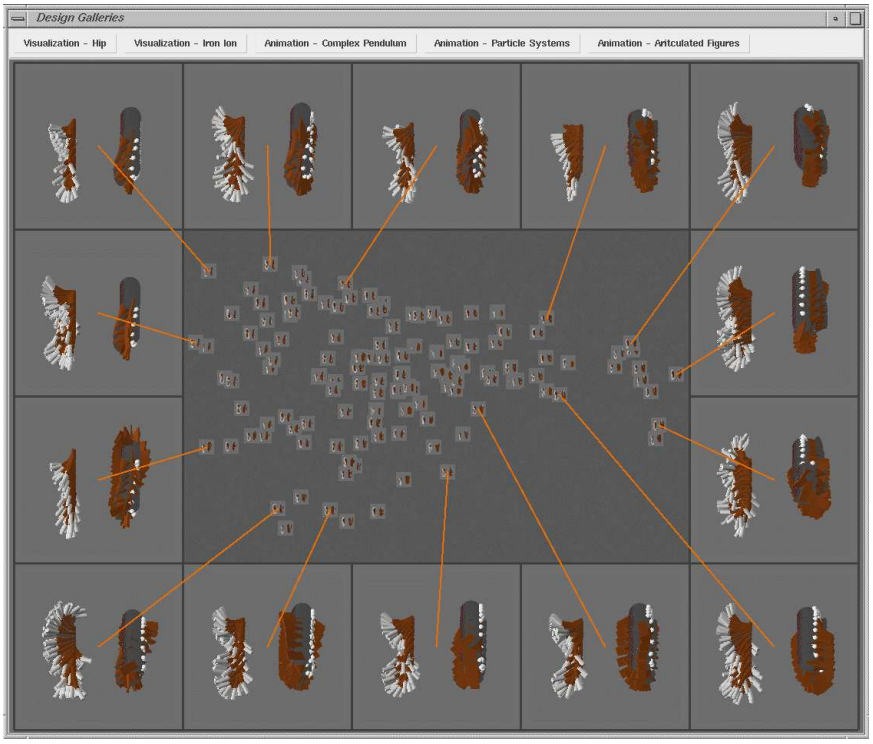


Figure B.6: Design Gallery for HopperDog.

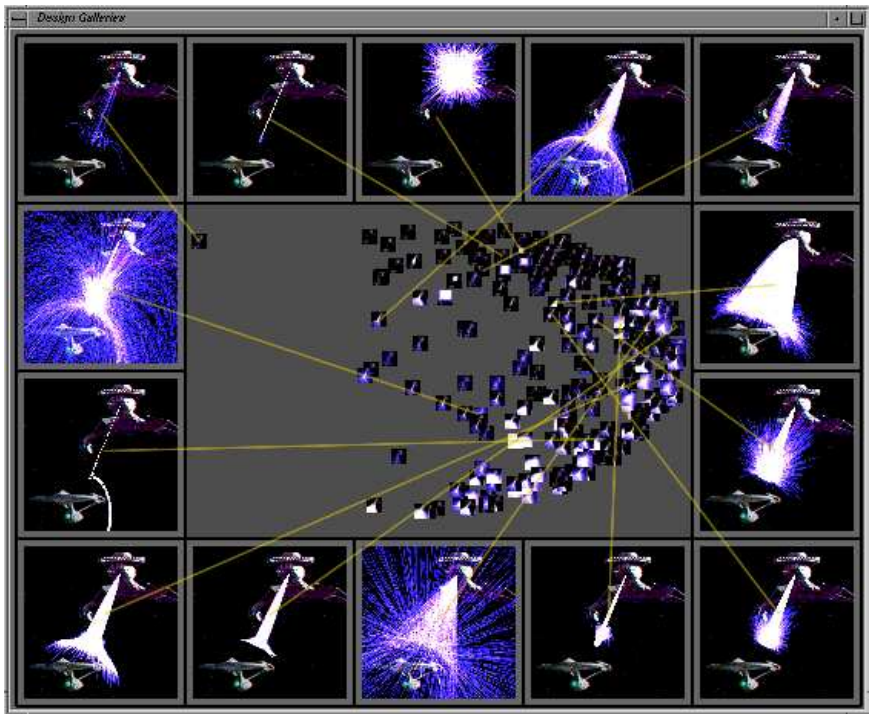


Figure B.7: A Design Gallery for a particle system.

from the origin and the individual variation in this distance, their spread from the average beam, the average velocity of the entire system, and the individual variation from this average (we take logs of all of these quantities except for the beam spread). These six measures are included for each of the two distinguished times (once midway through the simulation, and once at the end), resulting in 12 output parameters.

Each thumbnail is a static image built by compositing the animation frames. Although the resulting image does not look like any single frame of the animation, we believe users can easily establish a mental correspondence between the static images and the nature of their associated animations. As with the other animation-based DG interfaces , images in the gallery can be animated. Figure B.7 shows the Design Gallery of variations on an animator's sketch of the Klingon beam weapon.

Appendix C

Floor Plan Segmentation: Additional Example Diagrams

In Chapter 5 we described an example interaction in which the user and computer collaborated to generate a region segmentation (and corresponding graphic) for a sample floor plan. Figures C.1 and C.2 provide two additional sample floor plans. Notice the very different styles of these floor plan images. The Berkeley floor plan, Figure C.2, seems to have been derived directly from the blue prints. Many of the "walls" (i.e. boundaries) between adjacent rooms are missing pixels. The corruption is most likely due to scanning errors. In contrast, the MIT floor plan, Figure C.1, was most probably produced using a simple drawing editor. It contains partially bounded regions as well; doors are indicated by gaps in the walls, rather than by door swings. Our system works equally well with both styles of floor plans. The false colorings in the images depict the system-generated regions, without any additional user input.

Figure C.1: MIT floor plan.

Figure C.2: Berkeley floor plan.

References

Ahlberg, Christopher and Ben Shneiderman. 1994. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proceedings of Human Factors in Computer Systems (CHI)*, pages 313–317, April.

Anonymous. 1996. Toy story. Available at `http://dreistein.com/kino/infos/Toystory/notes.shtml` .

Antoine, Dominique. 1991. CIPLAN: A model-based system with original features for understanding French plats. In *Proceedings of the First International Conference on Document Analysis and Recognition*, pages 647–655, Saint-Malo, France, October.

Baecker, Ronald and Ian Small. 1990. Animation at the interface. In Brenda Laurel, editor, *The Art of Human-Computer Interface Design*. Addison-Wesley Publishing Co., Reading, MA.

Baecker, Ronald M., Jonathan Grudin, William A. S. Buxton, and Saul Greenberg. 1995. Vision graphic design and visual display. In *Readings in Human-Computer Interaction: Toward the Year 2000*. Morgan Kaufmann, second edition edition. Introduction to the chapter.

Baker, Ellie. 1997. Thesis in progress.

Baker, Ellie and Margo Seltzer. 1994. Evolving line drawings. In *Proceedings of Graphics Interface 94*.

Battista, Giuseppe Di, Peter Eades, Roberto Tamassia, and Ioannis Tollis. 1994. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry: Theory and Applications*, 4:235–282.

Baudel, Thomas. 1994. A mark-based interaction paradigm for free-hand drawing. In *Proceedings of User Interface Software and Technology (UIST)*, pages 185–192, Marina del Rey, CA, Nov.

Beucher, S. and C. Lantuejoul. 1979. Use of watersheds in contour detection. In *Proc. of the International Workshop on Image Processing, Real-Time Edge and Motion Detection/Estimation*, pages 17–21, Rennes, France, September.

Boatto, Luca, Bincenzo Consorti, and Monica Del Buono. 1992. An interpretation system for land register maps. *Computer*, 25(7):25 – 33, July.

Bohringer, Karl-Friedrich and Frances Newbery Paulisch. 1990. Using constraints to achieve stability in automatic graph layout algorithms. In *Proceedings of Human Factors in Computer Systems (CHI)*, pages 43–51.

Borning, Alan. 1979. *ThingLab – A Constraint-Oriented Simulation Library*. Ph.D. thesis, Stanford University, Stanford, CA.

Bowman, W. 1968. *Graphic Communication*. John Wiley & Sons.

Brandenburg, Franz J., editor. 1995. *Proceedings of the Symposium on Graph Drawing*, volume 1027 of *Lecture Notes on Computer Science*. Springer.

Bridgeman, Stina, Ashim Garg, and Roberto Tamassia. 1996. A graph drawing and translation service on the WWW. In Stephen North, editor, *Proceedings of Graph Drawing '96*, volume 1190 of *Lecture Notes on Computer Science*, Berkeley, CA, October. Springer Verlag.

Caldwell, C. and V. S. Johnston. 1991. Tracking a criminal suspect through face space with a genetic algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 416–421.

Casey, Richard G. and George Nagy. 1991. Document analysis: A broader view. In *Proceedings of the First International Conference on Document Analysis and Recognition*, pages 839–849, Saint-Malo, France, October.

Catmull, E. 1978. The problems of computer assisted animation. In *Proceedings of SIGGRAPH 78*, pages 348–353. In *Computer Graphics* Annual Conf. Series, 1978.

Chai, Ian and Dov Dori. 1992. Extraction of text boxes from engineering drawings. In *Machine Vision Applications in Character Recognition and Industrial Inspection*, volume 1661, pages 38–49, San Jose, California, February.

Chi, H. U. 1985. Formal specification of user interfaces: a comparison and evaluation of four axiomatic approaches. *IEEE Transactions on Software Engineering*, 11(8):671–685.

Christensen, Jon. 1995. *Managing Design Complexity: Using Stochastic Optimization in the Production of Computer Graphics.* Ph.D. thesis, Harvard University, June.

Christensen, Jon, Joe Marks, and Stuart Shieber. 1995. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics*, 14(3):203–232, July.

Clement, T. P. 1981. The extraction of line-structured data from engineering drawings. *Pattern Recognition*, 14(1):43–52.

Dawkins, Richard. 1987. *The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design.* W. W. Norton and Company.

Dengler, Ed, Mark Friedell, and Joe Marks. 1993. Constraint-driven diagram layout. In *Proceedings of the 1993 IEEE Symposium on Visual Languages*, pages 330–335, Bergen, Norway, Aug.

Dix, A., J. Finlay, G. Abowd, and R. Beale. 1993. *Human-Computer Interaction.* Prentice Hall.

Dondis, D. 1973. *A Primer of Visual Literacy.* MIT Press, Cambridge, MA.

Donelson, William. 1978. Spatial management of information. In *Proceedings of SIGGRAPH 78.* In *Computer Graphics* Annual Conf. Series, 1993.

Dori, Dov. 1989. A syntactic/geometric approach to recognition of dimension in engineering machine drawings. *Computer Vision, Graphics, and Image Processing*, 47:271–291.

Dori, Dov. 1991. Symbolic representation of dimensioning in engineering drawings. In *Proceedings of the First International Conference on Document Analysis and Recognition*, pages 1000–1010, Saint-Malo, France, October.

Dullin, Holger R. 1994. Melnikov's method applied to the double pendulum. *Zeitschrift für Physik B*, 93:521–528.

Eades, Peter. 1984. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160.

Edmondson, Shawn, Jon Christensen, Joe Marks, and Stuart Shieber. 1997. A general cartographic labeling algorithm. *Cartographica*. To appear.

Fletcher, L. A. and R. Kasturi. 1988. A robust algorithm for text string separation from mixed text/graphics images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):900–918.

Frohlich, Michael and Mattiaas Werner. 1994. Demonstration of the interactive graph visualization system da Vinci. In *Proceedings of the Workshop on Graph Drawing 94*, pages 266–269.

Furnas, G. W. 1981. The fisheye view: a new look at structured files. Technical report, Bell Laboratories.

Furnas, George. 1986. Generalized fisheye views. In *Proceedings of CHI 86*, pages 16–23.

Furnas, George and Benjamin Bederson. 1995. Space-scale diagrams: Understanding multiscale interfaces. In *Proceedings of CHI 95*, pages 234–241.

Ganser, Emden R., Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. 1993. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, March.

Gleicher, Michael and Andrew Witkin. 1994. Drawing with constraints. *Visual Computer*, 11:39–51.

Grudin, Jonathan. 1993. Interface: an evolving concept. *Communications of the Association for Computing Machinery*, 36(4):110–119, April. Special Issue on Graphical User-Interfaces.

He, Taosong, Lichan Hong, Arie Kaufman, and Hanspeter Pfister. 1996. Generation of transfer functions with stochastic search techniques. In *Proceedings of Visualization 96*, pages 227–234, San Francisco, California, Oct.

Heydon, Allan and Greg Nelson. 1994. The Juno2 constraint-based drawing editor. Technical Report 131a, Digital SRC, Palo Alto, CA.

Himsolt, Michael. 1994. Graphed: A graphical platform for the implementation of graph algorithms. In *Proceedings of the Workshop on Graph Drawing 94*, pages 182–193.

Hower, Walter and Winfried H. Graf. 1995. Research in constraint-based layout, visualization, CAD and related topics: A bibliographic survey. Available at: `http://`.

Johnson, D. S. 1982. The NP-completeness column: an ongoing guide. *Journal of Algorithms*, 3(1):89–99.

Johnson, D. S. 1984. The NP-completeness column: an ongoing guide. *Journal of Algorithms*, 5(2):147–160.

Joseph, S. H. 1989. Processing of engineering line drawings for automation input to CAD. *Pattern Recognition*, 22(1):1–11.

Joseph, S. H. 1991. On the extraction of text connected to linework in document images. In *Proceedings of the First International Conference on Document Analysis and Recognition*, pages 993–999, Saint-Malo, France, October.

Kang, Tom, Josh Seims, Joe Marks, and Stuart Shieber. 1995. Exploring lighting spaces. Technical report, MERL: A Mitsubishi Electric Research Lab.

Kass, Michael, Andrew Witkin, and Demetri Terzopoulos. 1988. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331.

Kasturi, Rangachar and J. Alemany. 1988. Information extraction from images of paper-based maps. *IEEE Transactions on Software Engineering*, 14(5):671–675.

Kasturi, Rangachar, S. T. Bow, W. Elmasri, J. R. Gattiker, and U. B. Mokate. 1990. A system for interpretation of line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:978–992.

Kawai, John K., James S. Painter, and Michael F. Cohen. 1993. Radioptimization – goal-based rendering. In *Proceedings of SIGGRAPH 93*, pages 147–154, Anaheim, California, Aug. In *Computer Graphics* Annual Conf. Series, 1993.

Kochhar, Sandeep. 1990. A prototype system for design automation via the browsing paradigm. In *Proceedings of Graphics Interface 90*, pages 156–166, Halifax, Nova Scotia, May.

Kochhar, Sandeep. 1991. *Cooperative computer-aided design: a paradigm for automating the design of graphical objects.* Ph.D. thesis, Harvard University, Cambridge, MA.

Kosak, Corey, Joe Marks, and Stuart M. Shieber. 1994. Automating the layout of network diagrams with specified visual organization. *IEEE Trans. on Systems, Man and Cybernetics*, 24(3):440–454, March.

Kosslyn, Stephen M. 1989. Understanding charts and graphs. *Applied Cognitive Psychology*, 3:185–226.

Kruskal. 1977. Multidimensional scaling and other methods for discovering structure. In K. Enslein, A. Ralston, and H. S. Wilf, editors, *Statistical Methods for Digital Computers*, volume 3. Wiley, New York, pages 296–339.

Lai, Chan Pyng and Rangachar Kasturi. 1993. Detection of dimension sets in engineering drawings. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, pages 606–613, Tsukuba Science City, Japan, October.

Liu, Zicheng, Steven J. Gortler, and Michael F. Cohen. 1994. Hierarchical spacetime control. In *Proceedings of SIGGRAPH 94*, pages 35–42, Orlando, Florida, July. In *Computer Graphics* Annual Conf. Series, 1994.

Mackinlay, Jock D. 1986. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, April.

Marcus, Aaron. 1983. Graphical design for computer graphics. *IEEE Computer Graphics and Applications*, 3(4):63–70.

Marcus, Aaron. 1992. A comparison of graphical user interfaces. In A. Marcus, editor, *A Graphic Design for Electronic Documents and Usr Interfaces*. ACM Press.

Marcus, Aaron. 1995. Principles of effective visual communication for graphical user interface design. In *Readings in Human-Computer Interaction: Toward the Year 2000*. Morgan Kaufmann, second edition, pages 425–441.

Marks, J., B. Andalman, P. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. 1997. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of SIGGRAPH 97*, Los Angeles, California, Aug. To appear.

Marks, J. and M. Reiter. 1990. Avoiding unwanted conversational implicatures in text and graphics. In *Proceedings of the eigth national conference on artificial intelligence (AAAI) 90*, Boston, MA.

Marks, Joe. 1991. *Automating the Design of Network Diagrams*. Ph.D. thesis, Harvard University, Cambridge, MA, May.

Marks, Joe and Stuart Shieber. 1991. The computational complexity of cartographic label placement. Technical Report TR-05-91, Harvard University, Cambridge, MA.

McWilliams, James. 1995. Computer animation. `http://www.htimes.com/htimes/today/access/oldfiles/animate.html`.

Moran, T. 1981. The command language grammar: a representation for the user interface of interactive systems. *International Journal of Man-Machine Studies*, 15(1):3–50.

Myers, Brad A., Jade Goldstein, and Matthew A Goldberg. 1994. Creating charts by demonstration. In *Proceedings of Human Factors in Computer Systems (CHI)*, pages 106–111, Boston, MA, April.

Nardelli, E. and G. Proietti. 1993a. Advanced techniques for cadastral maps interpretation. Technical Report R.364, Instituto di Analisi dei Sistemi ed Informatica, Rome, Italy, September.

Nardelli, E. and G. Proietti. 1993b. An algorithmic approach to the interpretation of floorplan maps. Technical Report R.365, Instituto di Analisi dei Sistemi ed Informatica, Rome, Italy, September.

Nardelli, Enrico, Michelangelo Fossa, and Guido Proietti. 1993. Raster to object conversion aided by knowledge based image processing. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, pages 951–954, Tsukuba Science City, Japan, October.

Nelson, Greg. 1985. Juno, a constraint based graphics system. *Computer Graphics (Proceedings of SIGGRAPH '85)*, 19(3):235–243, July.

Newbery, F. J. 1988. Edge: An extendible directed graph editor. Technical Report 8/88, University of Karlsruhe, Institute for Informatics, Germany, June.

North, Stephen, editor. 1996. *Proceedings of the Symposium on Graph Drawing*, volume 1190 of *Lecture Notes on Computer Science*. Springer.

Norton, Charles D., Boleslaw K. Szymanski, and Viktor K. Decyk. 1995. Object-oriented parallel computation for plasma simulation. *CACM*, 38(10):88–100, October. Figure 3.

O'Gorman, Lawrence and Rangachar Kasturi. 1992. Special issues on document image analysis systems and techniques. *Computer*, 25(7), July.

Ousterhout, John K. 1994. *Tcl and the Tk Toolkit*. Addison Wesley.

Pao, Derek, H. F. Li, and R. Jayakumar. 1991. Graphic features extraction for automatic conversion of engineering line drawings. In *Proceedings of the First International Conference on Document Analysis and Recognition*, pages 533–541, Saint-Malo, France, October.

Perlin, Ken and David Fox. 1993. Pad: An alternative approach to the computer interface. In *Proceedings of SIGGRAPH 93*, pages 57–64, Anaheim, California, Aug. In *Computer Graphics* Annual Conf. Series, 1993.

Pixar. 1997. Pixar's recruiting page. `http://www.pixar.com/jobsite/recruiting.html`.

Plass, Michael Frederick. 1981. *Optimal Pagination Techniques for Automatic Typesetting Systems*. Ph.D. thesis, Stanford University.

Poulin, Pierre and Alain Fournier. 1992. Lights from highlights and shadows. In *Proceedings of the 1992 Symposium on Interactive Graphics*, pages 31–38, Boston, Massachusetts, Mar. In *Computer Graphics* 25(2), 1992.

Preece, Jenny. 1994. *Human-Computer Interaction*. Addison-Wesley.

Press, William H., 1988. *Numerical Recipes in C*, chapter Minimization or Maximization of Functions. Cambridge University Press.

Reeves, W. T. 1983. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. on Graphics*, 2:91–108, Apr.

Rosenthal, D. E. and M. A. Sherman. 1986. High performance multibody simulations via symbolic equation manipulation and Kane's method. *Journal of Astronautical Sciences*, 34(3):223–239.

Roth, Steven F., John Kolojejchick, Joe Mattis, and Jade Goldstein. 1994. Interactive graphic design using automatic presentation knowledge. In *Proceedings of Human Factors in Computer Systems (CHI)*, pages 112–117, Boston, MA, April.

Roth, Steven F. and Joe Mattis. 1990. Data characterization for intelligent graphics presentation. In *Proceedings of Human Factors in Computer Systems (CHI)*, pages 193–200, Seattle, WA, April.

Rubner, Y., C. Tomasi, and L. J. Guibas. 1997. Adaptive image embeddings for database navigation. Submitted to the CVPR Workshop on Content-Based Access of Image and Video Libraries, June.

Rubner, Yossi, Leonidas Guibas, and Carlo Tomasi. 1997. The earth mover's distance, multi-dimensional scaling, and color-based image retrieval. In *Proceedings of the ARPA Image Understanding Workshop*, May.

Ryall, Kathy, Joe Marks, Murray Mazer, and Stuart Shieber. 1993. Annotating floor plans using deformable polygons. Technical Report TR-24-93, Harvard University.

Schoeneman, Chris, Julie Dorsey, Brian Smits, James Arvo, and Donald Greenberg. 1993. Painting with light. In *Proceedings of SIGGRAPH 93*, pages 143–146, Anaheim, California, Aug. In *Computer Graphics* Annual Conf. Series, 1993.

Seligmann, D. and S. Feiner. 1991. Automated generation of intent-based 3D illustrations. *Computer Graphics*, 25(4):123–132, July. (Proceedings ACM SIGGRAPH '91, Las Vegas, NV, July 28-August 2, 1991).

Sims, Karl. 1991. Artificial evolution for computer graphics. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, volume 25, pages 319–328, Las Vegas, Nevada, July.

Sims, Karl. 1994. Evolving virtual creatures. In *Proceedings of SIGGRAPH 94*, pages 15–22, Orlando, Florida, July. In *Computer Graphics* Annual Conf. Series, 1994.

Sistare, Steven. 1990. *A graphical editor for three-dimensional constraint-based geometric modeling.* Ph.D. thesis, Harvard University.

Sistare, Steven. 1991. Interaction techniques in constraint-based geometric modeling. In *Proceedings of Graphics Interface '91*, pages 85–92, Calgary, Alberta, June.

Smith, J. R. 1991. Designing biomorphs with an interactive genetic algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 535–538.

Sugiyami, K., S. Tagawa, and M. Toda. 1981. Methods for visual understanding of hierarchical system structures. *IEEE Trans. on Systems, Man and Cybernetics*, 11(2):109–125, Feb.

Sutherland, Ivan. 1963. *Sketchpad: a man-machine graphical communication system.* Ph.D. thesis, MIT.

Suzuki, Satoshi and Toyomichi Yamada. 1990. MARIS: Map recognition input system. *Pattern Recognition*, 23(8):919–933.

Tamassia, R., G. D. Battista, and C Batini. 1989. Automatic graph drawing and readability of diagrams. *IEEE Trans. on Systems, Man and Cybernetics*, 18(1):61–79, Jan./Feb.

Tamassia, Roberto and Ioannis G. Tollis, editors. 1994. *Proceedings of the Symposium on Graph Drawing*, volume 894 of *Lecture Notes on Computer Science*. Springer.

Tang, Diane, J. Thomas Ngo, and Joe Marks. 1995. N-body spacetime constraints. *Journal of Visualization and Computer Animation*, 6(3):143–154.

Thomas, Bruce H. and Paul Calder. 1995. Animating direct manipulation interfaces. In *Proceedings of User Interface Software and Technology (UIST) '95*, pages 3–12, Pittsburgh, PA, Nov.

Todd, Stephen and William Latham. 1992. *Evolutionary Art and Computers.* Academic Press, London.

Tom Sawyer Software Corporation. 1991. Graph layout toolkit. Berkeley, CA.

Torgerson, Warren S. 1958. *Theory and Methods of Scaling.* Wiley, New York. See especially pages 254-259.

Tufte, E. 1983. *The visual display of quantitative information.* Graphics Press.

Tufte, E. 1990. *Envisioning Information.* Graphics Press.

Tufte, E. 1997. *Visual Explanations: Images and quantities, evidence and narrative.* Graphics Press.

University of Passau. 1997. Graphlet: A toolkit for implementing graph editors and graph drawing algorithms. Home page at `http://www.uni-passau.de/Graphlet/`.

van de Panne, Michiel and Eugene Fiume. 1993. Sensor-actuator networks. In *Proceedings of SIGGRAPH 93*, pages 335–342, Anaheim, California, Aug. In *Computer Graphics* Annual Conf. Series, 1993.

Ventrella, Jeffrey. 1995. Disney meets Darwin – the evolution of funny animated figures. In *Proceedings of Computer Animation 95*, pages 35–43, Apr.

Vincent, L. and P. Soile. 1991. . watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–597, June.

Wegner, Peter. 1997. Why interaction is more powerful than algorithms. *CACM*, 40(5):80–91, May.

Wiley. 1994. How dogs interpret floor plans... (Non-Sequitur). Distributed by the *Washington Post Writers Group*, a syndicated service of *The Washington Post*.

Witkin, Andrew and Michael Kass. 1988. Spacetime constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, volume 22, pages 159–168, Atlanta, Georgia, Aug.

Wong, K. Y., R. G. Casey, and F. M. Wahl. 1982. Document analysis system. *IBM Journal on Research and Development*, 26(6):647–656, November.

Yuille, Alan L., Peter W. Hallinan, and David S. Cohen. 1992. Feature extractions from faces using deformable templates. *International Journal of Computer Vision*, 8(2):99–111.