Large Scale Relational Information
Visualization, Clustering, and Abstraction

by

Aaron J. Quigley

B.A. (Mod.) (Trinity College Dublin) 1995

A thesis submitted to

The Department of Computer Science and Software Engineering

The University of Newcastle

for the degree of

DOCTOR OF PHILOSOPHY

August, 2001

Declaration

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

Aaron J. Quigley

Newcastle

August 2001

For my Father and Mother, John and Noelle,

Forever in my mind

Acknowledgements

I would first like to thank my supervisor, Professor Peter Eades, for his clear guidance, encouragement, wise counsel, and unwavering support over the past few years. His influence has been invaluable in my development as a researcher. I truly owe him a great debt. My thanks also to my co-supervisor Associate Professor A.S.M Sajeev for his support, comments, and suggestions in the completion of this thesis.

I would also like to thank the Department of Computer Science and Software Engineering for first offering me a Research Scholarship and later a position as a staff member. The logistical and financial support provided for equipment and to attend many conferences has greatly added to my breadth of knowledge as a researcher.

Next, I would like to offer my thanks to many people at the University of Newcastle: the office staff, for providing a friendly face; the systems staff, for providing an excellent environment; my tutors, for being excellent at what they do; my students, for being understanding while I did this work; and to Ljiljana for being a friend.

My thanks also go to Ira Baxter at Semantic Designs for offering me a research internship during the course of this work, to the University of Limerick for granting me a research stay and finally to the Basser Department of Computer Science for giving me a visiting scholar position, during the completion of this thesis. Thanks also to Dr. Rainer Koschke and Dr. Ronald F. Boisvert for providing me with much of the raw data and comparative images used in my case studies.

For their love and inspiration, which has guided me through every crucial stage in my life, I would like to thank my parents, who are gone but never forgotten. I know you are watching over me. Finally, I would like to express my love and eternal devotion to my family, friends, and Brad, you complete me.

Contents

List of Figures

List of Tables

Abstract

The past decade has seen an explosion in the amount of information that is collected, analyzed, explored, or simply stored in the hope that one day it can be used. The rate at which we can collect and store data is rapidly outstripping the provision of tools for the effective analysis and exploration of such data. This thesis is concerned with the provision of models and methods for such tools.

Relational information visualization is concerned with the presentation of *abstract* relational data, in a visual form. The essential idea in relational information visualization is that the user's perceptual abilities are employed to understand and explore such information. Visually, humans can perceive more patterns linking local features in the data.

Graph models are typically used to represent relational information, where the visualization of such graphs is referred to as graph drawing. Existing models and methods for graph drawing tend to effectively deal with only relatively small graphs (at most a hundred nodes). This thesis is concerned with investigating efficient techniques for drawing large graphs with thousands of nodes.

Efficiency in the production of the drawing is only one of the many related issues when dealing with large graphs. Equally significant are the problems of screen space use, the cognitive load on the user, and the time to render the picture. Further, the picture created must be a measurably high quality. The central paradigm presented in this thesis marries a solution to all these problems using a single graph model.

This thesis provides the models, measures and methods required to produce and evaluate the drawing of large amounts of relational information. The effectiveness and efficiency of the methods are evaluated with rigorous quality measures using data from application domains.

Introduction

"...all life is only a set of pictures in the brain, among which there is no difference betwixt those born of real things and those born of inward dreamings, and there is no cause to value one above the other." - H.P. Lovecraft

1.1 Purpose

A byproduct of the explosive growth in the use of computing technology is that organizations are generating, gathering, and storing data at a rate which is doubling every year [298]. The ability for a mid-sized organization to store terabytes of data is easily within reach. The provision of massive storage technology is rapidly outstripping the provision of tools for the effective analysis and exploration of such voluminous data. Clearly this data is of little value unless useful information and hence knowledge can be derived from it.

Application domains which deal with such voluminous data include: geographic information systems (see [220, 282]), geophysical data systems (see [46, 155, 205]), financial analysis systems (see [278]), software development (see [37, 161, 211]), software reverse engineering (see [199, 221, 238]), and software evolution (see [236, 246, 296]).

The amount of data makes the analysis task difficult. One approach to this problem is to convert the data into pictures and models that can be graphically displayed. The intuition behind the use of such graphics is that human beings are inherently skilled at understanding data in visual forms.
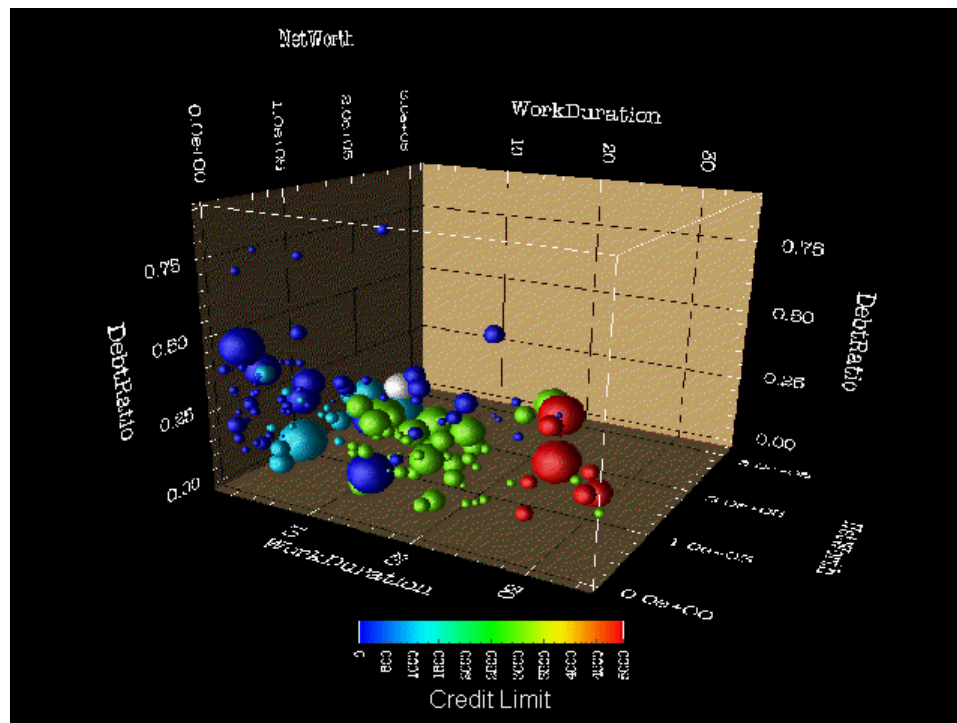
Figure 1.1: Credit Card Application Data Visualization. Reproduced by courtesy of Frank Suits (IBM)

For example, Figure 1.1 shows the results obtained for a set of credit card applications. The colour of the sphere is the credit limit for that card holder as determined by a software analysis. A banker can analyze such data in this visual form to see patterns that may help with marketing or risk analysis. For example, the vast majority of those with a long work history have a large credit limit, regardless of current debt ratio.

It is becoming increasingly apparent that more powerful graphical information exploration tools are required as the amount and complexity of data that these tools are expected to handle steadily increases. Large scale information visualization is the process of graphically representing large amounts of abstract information on screen, which a user can interpret in ways not possible from the raw data alone. *This thesis is concerned with the graphical display of large amounts of information*.

In some application domains the information space can be modeled in terms of its atomic *entities* and their *interrelationships*, that is, as *relational information*. Techniques which produce graphical representations or abstract views of such relational information now form a substantive component of many graphical software systems. Examples of do-
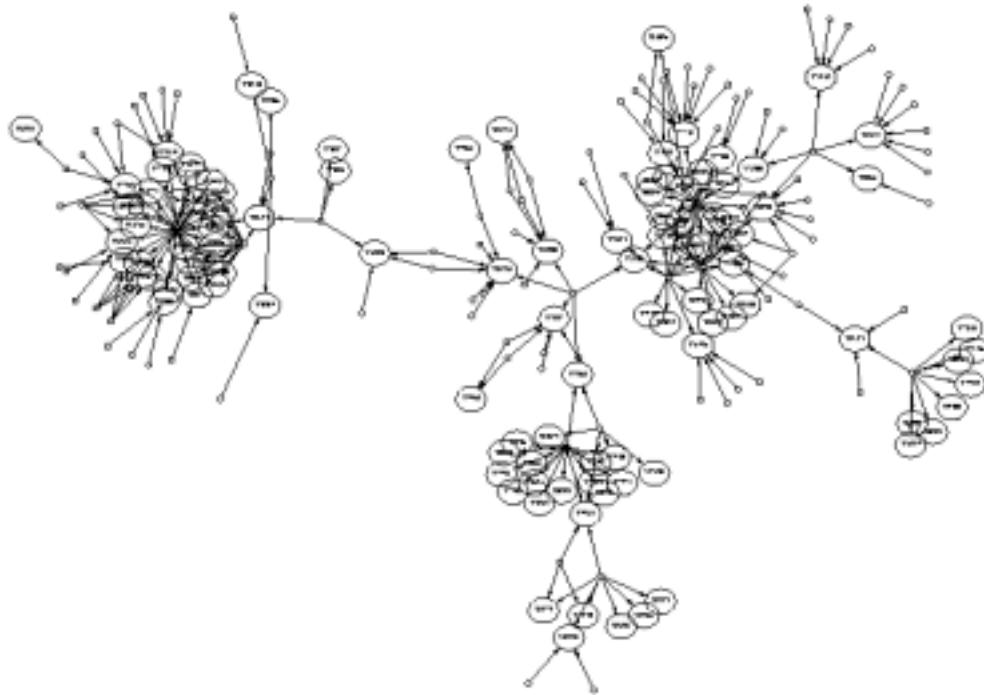
Figure 1.2: Dependencies between modifications to a large program. Reproduced by courtesy of Stephen North (AT&T), from [133]

mains which include the use of graphical relational information display include: software understanding (see [3, 12, 42, 198, 211, 242]), program comprehension (see [219, 101, 165, 174, 176, 263]), software visualization (see [11, 12, 37, 24, 143, 227, 260]), and web site maintenance (see [135, 168, 200, 203, 296, 297]). *This thesis is concerned with the visualization of large amounts of relational information from such domains.*

Relational information is typically modeled in terms of a graph $\mathcal{G}$; the atomic entities of the domain form the set of nodes $\mathcal{N}$ and the interrelationships form the set of edges $\mathcal{E}$. An example of a picture of relational information is shown in Figure 1.2, where the edges represent dependencies between modifications to a large software system. Note that in this picture the natural clusters of the software modules are apparent.

In contrast, consider the relational information drawn in Figure 1.3, which aims to show the Internet infrastructure in South Korea. In this drawing, the relationships (which are crucial in understanding the actual infrastructure) are difficult to identify and follow. The layout of this graph is of poor quality. The problem of creating a high quality picture
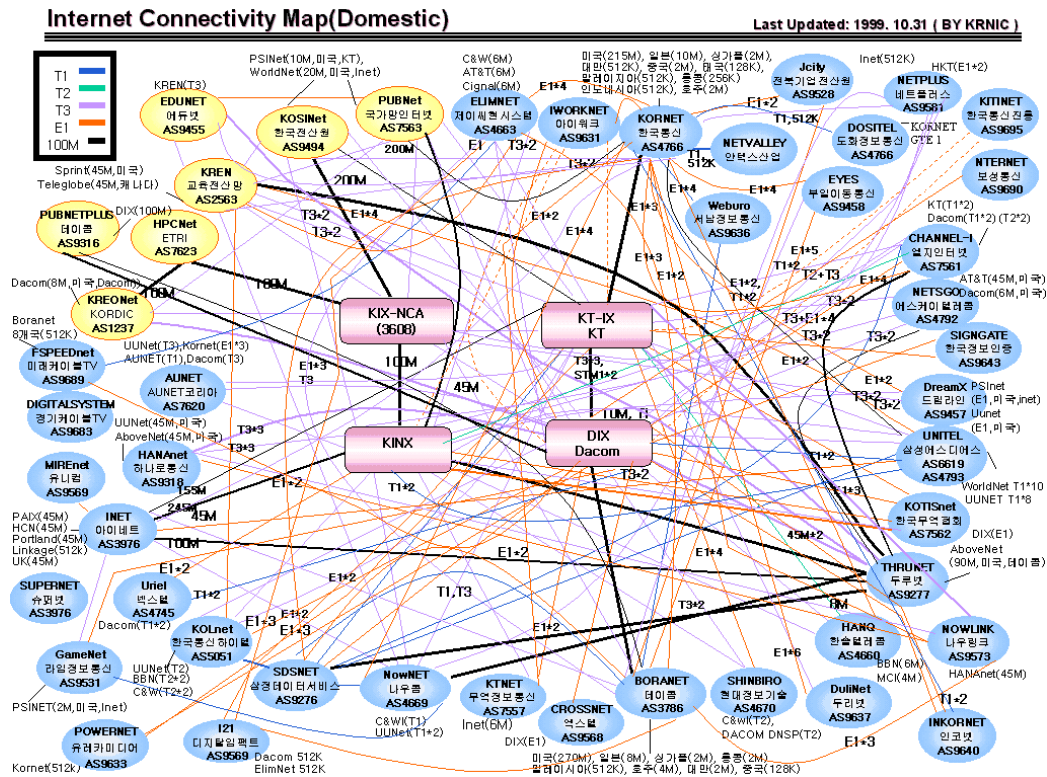
Figure 1.3: Internet infrastructure in South Korea, NIC Korea

of a graph, is to assign a location for each node and a route for each edge, so that the picture is easy to follow; this is the classical problem in *Graph Drawing* [60]. *This thesis is concerned with drawing large graphs, that is, graphs with thousands of nodes and edges*.

A good visual representation of a graph can effectively convey information to the user but a poor representation can confuse or worse, mislead [74, 229, 230, 289]. Graph drawing aims to develop algorithms and methods that produce high quality pictures that are easy to follow. Generally, there are four related problems when dealing with the visualization of large graphs. A brief discussion of each problem follows.

1. Computation: Classical graph drawing algorithms and approaches tend to deal only with relatively small graphs. This has resulted in the development of techniques which are unable to scale when drawing larger graphs. The primary bottleneck is the large amount of computational effort these methods require to layout even medium sized graphs. The graph in Figure 1.4 is a simplified model of a small part of the Internet (28107 nodes and
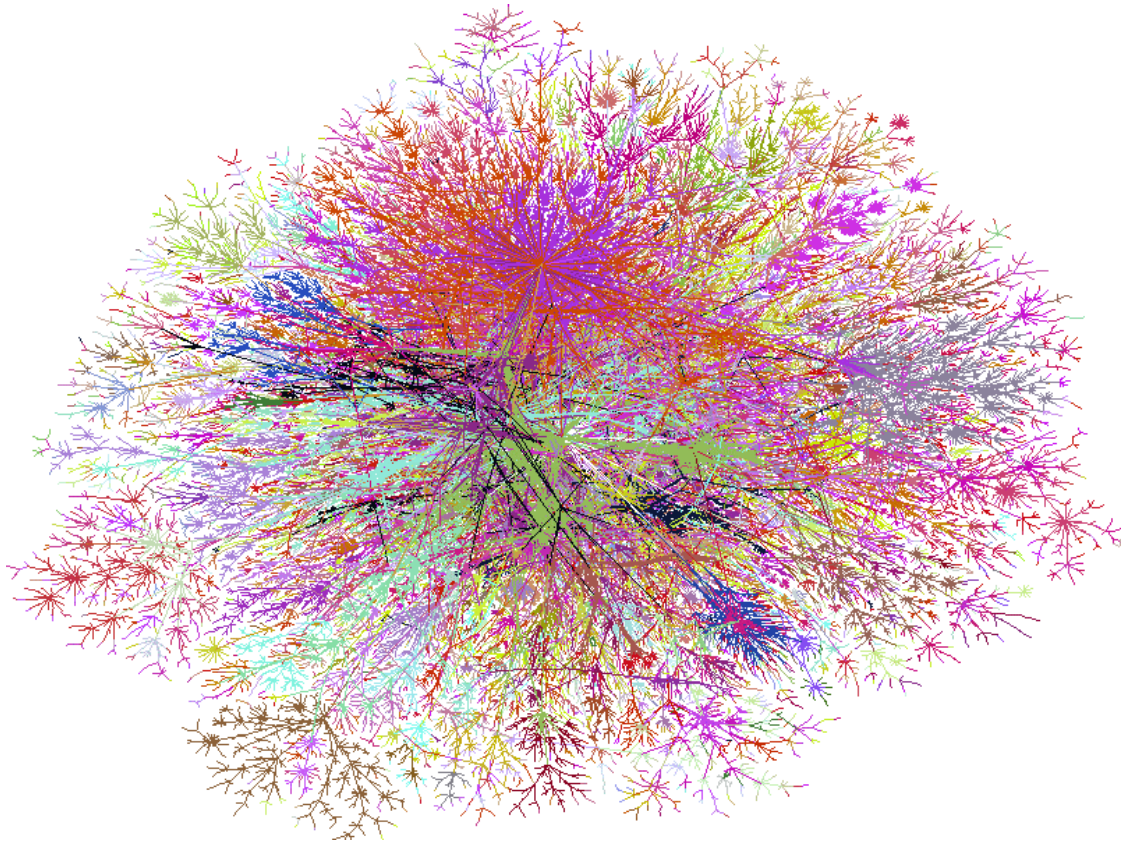
Figure 1.4: Artistic visualization of internet connectivity and geography, courtesy of Bell Labs

29664 edges). This drawing took 20 CPU hours on a 400 Mhz Pentium to layout. The time complexity of the algorithm used to produce the layout in Figure 1.4 is $\theta(n^2)$, where n is the number of nodes [69]. Even with expected increases in processor speed, one cannot expect to use this algorithm for real time layout. Fast layout algorithms are required, even for large graphs, as interactive applications require real-time response. The computational efficiency of a graph drawing algorithm and the size of the graph to render are crucial factors for any practical technique.

2. Screen Space: The challenging problem of making fast algorithms is further compounded by the need to make effective use of the screen space. Showing part of the entire layout in detail or zooming out to fit the entire drawing are common techniques. There are a variety of other visualization techniques which attempt to fit large amounts of relational information onto a computer screen. Sophisticated interactive systems employ the recur-

sive use of *glyphs* to elide parts of the drawing [289].

3. Cognitive Load: Related to the problem of the effective use of screen space is the issue of load, in terms of cognition. Even if the problems of computational cost and screen space can the solved, there is clearly a need to reduce the *cognitive* load placed on the user caused by drawing too much extraneous information. When dealing with large amounts of information, the overriding desire is to simplify the drawing to highlight the global structures while deemphasizing the irrelevant detail.

4. Rendering: Interactive drawing systems often draw each node and edge as a unique element, underpinned by a query model which supports the interactive selection of nodes and edges. As the graphs become large, it is difficult to maintain responsive realtime updates if thousands to tens of thousands individual graphical elements must be redrawn each time a large modification to the visualization occurs.

This thesis addresses the four problems of; computation, screen space, cognitive load, and rendering.

In addition, this thesis is concerned with the *quality* of the pictures produced. It is important to remember Tufte's assertion [278] that:

> "...if a visualization isn't worth a thousand words, the hell with it"

Thus along with the ability to quickly layout and concisely abstract the graphs, we require hard measures to evaluate the drawings and the abstract representations produced.

1.2   Contributions of this Thesis

The goal of this thesis is to develop models, measures, and methods which enable good visualizations of large amounts of relational data.

As such, the foremost contribution of this thesis is the establishment of the FADE visu-

alization paradigm, that marries rapid graph drawing, geometric clustering, visual abstraction, and measurement based on a dimension free hierarchical compound graph model.

The specific contributions of this thesis are:

- The introduction of computationally inexpensive methods to create the hierarchical compound graph. The goal is to rapidly form a clustering of the graph that exhibits relatively high cohesiveness and low coupling.

- A suite of fast approximate force directed algorithms, based on this graph model, to layout and display the natural clusters in a graph. These algorithms provide a significant algorithmic improvement to the classical force directed graph drawing algorithm for large graphs.

- The notion of a *visual précis* based on this model. This notion admits a range of abstract views of the underlying graph drawing, reducing both the visual weight and the time to render the drawing. Such views provide a skeleton of the graph at various levels of abstraction. Larger graphs can be represented more concisely, on a higher level of abstraction, with fewer graphics on screen.

- Quality measures of the model, the drawings, and précis from the model.

We have evaluated and validated this graph model and the FADE paradigm in two case studies, each of which involves the visualization of large amounts of application domain specific relational information. We use a prototype graph drawing system which implements the FADE algorithms based on the hierarchical compound graph model. The results and subsequent discussion of these results are supporting contributions of this thesis. The case studies are:

- A *Software Visualization* case study, which presents graph drawings, visual précis, aesthetic, and clustering results of large graphs extracted from a range of views of different software systems.

- A *Matrix Market* case study, which presents a comparative study of existing visualizations with FADE graph drawings and visual précis along with aesthetic and clustering results.

1.2.1 Publications

Some the ideas presented in this thesis have already been published in the following papers:

† Aaron Quigley and Peter Eades 2000 [237]:

"FADE*: Graph Drawing, Clustering, and Visual Abstraction*" in Joe Marks ed.: *Proc. 8th Int. Symp. Graph Drawing, GD* (20-23 Sep 2000; Williamsburg, USA); Springer-Verlag, *Lecture Notes in Computer Science, LNCS 1984:197-210* (ISBN: 3-540-41552-8).

† Aaron Quigley, Margot Postema and Heinz Schmidt 2000 [238]:

"*ReVis: Reverse Engineering by Clustering and Visual Object Classification*", in *Proc. Australian Software Engineering Conference, ASWEC2000* (28-29 April 2000; Canberra, Australia); IEEE Press, Australian National University, pp. 119-125 (ISBN:0-7695-0631-3).

† Aaron Quigley and Peter Eades 1999 [236]:

"PROVEDA*: A scheme for Progressive Visualization and Exploratory Data Analysis of Clusters*" in *Proc. Software Visualization Workshop, SOFTVIS'99* (1-2 Dec 1999; Sydney, Australia); University of Technology Sydney, pp. 67–74 (ISBN: 0-7259-1081-X).

† Aaron Quigley 1999 [233]:

"*Automated Tool Support for a large scale diagramming Tool*", in *Proc. 2nd Australian Work. on Constructing Software Engineering Tools, AWCSET'99* (1st October 1999; Sydney, Australia); Macquarie University Sydney, pp. 55-58 (ISBN: 0 864 18 573 1).

† Aaron J. Quigley [239]:

"*Large Scale 3D Clustering and Abstraction*" in Jesse Jin ed.: *Proc. Pan-Sydney Area Work. On Visual Information Processing, VIP2000, to appear* (1-2 Dec; Sydney, Australia).

1.3 Summary

The rest of this thesis is arranged as follows:

⬦ Chapter 2 introduces the visualization and presentation background details for the areas addressed in this thesis. Graph drawing, aesthetics, algorithms, and prior work on large scale graph drawing and clustering are described.

⬦ Chapter 3 develops models, methods, and measures for clustering and visual abstraction used within the FADE paradigm. The use of a "hierarchical compound graph" for graph clustering and quality measurement is presented. A geometric approach to the creation of hierarchical compound graphs and the notion of "visual précis" are described.

⬦ Chapter 4 describes the FADE paradigm and a suite of layout algorithms based on the models and methods introduced in Chapter 3. The FADE suite of efficient layout algorithms is derived from work in the domain of particle simulations which we review. The FADE paradigm marries the drawing, clustering, and abstract representation of graphs into a "progressive cycle" of measurable improvement.

⬦ Chapter 5 presents a case study on Software Visualization. The aim is to present the results of applying the layout algorithms, abstraction methods, and various measures from the FADE paradigm to the drawings of "views" from graphs extracted from a variety of software systems. The visualization of these views may help a reverse engineer or software evolver identify structures and natural clusters within the drawing of the system.

⬦ Chapter 6 presents a case study on Matrix Market Visualizations. We first aim to compare FADE drawings to existing structure and cityplot visualizations. The second aim is to present the results of applying the layout algorithms, abstraction methods, and various measures from the FADE paradigm to the drawings of the structured, semi-structured, and clustered matrices from a range of application domains. The visualization of this matrix data allows analysts to identify patterns, structures, and the natural clusters within these data sets.

⋄ Chapter 7 completes this thesis by summarizing the main findings and suggesting possible future research.

⋄ Appendix A gives a description of the pictures, videos, three dimensional models, and graph data included on the CD-ROM accompanying this thesis.

⋄ Appendix B gives a description of the performance results, horizon measurements, and clustering measurements from the case studies in Chapters 5 and Chapter 6 included on the CD-ROM accompanying this thesis.

Background

"Drawing is speaking to the eye; talking is painting to the ear." - Joseph Joubert

This thesis introduces the FADE paradigm where the goal is to provide fast layout algorithms and efficient abstract representations for large graphs of thousands of nodes. The definition of what constitutes a large graph varies considerably and has tended to evolve at the same rate as computing power increases. Whereas papers describing large graphs with 50 to 100 nodes in the early 80's were not uncommon, it is now commonly accepted that large graphs constitute thousands to tens of thousands of nodes. A related problem, is the issue of "screen real estate". *Screen real estate* is simply a way to describe the amount of screen area a visualization system has, as with physical real estate it is a scarce and expensive commodity that is not to be wasted.

In this Chapter we provide background details for relational information visualization and graph drawing. Section 2.2 describes graph drawing and some common graph drawing conventions. Several broadly accepted aesthetic criteria, for graph drawing, are described in Section 2.2.2. Graph drawing algorithms are discussed in Section 2.2.3 with particular emphasis in Section 2.2.4 on the "force directed" class of algorithm on which the FADE paradigm is based. Section 2.2.6 introduces the relevant background for clustering in graph drawing. And finally, Section 2.2.7 reviews recent developments in multilevel graph drawing methods.

We begin in Section 2.1 by introducing and reviewing visualization and specifically relational information visualization. Next, in Section 2.1.3 we describe "visual abstraction"
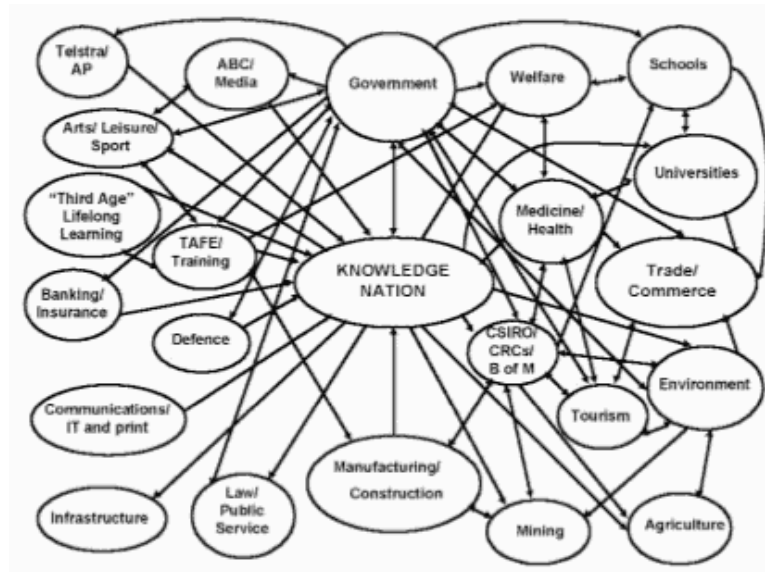
Figure 2.1: "Knowledge Nation" diagram from the Australian Labor Party [144]

and how it relates to large scale information visualization.

2.1 Visualization

Visualization is classically defined as the process of forming a mental image of some scene as described [293]. However, since the advent of graphic workstations it has become synonymous with the computational process of making data visible. With graphic workstations now so ubiquitous in almost every aspect of day to day life, it is prosaic to try and motivate interest in visualization by stating that images are a powerful way to show data.

We know visualization is important; what is more important to address is the question of "readability". Bad information visualizations are unfortunately all too common [89]. Examples include the widely scorned "meatballs and spaghetti" diagram [144] shown in Figure 2.1 from the Australian Labor Party. An equally poor visualization of the interrelationships between Korean Internet service providers is shown in Figure 1.3.

The central question of visualization is not, "do we use graphics to represent information?", rather it is "How do we create graphical presentations that are easy to understand and which effectively and efficiently convey information?".
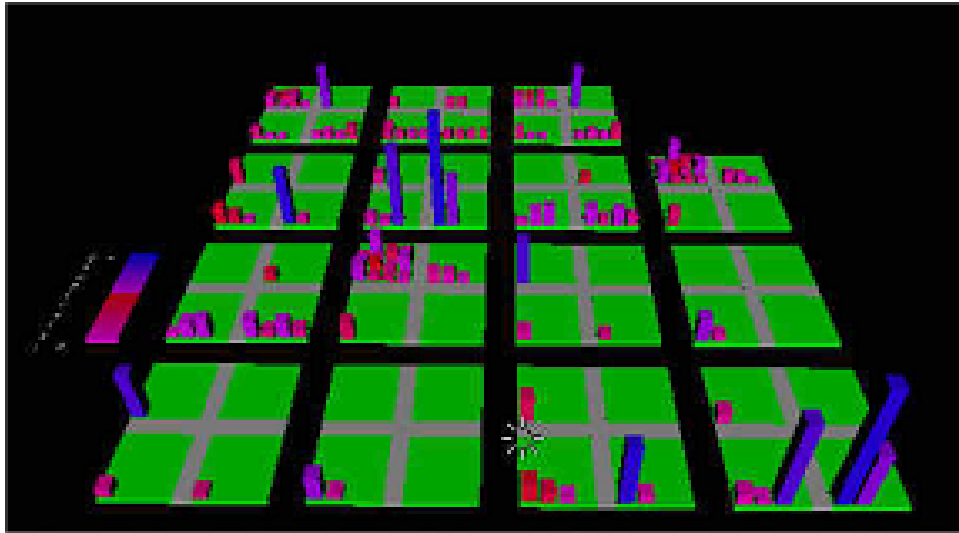
Figure 2.2: Visualization of a vector size measure of software [250].

2.1.1   Information Visualization

A number of visualization tools can be used to render a scene underpinned by a geometric model in two or three dimensions, such as the POV-RAY scene rendering tool [175] . This differs from *information visualization* which is the graphical presentation of *abstract* data. Classical visualizations such as those found in medical texts are often based on the presentation of a geometric model in some simplified form. With information visualization there is no *a priori* geometric model; rather there is abstract data in the form of a *symbolic model* of information. A symbolic model consists of a set of symbols which represent actual elements of information. To allow the symbolic model to be visualized it can be assigned a geometry. This geometry allows abstract concepts or measures to be visualized; see for example the *vector size measure of software* in Figure 2.2. In this visualization, the square regions represent modules of a software system. The height of each tower expresses the size of individual components within each module. And the colour of a tower represents its relative complexity.

An important issue in visualization is *readability*, that is, the degree to which something is intelligible and can easily be understood. Readability is not a measure of artistic worth or visual appeal. Historically, the question of readability has been confined to the field of typography, in which issues of design, arrangement, style, and appearance of type are

Figure 2.3: Alphabet of capital Roman letters with metaphorical ornaments by *Daniel Hopfer*

among the factors affecting the readability of text. Several readability measures for textual documents are available. These measures quantify the structure of language use and its relative level of complexity. These measures include the *Flesch reading ease* measure and the *Flesch-Kincaid grade level* measure [90]. For the purposes of this thesis, the need to communicate the *meaning* of the text is much more important than the need to convey a sense of *style*, that is, we desire function over form.

An artistic example of where function has lost out to form can be seen in Figure 2.3. This is a visually appealing type face with heavy embellishment but is not intended as a book face. It cannot be said that this representation *efficiently* conveys the letters of the English alphabet. Other, less ornate representations convey the information just as clearly with much less "visual ink" [277, 278].

This thesis is concerned with the readability of diagrams used to graphically represent relational information.

2.1.2 Relational Information Visualization

Relational information consists of elements of information and their interrelationships. Typically this is modeled in terms of a graph of nodes and edges. For complex relational
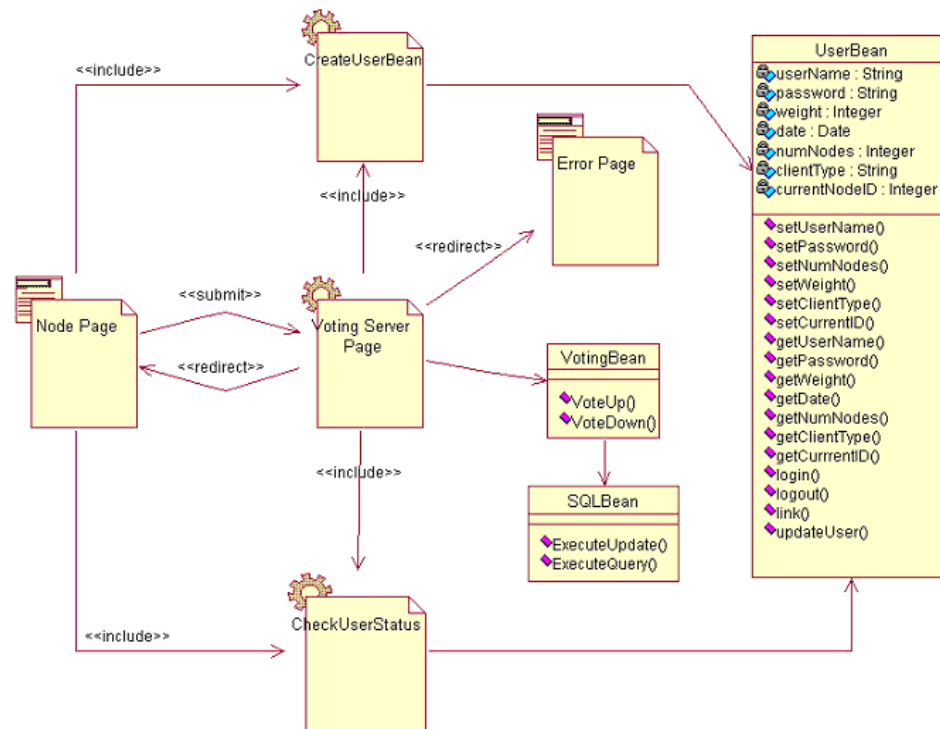
Figure 2.4: Class diagram of a web based software system, courtesy of Team 10 from the SENG205 Liquid Miro project [234]

information other graph models such as an "attributed graph" can be used. An *attributed graph* is used to model information which contains more than one attribute per data element. A relational information visualization is hence a simplified drawing that conveys the relations, elements, and attributes of the underlying abstract symbolic model of the information. An example of a relational information visualization is shown in Figure 2.4. This represents a software design in terms of classes, their attributes, and interrelationships. The shape of each element in this diagram indicates whether it is a client, server or form page on the WWW, or whether the element is a software component in the classical sense. The arcs indicate either software actions (such as getting the user name) or WWW actions (such as redirecting a URL request).

2.1.3   Visual Abstraction

In data modeling, *abstraction* is the process of deriving the essential features of the data. Abstraction, from the Latin *abstrahere* meaning "to withdraw", indicates that an *abstrac-*

tion process should remove unnecessary detail, to create the abstract form of the data which typically highlights its essential features.

Visual abstraction is the process of creating an image which departs from representational accuracy, to some extent. Abstract artists, such as Pablo Picasso, often used abstraction to select and then exaggerate or simplify the forms suggested by the world around them. The simplified drawings in medical illustrations, architectural sketches, and subway maps all employ some degree of visual abstraction to create the simplification. For the purposes of this thesis, an *illustration* is a picture or diagram that is used to clarify text.

A *photorealistic computer visualization* is the creation of an image that matches the underlying model as closely as possible. The model must therefore be very detailed, with lighting, reflection, transparency, and atmospheric conditions if it is to be displayed as realistically as possible. Hence, the model used must be very detailed. Intuitively the visualization should look like a photograph of the scene. *Synthetic Photorealism* is a large part of the field of computer graphics and has numerous conferences, symposia, and dedicated journals each year, such as SIGGRAPH [13, 59, 132, 170].

Abstract information typically has no photorealistic equivalent. For example, the vector size measure visualization shown in Figure 2.2 is used in cost and effort estimation of software projects [250]. The software modules, component sizes, and complexities have been mapped to this artificial simplified cityscape geometry. Often the visualization of abstract relational information is closer to the notion of illustration than photorealisim.

Examples of visual abstraction are common in medical illustration, architectural sketching, and subway cartography (see Figure 2.6); although these visualizations do not attempt to create photorealistic results. These visual abstractions are simplified drawings that convey the information in the underlying geometric model of an object. As such, they have been successfully used in a wide variety of application domains for hundreds of years.

In general, drawings of a model or information differ from photorealistic visualizations in terms of context sensitivity, information filtering, information hiding, visual distortion, elision, aesthetic appeal, and user control as described below.

Context sensitivity

Medical illustrations of sections of the human brain typically show an overview with fine levels of detail where they are needed. This *detail-in-context* view allows certain parts of the visualization to be selected then emphasized (while other parts are deemphasized). Often the "context" of the drawing can be at a different scale, so there is more space for the detailed parts of the illustration.

In this thesis, we extend this notion of context to "visual précis" (drawings) that encompass both a *local-* and *global-*context for a variety of different viewing schemes, as described in Section 3.11.

Filtering

Depending on the domain, filtering is often used before any visualization takes place. Elements of the model can be assigned an *a priori* importance or classification type, then only elements above a certain threshold or in a particular category are considered. For example, an architectural sketch of a new building may be based on a large underlying model that contains information about ducting, lighting, and landscaping. One filtered view may just include the basic structure of the building so the unnecessary details can be *filtered out* before this data is considered.

This technique is often applied in relational information visualization; for example the extraction of a minimum spanning tree of the data is a form of filtering [135, 201]. This tree can then be used to show the main "stem" of the relational information. Filtering is typically a pre-processing step that results in certain parts of the model being effectively ignored. Unlike hiding which incorporates the data but doesn't present it until called for.

Distortion

Distortion techniques include intelligent zoom [289], presentation emphasis [209, 257], and fisheye views such as that shown in Figure 2.5 [157, 207, 208, 256, 264, 265]. However, as the models become large, the cognitive load on the user or computational effort required to render such large amounts of graphical information becomes prohibitive. Hy-
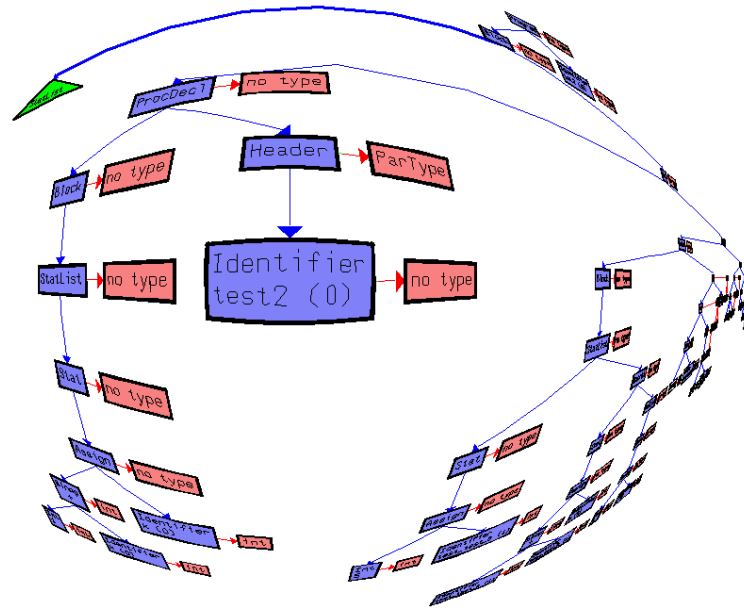
Figure 2.5: Fisheye view of a software call graph

perbolic views can also suffer from this cognitive and computational load [168]. Hybrid hyperbolic viewers for tree exploration, based on filtering coupled with "elision" techniques (see next section) have been developed by Munzner [200].

Hiding and Elision

Often in information visualization the amount of data in the model to be visualized is large. *Information hiding* is the computational process of selectively ignoring or not yet presenting parts of the information. Note, there is a clear distinction between filtering, which is a pre-processing step and hiding, which is part of the visualization process. Hiding techniques that can be directly applied to the drawings of large relational information models include: pan-scan [48, 122, 219] (small window view of a large virtual canvas), zooming [257, 280, 302], cluster based views, and fractal views [160].

Information hiding can be based on the notion of *visual elision*. Unlike filtering or simple hiding, elision methods attempt to "hint" at information that is not fully displayed. Numerous methods employ "glyphs" to convey information about the hidden part of the model. A glyph is a visual symbol, such as a stylised figure, that imparts information nonverbally. Glyphs can be stylised (coded) according to attributes such as colour, size,

orientation, shape, or texture. Clearly, this hiding process reduces the detail in the visualization. Less detail results in less graphical information to draw but this can also make it harder for the user to interpret the information using the more approximate view. The trade off can be simply stated as:

> *For large models, the simpler the visualization, the more approximate the view*
> *of the model.*

Hopefully, each glyph can effectively convey some useful meaning about the part of the model that it represents. In most systems that employ elision, the elided parts can be visualized in detail as they are needed.

If the model has an associated logical grouping then this often suggests the best choice for what parts should be hidden and elided. A "hierarchical-clustering" provides an even more sophisticated model for elision. Some applications use the natural clustering or hierarchical structure of the data to decide what parts are to be hidden. For example, a model of a car can be described in hierarchical terms with a *part-of* relationship. Then, instead of drawing the wheels at the greatest level of detail (tread, colour, letters, nuts, and bolts) a simple cylinder glyph can be used to suggest the wheel and the rest is hidden by elision.

Ware noted that elision in information visualization is analogous to the cognitive process of "chunking" [289]. *Chunking* is the process of cognitively grouping simple concepts into larger ones. Concepts, and hence chunks, are formed based on hypothesis testing. Multiple hypotheses are held in memory and are continually refined and evaluated against one another [55].

In this thesis, we integrate automatic and user directed elision techniques across multiple levels of detail. This integration allows effective visual information exploration and hypothesis testing of large relational data sets; see Section 3.11 for more details on the use and generation of "visual précis".

Aesthetic appeal

The visual abstraction of the London underground (with inset), shown in Figure 2.6, represents thousands of man-hours of continual refinement over the past seventy years. There are
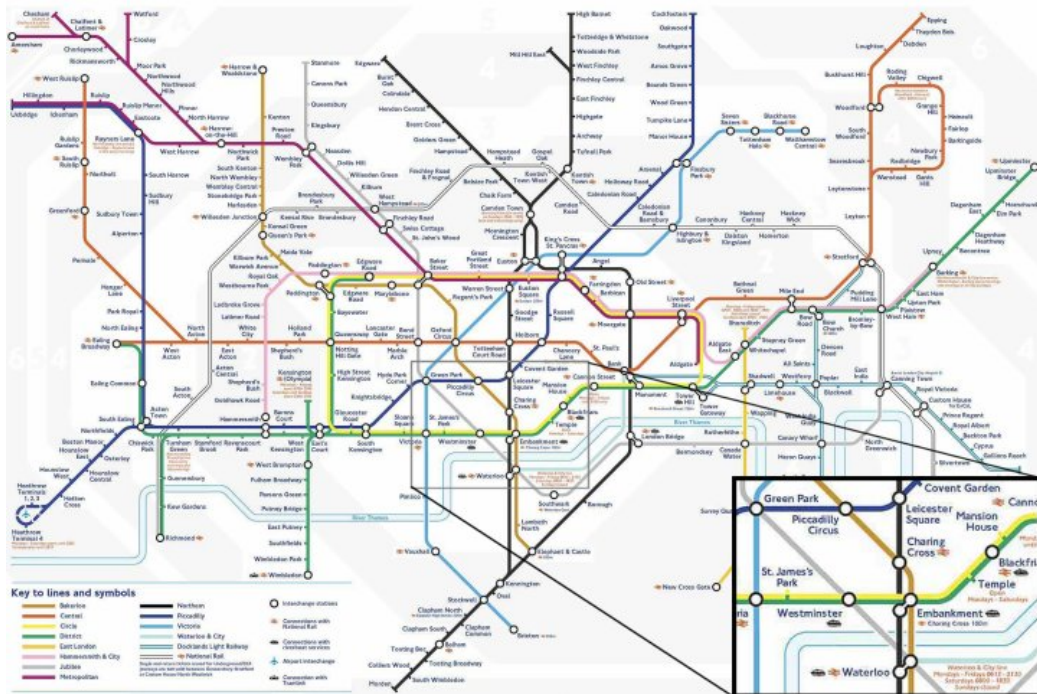
Figure 2.6: Diagram of the UK London underground, with inset

thousands of details from a possible model of the underground system that have been abstracted (filtered) away. The exact details of the topography are removed, although relative orientations are mostly preserved. For example, the *Embankment* station is topographically due north of *Westminster* station, not due east as the zoomed section of the map shown in inset would suggest. The number of tracks on a given line is not shown. The relative size of the stations is not shown; however the fact that a station is an interchange station often indicates that it is a "larger" station.

Clearly, such a map is not very useful for navigating the streets of London or as an aid in finding one station from another at street level. However, that is not the *intended purpose* of this visualization. This fact is *crucial* in determining the usefulness of a given visualization, that is, "What is its intended purpose?". Here the map is intended to help people plan journeys using the subway in order to get from station A to station B. In this regard, it has proved a useful, popular, and enduring visualization that continues to help millions of users daily.

As noted by Eades [70] and Strothotte [266] visualizations such as that in Figure 2.6 are very popular. These maps are readable and useful which typically stems from their

aesthetic appeal rather than geographic correctness. The aesthetic appeal stems from the simplicity and cleanliness of the drawing. Unnecessary detail has been removed to make room for station icons and names. The lines are drawn with uniform thickness, mostly at 0^0, 45^0 or 90^0, corners are rounded, lines do not cross at a station name unless it is an interchange station. Along with this, the colour coding avoids introducing visual noise by placing opponent colours in parallel.

Twelve colours:

- Red, Green, Yellow, Blue, Black, White,

- Cyan, Gray, Orange, Brown and Purple

are generally recommended for coding information. This set consists of the eleven colour names found to be most common in a cross-cultural anthropological study, with the addition of Cyan [289]. Additionally, this set is reasonably far apart in the colour space. All twelve of these colours are used in the London underground map (white is used as a negative space bounded between two thin black lines). The first six colours are easiest to discern and in the underground map these colours are used to code the most important lines, in terms of usage.

Relational information visualization is also guided by the aesthetics of the drawings produced. In this thesis we use our case studies to address the intended use and aesthetic appeal of graph drawing from various domains. Our goal is to demonstrate our FADE paradigm is useful for the large scale drawing of graphs to show their structure. We also aim to the use "visual précis" to show abstractions and improved aesthetic appeal, along with the use of colour to show low level patterns in the data.

User Control

User control for information visualization, simply stated, is putting the user into the information visualization loop. Instead of simply entering the model, running the system and viewing the result (as would happen with a photorealistic rendering, for example), the user is interacting with the visualization system. Users can perform actions which include: selecting parts of the data they want to see in more or less detail, performing searches,
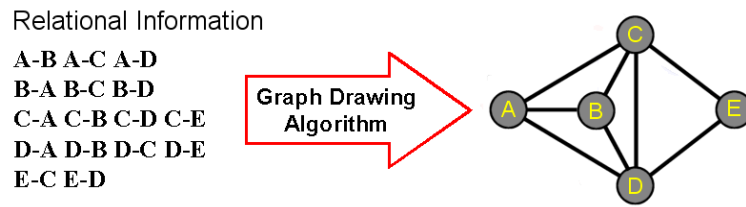
Figure 2.7: Making a picture from abstract relational information

moving parts of the data around, deleting, and editing. As such, information visualization must be construed as a process of iterative observation and exploration of the information available, that is, a form of exploratory data analysis.

2.2 Graph Drawing

Graph drawing is the process of making a picture from relational information. Research in graph drawing has developed considerably since graphics workstations were introduced in the 1980s [32, 60, 69, 97, 121, 147, 241, 248]. The problem is to develop a *graph drawing algorithm*, which assigns a location for every node and a route for every edge. Once the graph drawing algorithm has assigned a geometry, it is then possible to render a picture, that is, a visualization of the graph. This process is illustrated in Figure 2.7. We now define the basic combinatorial concepts that are required for this thesis.

An *undirected graph* $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, consists of a finite nonempty set $\mathcal{N}$ of nodes and a finite (possibly empty) set $\mathcal{E}$ of edges. An edge $e \in \mathcal{E}$ is an unordered pair of nodes. An edge $e = (u, v)$ is said to *join* or *connect* the nodes u and v where $u, v \in \mathcal{N}$. If $e = (u, v)$ is an edge of $\mathcal{G}$, then the nodes u and v are called *adjacent* nodes or *endpoints* of e, while u and e are *incident*, as are v and e. The *order* of $\mathcal{G}$ refers to the cardinality of the node set, whereas the *size* of $\mathcal{G}$ refers to the cardinality of the edge set. Therefore a graph has both an order n and size m. A *simple* graph is one that contains no "loops" or "multiple edges". A *loop* is an edge $e = (u, u)$ that connects u to itself. *Multiple edges* exist when two distinct edges in $\mathcal{E}$ connect the same pair of nodes in $\mathcal{N}$. An example of a graph drawing of a undirected graph is shown in Figure 2.8.

The *degree* of a node n, is the number of edges in $\mathcal{E}$ incident on n. An *isolated node*

has degree 0 and an *end node* has degree 1. For example, a complete binary tree of depth x has 2^x end nodes and for $x \geq 1, 2^x - 2$ nodes of degree 3 (the non-leaf nodes of the tree) and one node of degree 2 (the root node). The *minimum degree* of $\mathcal{G}$ is the minimum degree among the nodes $\mathcal{N}$ of $\mathcal{G}$. The *maximum degree* of $\mathcal{G}$ is similarly defined. In graph drawing, the *degree of a graph* refers to the maximum degree. In a *graph of regular degree* every node has the same degree; examples include the Petersen graph and any "complete graph". A graph is *complete* if every two distinct nodes are adjacent. A more useful measure, for non-regular graphs, is the *average degree of a graph* which is the average number of edges incident on each node of the graph.

Examples of these measures are:

- The knowledge nation graph shown in Figure 2.1 has 23 nodes, 55 edges, degree of 19 and an average degree of 5.

- The class diagram shown in Figure 2.4 has 8 nodes, 11 edges, degree of 4 and an average degree of 3.

- The example graph shown in Figure 2.9 and 2.8 has 16 nodes, 26 edges, degree of 4 and an average degree of 3.

A *sub-graph* $\bar{\mathcal{G}}$ of $\mathcal{G}$ consists of a set of nodes and edges of $\mathcal{G}$. Formally, $\bar{\mathcal{G}} = (\bar{\mathcal{N}}, \bar{\mathcal{E}})$ with $\bar{\mathcal{N}} \subseteq \mathcal{N}$ and $\bar{\mathcal{E}} \subseteq \mathcal{E}$, such that both end points of each edge in $\bar{\mathcal{E}}$ are in $\bar{\mathcal{N}}$. Edges between elements of $\bar{\mathcal{N}}$ are called *internal edges* and all other edges are referred to as *external edges*, as they have at least one end point that is external to the sub-graph $\bar{\mathcal{G}}$. A sub-graph that has the same order as $\mathcal{G}$, is called a *spanning sub-graph* of $\mathcal{G}$. If $\bar{\mathcal{G}}$ is a sub-graph of $\mathcal{G}$, then $\mathcal{G}$ is the *supergraph* of $\bar{\mathcal{G}}$.

Undirected graphs are often used for modeling symmetric relationships between elements of information. As such they provide a simple and extensible way to represent bidirectional relations or relations and their inverse relations. However, in some modeling problems, the symmetric aspect of these graphs does not adequately satisfy the requirements of the problem domain. Instead a different type of graph, namely a "directed graph" is used. A *directed graph* or *digraph* consists of a set of nodes and edges, where each edge

is an *ordered pair* of nodes. Edges in a directed graph are called *arcs* or directed edges. The ordering gives each edge a specific "direction" or "orientation", which is typically represented by an arrow-head in a graph drawing. An example of a graph drawing of a directed graph is shown in Figure 2.18.

Other graph models, such as the hypergraph [149], the clustered graph [86], and the compound graph [268] are discussed in more detail in Section 3.3.

It is important to note the size of the graphs we are concerned with in this thesis. For example, the annual Graph Drawing competition considers large graphs to be in the range of 1000 nodes. For the purposes of this thesis, we are interested in simple undirected graphs that have a large order n greater than $5,000$ and that have a size m within an *order of magnitude* of the order n of $\mathcal{G}$. This means we are not interested in, for example, visualising the complete graph on $5,000$ nodes (which has $12,497,500$ edges). Likewise, the approaches presented in this thesis are not sufficient for visualizing web-graphs, without some pre-processing or additional filtering approaches. Instead we would like to visualize graphs of $5,000$ nodes that contain upto $50,000$ edges.

2.2.1 Drawing Conventions

The combinatorial properties, mentioned above, of a graph $\mathcal{G}$ can be determined before any graph drawing takes place. Graph theoretic properties such as whether the graph is directed or undirected, or whether the graph is planar or not, determine the *class of the graph*. Often this class indicates which particular "graph drawing convention" should be used. A *drawing convention* is a not a formal agreement about how a drawing should be created but it is rather a specific rule that a drawing must follow.

Common two dimensional drawing conventions include:

- Planar drawing, where no two edges cross, see for example Figure 2.4.

- Straight-line drawing, where edges are drawn as straight lines, see for example Figure 2.8.

- Polyline drawing, where edges are drawn as a sequence of connected lines, see for
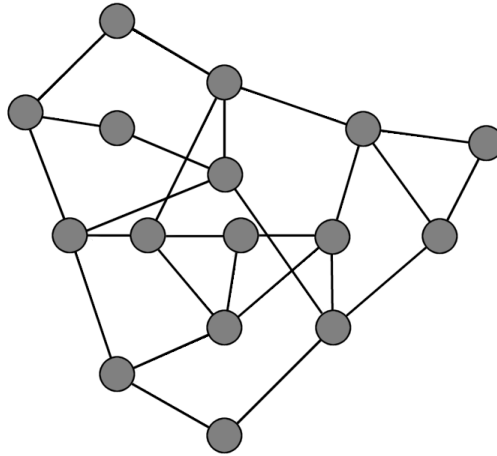
Figure 2.8: A straight-line drawing

example Figure 2.9.

- Orthogonal drawing, where edges are drawn as polylines, consisting of horizontal and vertical segments. Nodes are drawn at integer x,y coordinates of a rectangular grid, see for example Figure 2.19

- Downward drawing, where the edges of an acyclic digraph are drawn as monotonically decreasing arcs in the vertical direction, see for example Figure 2.18.

The most appropriate set of drawing conventions for a graph is often application domain specific and dependent upon the combinatorial properties of the graph. If these conflict then it becomes a matter of determining an appropriate trade off or changing the combinatorial properties of the graph to suit the convention required.

2.2.2   Drawing Aesthetics

The question of readability does not just pertain to text, it also clearly applies to drawings. For drawings, the question is "For a given drawing how easy is it to understand and how effectively and efficiently does that drawing convey information?". Graph drawing algorithms attempt to find a geometrical configuration of nodes and edges which has a high level of readability, according to some set of criteria.
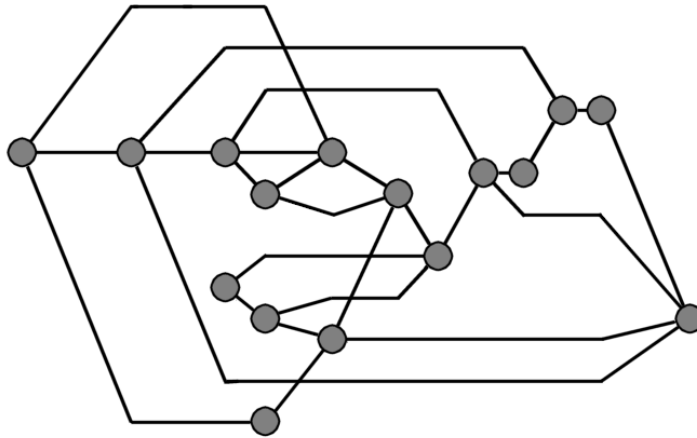
Figure 2.9: A polyline drawing

Regardless of the nature of the graph, or the method used to draw that graph, the primary requirement is that the resultant drawing should be readable. Research has shown that maximizing the readability of a drawing is crucial to conveying the information contained in the underlying graph [52, 114, 229, 230, 231, 232, 273].

Unfortunately, readability is often a very subjective matter and measuring the readability of a specific drawing is open to even more aspects of personal taste and preference. For example, the drawing in Figure 2.1 was heavily scorned throughout the Australian media when it was first published. Could this be the "best" possible drawing of the knowledge nation graph? Without *objective* measures, it is impossible to compare and contrast two drawings or even two layout methods in a scientific manner.

The identification of important features of drawings has been researched since graph drawing algorithms were first developed [60, 229, 230]. The features identified are used to form measures of readability. These features of the drawing are typically called *aesthetic criteria* and have been codified as a set of formal aesthetics. Broadly speaking, measuring a graph drawing in terms of these aesthetic criteria shows whether the drawing has "great beauty" or not.

Although the features of the drawing which impact the formal aesthetic are not independent, a broadly accepted set of base goals (aesthetic criteria) have been identified. Some of the more significant aesthetic measures are informally described below.

Minimizing the number of edge crossings, has been shown to be among the most impor-

tant goals for the creation of an aesthetically pleasing graph drawing [114, 229, 230, 232, 289]. Drawings with a large number of crossings, especially those caused by long edges, are difficult to follow [229]. The drawings in Figure 2.10 to 2.17 show the difference between a drawing with many edge crossings, all the way through to a drawing of the same graph, with none. A random positioning of the nodes of the graph produces a drawing similar to the one displayed in Figure 2.10. This graph drawing has unquestionably poor readability due to the large number of edge crossings. An alternate example is the Korean ISP information visualization show in Figure 1.3, which exhibits many edge crossings. The results of applying an edge crossing measure to a series of layouts can be seen in Figures 2.10- 2.17.

Maximizing edge length uniformity is often used in applications where all edges are of equal significance. One way of representing this is by ensuring that the length of each edge in the drawing is uniform. This aesthetic criteria can be extended to edge set length uniformity, where edges are assigned to categories, each of which has a desired edge length. Often we wish to maximize the uniformity of the length of each edge, or set of edges. The results of applying such a measure, where the ideal edge length is 300 units, to a series of layouts can be seen in Figures 2.10 to 2.17.

Maximizing the distance between non-adjacent nodes ensures that no false relationships, based on proximity are inferred. If nodes that are related are drawn close together then nodes that have no direct relationship should not be close. Cognitively the worst case occurs when geometrically close yet non-adjacent nodes, appear to the user as logically connected. For example, the drawing shown in Figure 2.12 appears to have a dense grouping of nodes of the left of the drawing. This visual grouping occurs because many non-adjacent nodes are being drawn close together. In the context of the other drawings this intuition is clearly false, but alone this grouping might be perceived as a more connected set than it actually is. The results of applying an averaged non-adjacency distance measure to a series of layouts can be seen in Figures 2.10 to 2.17.
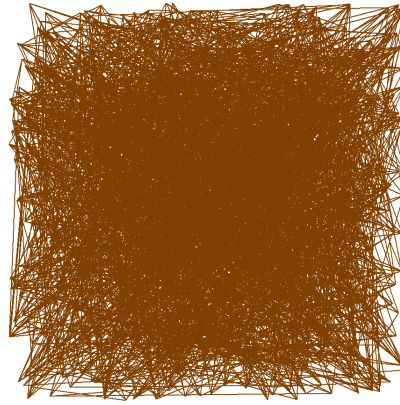
Figure 2.10: Random Drawing of an 800 Node Triangular Mesh which has 109425 edge crossings and average edge length of 6018 and an non-adjacent node distance of 10504
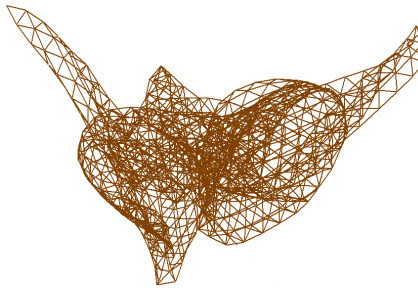
Figure 2.11: Figure 2.10 after 10 iterations of a force-directed layout. Crossings = 5964 Avg. Edge Len= 293 Non Adj. Node Dis = 2497
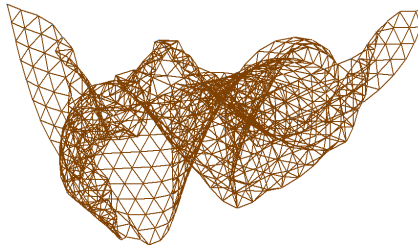
Figure 2.12: Figure 2.10 after 20 iterations of a force-directed layout. Crossings = 5096 Avg. Edge Len= 269 Non Adj. Node Dis = 2605
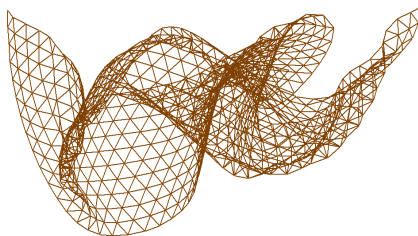
Figure 2.13: Figure 2.10 after 40 iterations of a force-directed layout. Crossings = 2803 Avg. Edge Len= 255 Non Adj. Node Dis = 3292
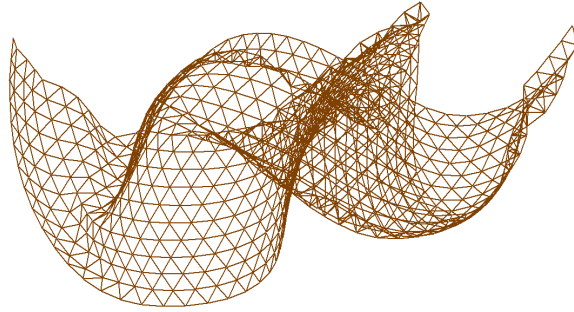
Figure 2.14: Figure 2.10 after 70 iterations of a force-directed layout. Crossings = 2632 Avg. Edge Len= 262 Non Adj. Node Dis = 3682
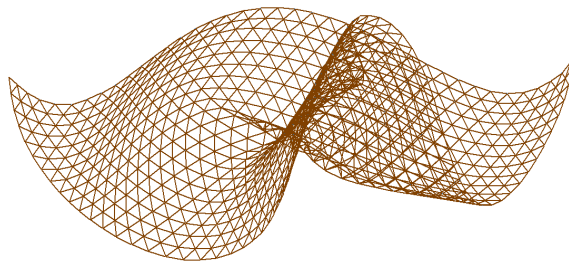
Figure 2.15: Figure 2.10 after 100 iterations of a force-directed layout. Crossings = 1876 Avg. Edge Len= 273 Non Adj. Node Dis = 4099
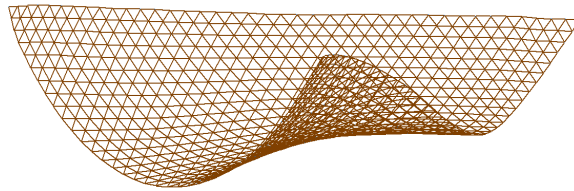
Figure 2.16: Figure 2.10 after 140 iterations of a force-directed layout. Crossings = 588 Avg. Edge Len= 277 Non Adj. Node Dis = 5632
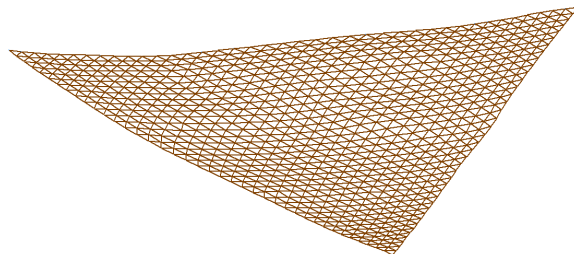
Figure 2.17: 800 node triangular mesh drawn with no crossings and an average edge length of 287 and a non-adjacent node distance of 6444.

Maximizing the symmetries in the drawing aims to display whether the underlying graph has duplicate parts, or near duplicate parts in its structure. The symmetrical graph drawing should reflect a balance in displaying those symmetries. Typically, symmetries provide a formal balance to the layout which can make the process of understanding that graph easier. If repeated or *near repeated* sections of the graph are drawn with rotational or reflexive symmetry, then the understanding of one section often results in a faster comprehension of the other symmetric sections. Informally, the sequence of drawings in Figures 2.10 to 2.17 increase in symmetry. Measures of symmetry are difficult to define and compute; see [229]. The difficulty often arises from the nature of measuring exact versus near symmetry. Users can often perceive symmetries even when the underlying drawing has no strict reflections or rotational symmetries.

Maximizing the angular resolution of the drawing aims to ensure the individual edges drawn, are *clear* and *distinct*. The *angular resolution* of a drawing is the minimum angle formed between a pair of edges that are either crossing or incident on the same node. A drawing that exhibits a low angular resolution, such as Figure 1.3 typically suffers a visual effect called *blobbing* which makes identifying individual edges difficult and hence makes the drawing hard to follow and understand.

Area is a measure of how efficiently a drawing uses available screen space. The area occupied by a drawing is typically measured by the maximum x and y-extent of the node positions, and the z-extent in the case of measuring volume for three dimensional drawings. The goal of this aesthetic is to ensure that area efficient drawings are produced, since screen real estate is a valuable commodity not to be wasted.

Aspect Ratio is a ratio measure of the length of the longest side to the shortest side of a rectangle which encloses all the nodes of the drawing. A drawing with a high aspect ratio may be difficult to effectively visualize as it will not conveniently fit on a computer monitor. It is not uncommon for very large graphs, drawn with the popular Sugiyama style layouts, to suffer from very high aspect ratios. This is primarily due to the graph's topology and the ranking process done within the Sugiyama algorithm.

Other Aesthetics

Given the nature of the graphs that this thesis aims to address, several other important criteria, that any reasonable multi-level layout technique should meet are introduced. An informal description of three such criteria are given below.

Minimize the introduction of edge crossings by abstraction to ensure that higher level views or visual précis are not less aesthetically pleasant than the drawing of the underlying graph. Any abstraction, represents a simplified form of the underlying graph. Depending on the size and combinatorial properties of the graph, this simplification departs from representational accuracy, to a variable range of possible degrees. Using an automatic simplification method it is entirely possible to generate a simplified drawing with edge crossings, where the underlying graph drawing has none. Any simplification method should aim to minimize the introduction of such artifacts of the simplification process.

Glyphs representing groups of nodes should not overlap as this would severely affect the readability of the drawing. An abstraction process groups nodes, which are then drawn using a glyph. As with the underlying drawing, nodes which are drawn close together or overlapping imply relationships where none exist. At higher levels of abstraction, the false positives may result in a more distorted view of the underlying elements and their interrelationships.

Minimize the variance in abstraction aspect ratios to ensure a smooth visual mapping between abstraction levels. The FADE drawing paradigm is based on providing high level simplified views of the underlying graph structure. Users can move between levels or can selectively show various parts at different levels of abstraction. Maintaining a similar aspect ratio between each visual précis of the graph provides a visual landmark to aid the user in moving between levels of detail.

Typically, altering the layout of a graph to improve one aesthetic criteria can negatively impact another, that is, the criteria are not independent and some trade off between the criteria must be determined. This determination is often based on the nature of the application, the type of the graph and the purpose to which the drawings are put. This determination gives rise to a sub-set of aesthetic criteria that are important for a given
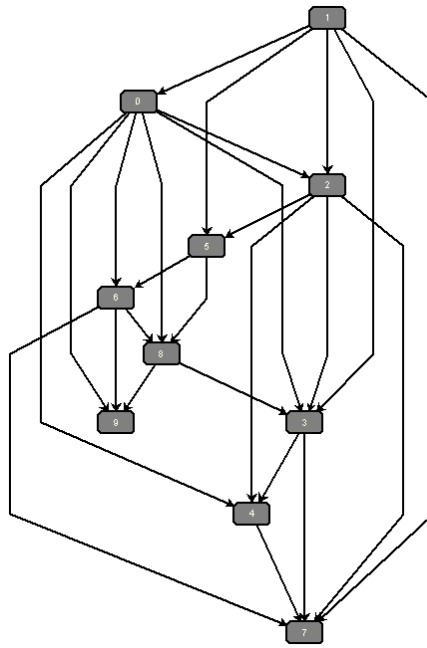
Figure 2.18: A graph drawn with the Sugiyama Algorithm, using GraphWin (AGD)

application domain. This determination is crucial, as attempting to satisfy a large number of criteria simultaneously is at best computationally expensive and at worse futile and infeasible [52, 60, 114, 229, 232, 273].

2.2.3 Graph Drawing Algorithms

Hundreds of graph drawing algorithms, refinements, and specializations have been developed over the past twenty years [60, 76, 272]. Most of these attempt to produce drawings according to some sub-set of the drawing conventions outlined in Section 2.2.1 and many attempt to address the drawing aesthetics discussed in Section 2.2.2. Many toolkits have been developed from these algorithms. A typical toolkit contains three or four paradigms. One such toolkit is AGD [2]. Three pictures from AGD are shown in Figures 2.18, 2.19, and 2.8.

Figure 2.18 shows a *layered* graph drawing from AGD. Layered graph drawings are also know as *hierarchical* or *Sugiyama style* layouts. Typically this drawing convention is used to effectively represent hierarchies of information. Informally, layered graph drawing algorithms consist of three steps. The first step is called layer assignment, where the nodes
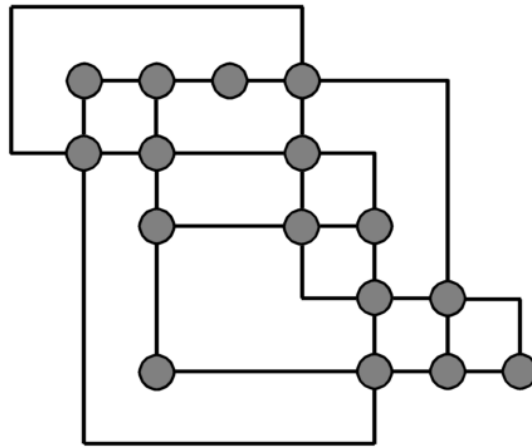
Figure 2.19: An orthogonal drawing

of the graph are partitioned into k layers. This partitioning can be based on the domain knowledge of the graph, such as a set of partial orders or is based on the combinatorial properties of the graph. The second step consists of the nodes in each layer being permuted to reduce the number of edge crossings between layers. The third step alters the horizontal positions of the nodes, and edges are straightened to improve the readability of the drawing. The original layered graph drawing algorithm was first proposed by Sugiyama, Tagawa and Toda [271]. Subsequent work has refined the method along with adding many new techniques for layer assignment and crossing reduction [40, 73, 76, 210]. The layered graph shown in Figure 2.18 consists of 10 nodes and 25 edges drawn on 9 layers.

Figure 2.19 shows an *orthogonal* graph drawing from AGD. The objective of an *orthogonal* graph drawing algorithm, is to produce a drawing with node centres and edge bends on "grid points", with edges drawn as polylines consisting of horizontal and vertical line segments [36, 60, 261].

A variety of orthogonal graph drawing algorithms exist [25, 27, 77, 78, 179, 215, 216, 217, 261], and each tries to satisfy a set of aesthetic criteria; these include minimizing the area (or volume), minimizing the number of edge bends, minimizing the maximum number of bends in a single edge, minimizing crossings (or intersections) and giving a good aspect ratio to the drawing. The notion of orthogonal layout has also been extended to three dimensional orthogonal drawing [17, 26, 51, 77, 78, 179, 218].

The planar orthogonal graph drawing shown in Figure 2.19 has 16 nodes and 24 edges
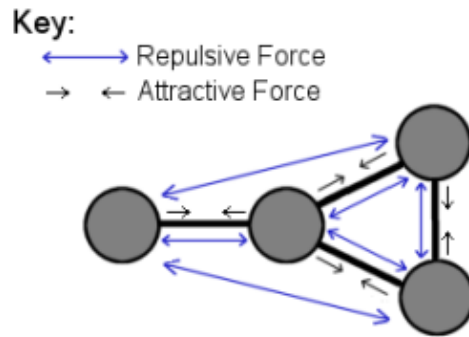
Figure 2.20: Example of edge and nonedge forces in a simple force directed model.

and contains 9 edge bends. This is the same graph as shown in Figure 2.9, which is a layered polyline drawing.

Figure 2.8 shows a *straight line* graph drawing from AGD, produced by a "force directed" layout algorithm. This general class of layout algorithms are popular and in widespread use. Experience with force directed algorithms show that they often produce aesthetically pleasing drawings. We now describe "force directed" algorithms more formally, as it is this class of algorithm we base our FADE drawing and visual abstraction paradigm on.

2.2.4 Force Directed Graph Drawing Algorithms

Force directed graph drawing algorithms, also know as *spring algorithms* or *spring embedders* continue to figure notably among the latest developments in graph drawing [75, 98, 116, 121, 237, 283]. Force directed algorithms tend to emphasize symmetry, maximize edge length uniformity, maximize the distance between non-adjacent nodes, and as by-product tend to minimise the number of edge crossings.

Force directed algorithms view the graph as a *virtual physical system*, where the nodes of the graph are bodies of the system. These bodies have forces acting on or between them. Often the forces are physics-based, and therefore have a natural analogy, such as magnetic repulsion or gravitational attraction. For example in Figure 2.20 the edges can be modeled as gravitational attraction and all nodes have an electrical repulsion between them. It is also possible for the system to simulate unnatural forces acting on the bodies, which have
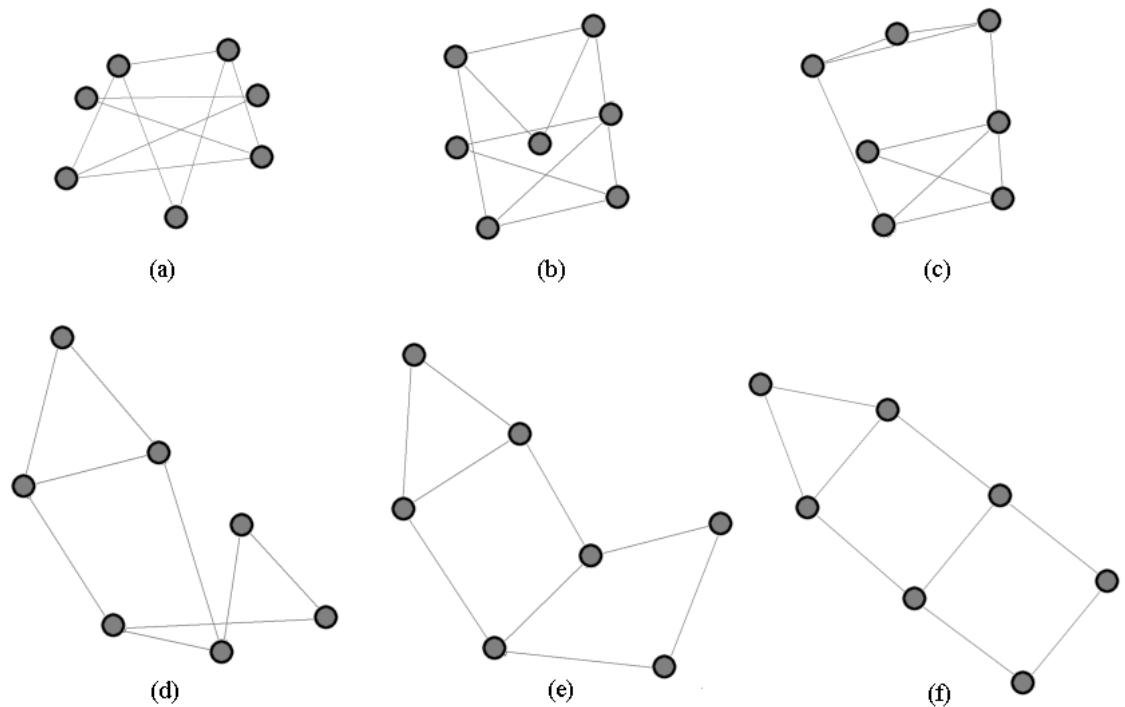
Figure 2.21: Showing the spring model from initial drawing (a) to the final layout(f) with a minimum energy configuration.

no direct physical analogy, for example the use of a logarithmic distance measure rather than Euclidean [69].

Regardless of the exact nature of the forces in the virtual physical system, force directed algorithms aim to compute a locally minimum energy layout of the nodes. This is usually achieved by computing the forces on each node and iterating the system in discrete timesteps. The forces are applied to each node and the positions are updated accordingly. A commplementary approach views the graph as an energy system and algorithms try to minimise the energy in the system [147].

Force-directed algorithms are often used in graph drawing due to their flexibility, ease of implementation, and the aesthetically pleasant drawings they produce. However, classical force directed algorithms are unable to handle larger graphs due the inherent $O(n^2)$ cost at each timestep, where n is the number of bodies in the system. This is a common problem and has prohibited the practical use of force directed algorithms for even moderately sized graphs of a few hundred nodes. The FADE layout paradigm, introduced in this thesis, overcomes this computational limitation to allow large graphs to be drawn, and abstractly

represented.

Background

The simulation of a virtual physical system for object placement pre-dates the development of force directed algorithms for graph drawing [240]. Given that it is NP-hard to draw a graph so that all edge lengths are the same, Eades first proposed a heuristic algorithm for drawing undirected graphs in two dimensions, based on simulating a virtual physical model [69]. This model is now referred to as the "spring model", since each node is modeled as a ring with springs replacing the edges. In this model non-adjacent nodes repel each other according to an inverse square law. Given an initial random layout, the springs and the repulsive forces move the system to a locally minimal energy state, that is, an equilibrium configuration, which is then drawn. Eades noted that in such an equilibrium configuration, all the edges typically have relatively uniform length, and nodes not connected by an edge are drawn far apart. Further, drawings of an equilibrium configuration tend to display the underlying symmetries in the graph [60, 75, 102, 193, 214]. The example shown in Figure 2.21 shows the effect of iteratively applying a simple force directed algorithm to an initial random layout of a graph. The resultant drawing exhibits the three previous aesthetic criteria noted, namely: edge length uniformity, node separation and symmetry.

Since the original spring model, there have been a large number of refinements and specializations to the class of force directed algorithms [18, 23, 69, 75, 97, 102, 115, 116, 118, 148, 237, 270, 269, 280, 283]. The flexibility and simplicity of the original force directed approach has allowed numerous domain specific customizations, algorithmic improvements and applications to be developed [60, 93, 135, 193, 237, 238, 249, 269, 295].

Refinements

One of the most significant refinements to the basic spring model is the force directed algorithm of Kamada and Kawai [147] which attempts to model the graph theoretic distance between two nodes by their geometric distance in the graph drawing. The algorithm models a force between each pair of nodes which is proportional to the difference between their

geometric and graph theoretic distances. At each step in the algorithm, the node with the highest energy is moved to a new location. This algorithm performs a gradient descent based on classical numerical analysis methods to reach a local minimum.

An inherent weakness of many force directed algorithms, such as the spring model and the Kamada and Kawai method, is that they often result in equilibrium configurations at a local minimum such as that shown in frame III of Figure 2.22. Drawings of such local minimum are less aesthetically appealing than a global minimum. To overcome this problem numerous optimizations and heuristics have been proposed.

A class of force directed algorithms that uses "randomness" to avoid ending up at a local minimum was pioneered by Davidson and Harel [57] and independently by Mendonca [189]. This approach draws from a statistical mechanics technique called *simulated annealing*. The goal is to reduce some cost function of the system, while avoiding local minima. The key to such approaches is that a probability function is used to allow *increases* in the cost, in the hope of reaching a lower global cost in the long run. Coupled with this in simulated annealing, is the notion of a falling "temperature". At the beginning of the simulation the temperature is high, so the probability of a cost *increase* is high, such as the move from configuration I to a in Figure 2.22. As the simulation progresses, the temperature is steadily lowered according to some cooling schedule, so the probability of choosing a state with a higher cost tends to zero. The coupling of a more simplified simulated annealing process with a force directed layout, was independently developed in the field of information visualization for document clustering [44]. The Harel simulated annealing approach has been refined and extended in a variety of algorithms and systems. Some of the improvements include: more efficient adaptive cooling schedules [94, 279, 280] and parallel algorithms [39, 54, 162, 197]. The main drawback of simulated annealing for graph drawing, is the fact that it is computationally expensive, so its use in graph drawing, where interactivity is crucial, is often infeasible.

Fruchterman and Reingold's grid based force directed approach [97] uses a more simplistic cooling schedule than that of the simulated annealing class of algorithms. Finally, an alternate approach to find the global minimum is the *conjugate gradient method* as presented in [280].
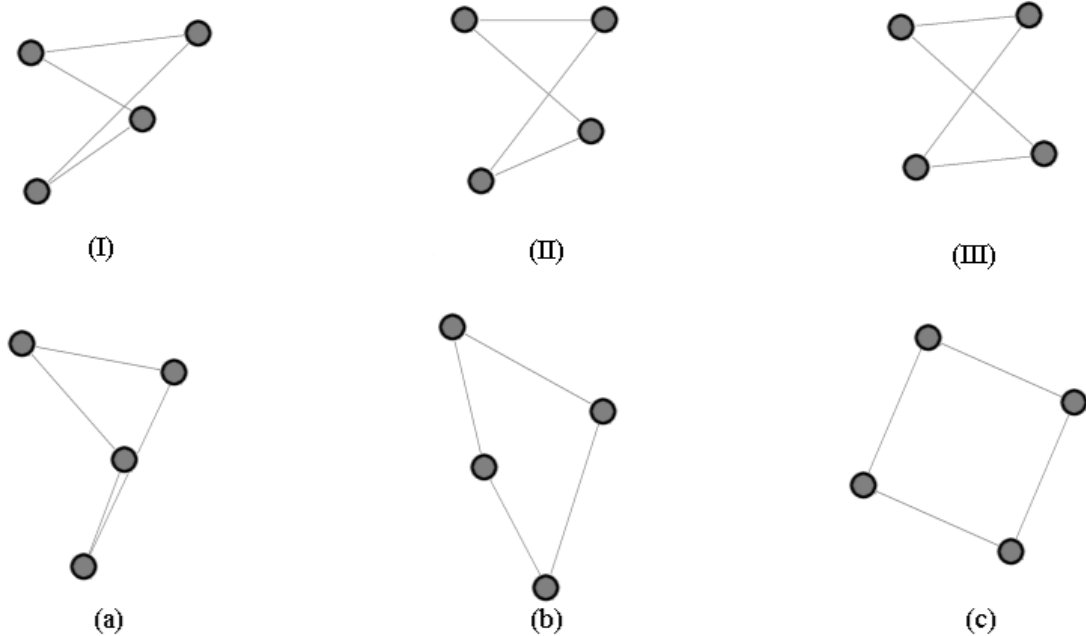
Figure 2.22: Frames I,II,III show the steepest decent approach to energy minimization. Frames I,a,b,c show a simulated annealing approach

Specializations

The range of specializations and customized applications, that use the force directed approach include: *magnetic* or *orthogonal* springs for directional edge alignment [139, 206, 270, 269], forces that maintain topology [23], symmetric graph drawing [75, 184] to display geometric automorphism groups [75, 184], dynamic graph layout [34, 180], three dimensional spring algorithms [98, 167, 197, 214, 239], cluster visualization [44, 136, 236, 288], online graph exploration [135, 136], web site visualization [68, 71, 126, 249, 296], avoiding node overlaps [102, 295], software visualization [97, 119, 238, 243, 262, 263, 264], constraint based layout [118, 148, 288], three dimensional viewpoint location [134, 292], and more recently, large scale graph layout [98, 99, 113, 115, 116, 237, 239, 283, 284].

We now review a selected range of specializations.

- Although most force directed algorithms treat nodes as singularities, in practice this is not the case. In many applications nodes must be represented as labeled entities.

Gansner *et al*. [102] have shown a method to reduce node overlaps and visual clutter. This approach, based on an improved force directed layout, applies a post process node positioning using a Voronoi decomposition of space [91].

- Tunkelang [280] provides a force directed algorithm framework for experimenting with numerical optimizations. This framework supports the interactive drawing of small to medium sized graphs. Tunkelang includes an empirical evaluation of a variant of the conjugate gradient method search strategy to find a global minimum. This framework also supports simple Barnes-Hut and grid variant optimizations to the simple force directed approach.

- Brandes and Wagner [34] have shown a forced directed algorithm using a bayesian paradigm for "dynamic graph layout". *Dynamic graph layout* refers to the layout of graphs that change over time.

- Huang *et al*. [71, 135, 136] provide an online modified force directed algorithm for graph exploration, where the graph is partially unknown. Along with the classical repulsive and attractive forces this algorithm includes many unnatural forces to indicate clusters and the direction of exploration.

- Wills *et al*. [295, 296] demonstrate a post process force directed repelling algorithm to avoid node overlaps, for large to very large graphs. This method is used in conjunction with a computationally inexpensive initial layout method such as a hexagonal or circular layout.

- Frick's GEM algorithm [94] incorporates several heuristic refinements to the basic force directed approach.

Along with various specializations, there have also been a wide variety of graph drawing systems developed which include force directed methods, such as: daVinci [96], GraphViz [164], FADE [237], GEM [95], GraphEd [127], Graph Layout Toolkit [181], Graphlet [129, 128], Jiggle [280], Leda [187, 186], and VCG [255].

However, one common feature of most force directed layout schemes is their inability to scale when drawing larger graphs. The problem stems from the high computational cost

of force directed placement, typically $O(n^2)$ as noted in [23, 60, 97, 118, 136, 135, 173, 237, 249, 280, 284]. Although such algorithms and their specializations are clearly very popular and well researched, it was not until very recently that research addressed their use in drawing large and very large graphs.

2.2.5 Large Scale Graph Drawing

There are several inter-related and inter-dependent issues which must be addressed when dealing with large graph drawings; these include:

- the computational effort involved in determining an aesthetically pleasant drawing

- the effective use of screen space

- the cognitive effort placed on the end user when viewing a large graph drawing

- the ability to present high level visual structures, such as clusters

- and the actual time to render large amounts of two or three dimensional graphical information.

Graph drawing systems that deal with thousands of nodes typically address one or possibly two of these issues, but rarely all five. We now give an overview of existing research that addresses these issues.

- Fruchterman and Reingold's grid based, force directed approach [97] places sets of nodes into cells based on their locations within a simple grid. To reduce the required number of computations, only nodes within cells or neighbouring cells exert forces on one another. This method draws on research from the domain of gravitational physics, specifically methods to address the "N-BODY" problem. The computational efficiency aspects of the FADE paradigm, introduced in this thesis, likewise draw on research into the "N-BODY" problem; see Chapter 4 for more details.

- Graph exploration techniques [71, 135, 136, 137, 200, 201] are based on a *query graph model*. These techniques attempt to address most of the problems associated with large graph drawing, by allowing the user to interactively visualize only a

Figure 2.24: Skeleton view with meta-node glyphs. Reproduced by courtesy of Scott Marshall from I. Herman *et al*. (1998) [123]

Figure 2.23: A *schematic view* of the tree

small portion of the entire graph at any one time. Underlying this graph drawing approach is an interactive graph query system which supports the *query-exploration* cycle. These approaches differ significantly from traditional graph drawing systems, since they rely on visualising only a relatively small part of the graph. OF-DAV [71, 135, 136, 137] augments this approach by adding a trail of focus nodes which indicates where the user has previously explored the graph. The hyperbolic visualization approach of Munzner [200, 201] uses a three dimensional hyperbolic view with the current focus node at the center. This hyperbolic view is based on a Klein model where the projected area is a sphere in three dimensional space.

- An algorithm for the directed acyclic graph drawing that employs information hiding to limit the number of visual element on screen, in terms of extracting a skeleton of the graph, is presented by Herman *et al*. [123]. The skeleton of a graph is the set of nodes and edges that are determined to be significant by a given metric (see Figure 2.24). This skeleton is considered to be the *structural backbone* of the graph and contains signifigant *landmarks* of the graph. This extension of the Reingold and Tilford algorithm [241] provides *schematic views* of the graph based on the skeleton and replaces non-skeletal parts by a colour saturated Trapezoidal glyph set (see Figure 2.23).

- *NicheWorks* is a graph drawing system for very large graphs in the region of 10,000 to 100,000 nodes [295, 296]. The primary focus of this system is to address the computational effort involved in large scale graph drawing. This system, although origi-

nally designed for telephony applications, has been extended to other problem areas such as software modification visualization, web site analysis and the exploration of databases for data patterns. The basic approach of NicheWorks is to initially position the nodes according to computationally inexpensive ($O(n)$) layout methods. These methods include: circular layout, hexagonal layout, and radial layout. This approach works well when the intention is to see the overall structure in the graph; whether it is connected, disconnected, or two-connected. After the initial placement, incremental algorithms such as steepest descent, repulsion, and node swapping can be applied to improve the layout. Here large scale visualization is supported by pan and zoom features.

- Recent advances in the use of graph theoretic partitioning approaches for large graphs, which are suited for use in the force-directed algorithms, are discussed in Section 2.2.7.

The use of clustering, both to reduce the visual complexity and the computational complexity of producing the drawing, is an area of research that shows great promise in addressing the issues associated with large graph drawing. This is discussed next.

2.2.6 Clustering in Graph Drawing

Graph drawing systems which show high level clusters from the underlying graph, rather than the entire graph are becoming increasingly common [50, 66, 72, 73, 86, 177, 235, 237, 244]. These techniques typically concentrate on addressing the effective use of screen space by showing high level structures, that is, clusters, rather than the entire graph at the lowest level of detail.

In the area of graph drawing, Sablowski and Frick [249] first proposed a method which initially attempts a graph theoretic clustering to reduce both the size of the graph to be drawn and the visual complexity of the resultant drawing. Developed independently for document relationship visualization, the Narcissus system [119] showed that a force directed approach could be used to yield clusters that are "visually apparent". As noted by both sets of authors, their respective approaches coupled with a force directed layout, tend

to display the natural clusters of the graph or data set. It is this intuition, that is realized and evaluated in this thesis for large graphs and their visualization.

The *multilevel representation* introduced by Feng [86] shows the entire graph and its different levels of abstraction. Each level of abstraction represents a clustering of the nodes of the graph. Subsequent work has shown how this method can effectively use screen space by presenting high level structures, rather than the entire graph. Follow on work by Duncan *et al.* [50, 66] introduces a cluster-based drawing method which first hierarchically clusters the graph via a balanced aspect ratio (BAR) tree. This allows properties of the clustering such as balance and convex cluster regions to be achieved. This cluster tree can then be drawn on various levels of detail. Unfortunately, when the initial embedding is bad (according to various aesthetic measures) then this method can generate poor clusters with low cohesion and high coupling [50]. This technique is promising for interactive or dynamic graph layout if the goals of this method can be balanced against possibly large changes to the cluster tree. Large changes in the cluster tree, although not significant in a graph theoretic sense, can alter the layout enough to destroy the users' overall mental map.

The multiway ratio cut method of Roxborough and Sen [244], based on a technique from circuit partitioning, aims to simplify the drawing by first identifying natural clusters which can be drawn using a simple circular layout.

Higres, a simplified clustered graph editor, allows for the recursive definition and editing of large clustered graphs [177]. Finally, the clustered graph model has been widely accepted and can be found in commercial graph drawing software such as Graphlet, GLT and D-Abductor [194, 195].

The problems associated with the effective use of screen space and the cognitive effort placed on the end user when viewing a large graph drawing are common throughout the area of information visualization. Collectively, techniques that deal with forming an approximate or distorted view, to enhance the volume or readability of data, are methods for visual abstraction. This is dealt with in more detail in Section 2.1.3.
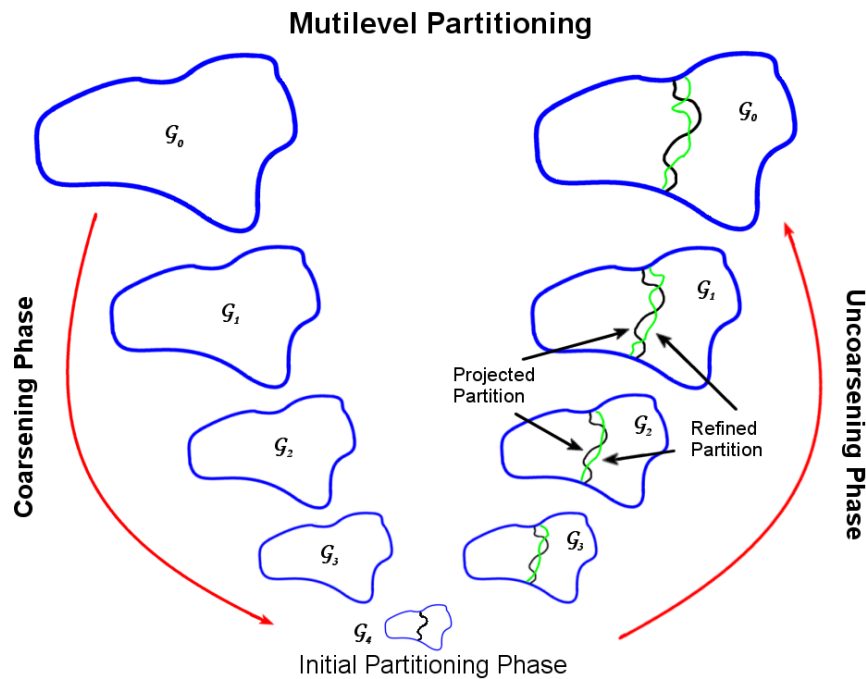
Figure 2.25: Multilevel graph partitioning, Figure adapted from [80, 150]. The initial clustering is projected up as the graph is successively uncoarsened. This is refined at each uncoarsening step to give the refined partitioning.

2.2.7   Multilevel Method Graph Drawing

Parallel to the work in this thesis, Walshaw introduced a multilevel algorithm for force directed graph drawing [283, 284]. Walshaw produced this force directed method motivated by the need to investigate alternately the *micro-* and *macro-*structures from the results of large graph partitioning algorithms. This layout method is based on a multilevel graph partitioning method. In general, *multilevel methods* partition a representative graph, called the "coarsened graph", which is much smaller than the underlying graph, called the "fine graph". The process of creating the smaller graphs, the *coarsened graphs* is referred to as *coarsening the graph*, such as the series of graphs $\mathcal{G}_0$ to $\mathcal{G}_4$ shown in Figure 2.25. The coarsening typically halts when the coarse graph has a few hundred nodes. The *initial partitioning* is then made on the smallest coarse graph, for example $\mathcal{G}_4$ in Figure 2.25.

The layout algorithm proceeds by first applying a force directed layout to the coarsened graph. The graph is then projected and modified back though a series of increasingly "finer", that is, more detailed graphs. At each stage the refined partitioning is used to

minimize the amount of computation required in applying the forces in the system. As in the Fruchterman Reingold approach, non-adjacent regions do not exhibit forces on one another. Finally, the partitioning and associated layout of the original underlying graph is formed, see Figure 2.25.

This method, which is a development of great practical importance for graph drawing, differs from the FADE paradigm in several ways.

- The FADE paradigm doesn't require the computation of a new graph coarsening when the graph changes. As such it supports small graph updates and is suitable for dynamic graph layout and adjustment.

- The FADE paradigm supports multilevel visualization of large graphs. This is based on a geometric decomposition rather than some graph theoretic partitioning of the graph.

- The FADE paradigm offers geometric based visual précis along with measures of their accuracy in representing the underlying graph.

An earlier method by Harel *et al.* [113, 115] used a multilevel approach to speed up a simulated annealing process. This method was successfully demonstrated on generated graphs of up to 3000 nodes. This method, although based on the multilevel method, is not as elegant or scalable as the refined method presented by Walshaw, as it contains a quadratic time component. As noted in [115], a local beautification step, which tends to dominate the computation, was required to remove crossings for graphs of large generated binary trees.

A method by Kobourov *et al.* [98] based on the notion of "set filtration" and "neighbourhood determination" is actually another multilevel variant. A set filtration is a simple coarsening step, in the multilevel sense. The uncoarsening and refinement is achieved by the re-introduction of nodes in a barycenter placement manner. In this method the graphs are positioned in higher dimensions and projected down for "smoother" drawings in two or three dimensions.

Models: Clustering and Visual Abstraction

"Logicians may reason about abstractions. But the great mass of men must have images. The strong tendency of the multitude in all ages and nations to idolatry can be explained on no other principle." - Thomas Macaulay

This chapter describes the models, measures and methods used for the clustering, visual representation, and abstraction of large amounts of relational information. The "hierarchical compound graph model", described in Section 3.3, supports our FADE graph drawing and abstraction paradigm, described in Chapter 4. Further, this clustered graph model is used in various clustering quality measures described in this chapter. The application of our graph drawing and abstraction paradigm, based on this graph model and the testing of our clustering and graph drawing aesthetic measures, is described in the case studies in Chapters 5 and 6.

First, we motivate this chapter by giving an overview of the problems associated with large scale graph drawing in Section 3.1. We describe how the "hierarchical compound graph model" supports all the areas of large scale information visualization that we address in this thesis, from performance to visual abstraction. Section 3.2 presents an introduction to the clustering terminology used in this thesis. Section 3.3 describes models for graph clustering and specifically the "hierarchical compound graph model".

We provide an introduction to the notion of a "quality measure" of a graph clustering in Section 3.4. In Section 3.5, we describe in detail our clustering quality measures for "hierarchical compound graphs". These measures allow different clusterings of the same graph, from different layouts, to be compared and evaluated. Section 3.6 reviews existing

methods for graph clustering.

This thesis is primarily concerned with *geometric* graph clustering, so Section 3.7 reviews existing geometric graph clustering models. Section 3.7.1 discusses how "hierarchical space decomposition" methods form the basis of our hierarchical compound graph generation technique. These decomposition methods are computationally inexpensive, making them suitable for FADE. However, the "hierarchical compound graphs" created may exhibit many artifacts of the creation process. In the case studies, we show how the hierarchical compound quality measures can be used to test the relative inaccuracies in such generated graphs.

In Section 3.11, we describe how the "hierarchical compound graph model" can be used in the creation of a variety of abstract views ("visual précis") of the underlying graph. A *précis* is a sub-graph extracted from a "hierarchical compound graph". A *visual précis* is the drawing of an extracted sub-graph. The types of précis include: "horizons", "event horizons", "cut views", "multi-cut views", and "bell curve views".

3.1   Motivation

Informally, a hierarchical compound graph consists of a set of nodes and edges, clusters (groups of nodes) and "implied edges" (abstractions of edges). This chapter describes models for hierarchical compound graphs, their visual forms, and quality measures. The aim of these models is to address four related issues in large scale information visualization:

- There is a need to significantly improve the performance of classical force directed $O(n^2)$ graph drawing algorithms if they are to be useful for drawing large graphs.

- Drawing algorithms must address large scale *dynamic graph layout* if they are to be useful in real world application domains.

- Screen real estate is a scarce commodity, so the generation by "hierarchical clustering" of higher level views of large graphs are required. These views, called "visual précis", are projections of abstract representations of the underlying graph.

- Once higher level abstract views are introduced by "hierarchical clustering", then hard quantitative measures of such clusterings are needed.

The FADE paradigm includes clustering (to create abstractions of the large graph) and visualization (to create pictures of different levels of abstraction). The method uses a combination of geometric and combinatorial techniques. We map the graph nodes to geometric points, cluster these points using geometric methods, then use the result to produce a "hierarchical compound graph". As noted in other domains, to realize this model we must address two problems [80, 140]:

- Graph nodes have no intrinsic geometric information; we must synthesize the geometry.

- Geometric clustering is suitable for a graph only if the geometric distance between node images reflects the underlying graph theoretic relationships.

The FADE drawing paradigm described in the next chapter addresses both of these problems. In fact, the first problem is addressed by all graph methods, that is, their goal is to synthesize a geometry for the graph so that it can be visualized. The second problem is specifically addressed by force directed graph drawing methods such as those in FADE. Force directed methods attempt to produce a drawing so that related nodes are drawn close together.

The FADE paradigm operates on both geometric and combinatorial models for clustering and visual abstraction. This chapter describes models for both aspects, as well as measures for the quality of both.

3.2   Terminology for Clustering

Clustering is the process of grouping similar objects into naturally associated subclasses. Clustering is one method for partitioning the objects of a set. This process results in a set of "clusters" which somehow describe the underlying objects at a more abstract or approximate level. The process of clustering is typically based on a "similarity measure" which
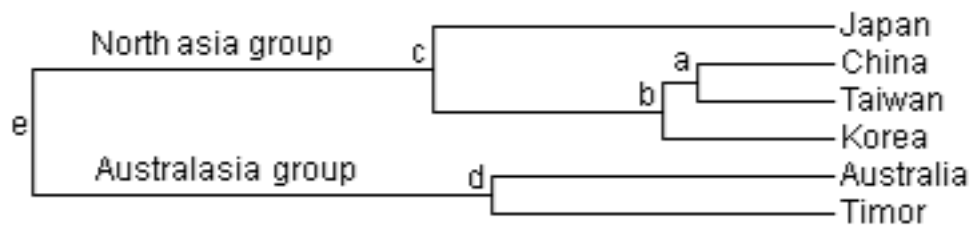
Figure 3.1: Six nations are clustered, based on a geopolitical similarity measure.

allows the objects to be classified into separate natural groupings. A *similarity measure* (or *dissimilarity measure*) quantifies the conceptual distance between two objects, that is, how alike or disalike a pair of objects are. Determining exactly what type of similarity measure to use is typically a domain dependent problem. A *cluster* is then simply a collection of objects that are grouped together because they collectively posses a strong internal similarity based on such a measure. For example, the clustering in Figure 3.1 is based on a similarity measure that uses political considerations, geographic connectivity, and geographic distances to cluster the six countries. The higher level groupings c and d, represent abstractions of the underlying countries. *Point data* consists of a set of attributed items. Clustering of point data attempts to place items with similar attributes into groups. A *clustering method* is a procedure which yields a set of clusters that possess strong internal similarities.

Clustering is a fundamental scientific process in a variety of disciplines and has been studied for hundreds of years. Clustering is used in areas such as: medicine [8, 259, 304], anthropology [65, 178], economics [67, 156], soil analysis [303], data mining [108, 140, 286, 287], reverse engineering [213, 294], program comprehension [107, 183, 281, 301], software maintenance [182], and software engineering in general [82, 103, 104, 130, 138, 204]. Basically any field of endeavor that necessitates the analysis and comprehension of large amounts of data may use clustering.

Formally, a clustering[1] of a set $\mathcal{N}$ is a list $\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_k$ of subsets of $\mathcal{N}$, such that:

---

[1]This thesis concerns itself with *hard clustering* which differs from that of *fuzzy clustering* which assigns each node a degree of membership in several clusters.

$$\mathcal{C}_1 \cup \mathcal{C}_2 \cup \cdots \cup \mathcal{C}_k = \mathcal{N}, \text{and}$$

$$\mathcal{C}_i \cap \mathcal{C}_j = \emptyset \text{ for all } i \neq j.$$

(3.1)

The breadth of application of clustering has spawned a great wealth of supporting concepts and terminology. We now review the main items:

Unsupervised classification is the use of computing technology to aid in the automatic clustering of typically large data sets, without any *a priori* knowledge of their classification [8, 65, 140]. This clustering approach is commonly applied to problems in "exploratory data analysis", where little *a priori* knowledge is known about the structure of the data. Further, supervised methods typically employ a training set of classifiers, the formation of which is a computationally expensive process [156, 192]. The goal of unsupervised classification is to automatically create a set of clusters so that objects within one cluster are more similar to each other than they are to objects in another cluster. Recently, the term clustering has become synonymous with the term unsupervised classification.

Exploratory data analysis is the formation of a hypothesis about the structure and nature of particular data. Typically, exploratory data analysis is used when there is relatively little *a priori* knowledge about the data and any assumptions may cause misleading results. *Confirmatory data analysis* is based on a hypothesis that the data has a particular form or structure, and the analysis is simply performed to confirm the hypothesis. This thesis mainly considers clustering for exploratory data analysis and how this clustering relates to large scale visualization.

3.3 Models for graph clustering

As noted in Chapter 2, graphs are often used in modeling relational information. The scale of the information contained in such graphs has resulted in the need for methods to aggregate or approximate the graph. Such approximation techniques create representations at higher levels of abstraction and are often based on "graph clustering" techniques.

Graph clustering is the process of grouping similar nodes of a graph into a set of sub-

graphs. This process is often based on some similarity measure that is used to cluster the nodes. Unlike point data clustering, these similarity measures typically consider the edges as well as the nodes [79]. For example, the dissimilarity between two nodes may be proportional to the graph theoretic distance between them. Graph clustering is used in various fields such as; VLSI design [30, 41, 159], parallel processing [62, 64, 152, 153], network analysis [87, 173], hyper-media analysis [121, 135, 201, 203, 249, 276, 297], graph drawing [16, 66, 79, 86, 98, 210, 237, 283], and software maintenance and reverse engineering [9, 112, 141, 142, 163, 169, 188, 221, 262, 281].

The reasons for graph clustering vary but the goal of creating a smaller, more abstract and simpler representation of the graph is generally the same. Ideally the clusters formed should be the *natural clusters* of the graph; however as noted by Edachery [79], there is no universally accepted formal definition for a natural cluster, instead only some intuitive understanding, see Section 3.4.

Graph clustering methods take a graph $\mathcal{G}$ and produce a clustering $\mathcal{C}$ for that graph. Such methods can be dichotomized as either "graph theoretic" (discussed in Section 3.6) or "geometric" (discussed in Section 3.9). Broadly speaking a graph clustering method should produce clusters with high "cohesion" and low "coupling", that is, there should be many internal edges[2] and a low "cut size". The *cut size* or *external cost* of a clustering simply measures how many edges are external to all sub-graphs, that is, how many edges cross cluster boundaries. These edges, also referred to as an *edge separator set*, if taken away would result in a set of disconnected sub-graphs. Some degree of uniformity of cluster size is often a desirable property of the results produced by a clustering method. A *uniform graph clustering* is where $|\mathcal{N}_i|$ is "close to" $|\mathcal{N}_j|$ for all $i, j \in \{1, 2, 3...., k\}$. The formal definition of "closeness" here depends on the domain.

Hierarchical clustered graph. The clusters of a graph can be clustered themselves, to form a higher level clustering, and the clusters of clusters can be clustered, and so on. A "hierarchical clustering" is a collection of clusters with the property that any two clusters are either disjoint or nested [81, 140]. We need to model this "hierarchical clustering" in a formal way. A *hierarchical clustered graph* ($\mathcal{HCG}$) consists of an underlying graph $\mathcal{G}$ and

---

[2]This notion can be extended to include weighted graphs.
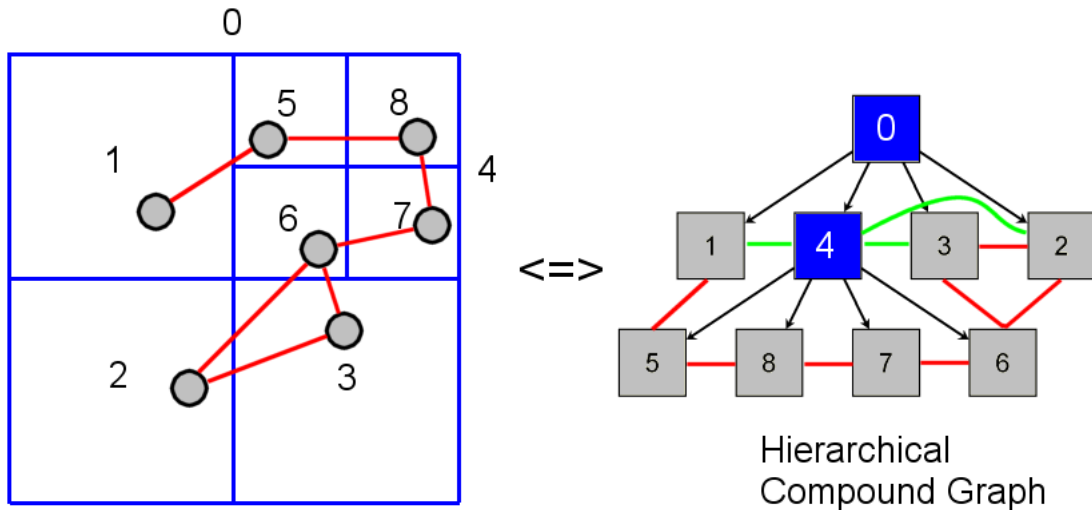
Figure 3.2: Hierarchical compound graph with nodes, clusters, edges in red and implied edges in green

a rooted tree $\mathcal{T}$ such that the leaves of $\mathcal{T}$ are exactly the nodes of $\mathcal{G}$ [86]. Here the tree $\mathcal{T}$ represents an inclusion relationship, so a leaf of the tree represents a node of the graph whereas an internal node of the tree, referred to as a *twig*, represents a set of graph nodes, that is, a cluster. Each twig t of $\mathcal{T}$ represents a cluster $\mathcal{C}_i$ consisting of the nodes of $\mathcal{G}$ that are the leaves of the sub-tree rooted at t.

Implied edges come about because the clusters produced by a graph clustering method have no *inherent* interrelationships (edges) between them but instead can have "implied edges". The intuition behind an implied edge is that two clusters are connected by an implied edge if the nodes that they contain are related. There are several ways to formalise this intuition. Unless otherwise noted, an implied edge is defined as follows. If node x in $\mathcal{C}_i$ has an edge to node y in $\mathcal{C}_j$, then there is an implied edge from $\mathcal{C}_i$ to $\mathcal{C}_j$. Multiple edges can be ignored, or summed to form weighted implied edges. Thresholding can form another type of implied edge, where clusters are connected by an implied edge only if there is at least a certain number of actual node-to-node edges. Clearly other approaches, including the use of domain knowledge about the node attributes and types of interrelationships, are possible.

A **Hierarchical compound graph** consists of a *hierarchical clustered graph* ($\mathcal{HCG}$) and

an implied edge set $\mathcal{I}$, as shown in Figure 3.2.

Précis are extracted sets of clusters, implied edges, nodes and edges from hierarchical compound graphs, used to form abstract representations of the underlying graphs. Suppose that $(\mathcal{G}, \mathcal{T}, \mathcal{I})$ forms a hierarchical compound graph with nodes $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k$ and clusters $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ which are nodes of $\mathcal{T}$ such that every leaf of $\mathcal{T}$ is a descendant of exactly one $\mathcal{C}_i$. The *précis* defined by $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ and $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_i$ is a graph whose node set is $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k)$ and $(\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_i)$ with implied edges between the clusters, real edges between the nodes, and implied edges between clusters and nodes (which are induced by inclusion). This model differs from the clustered graph model of Feng [86] with regard to implied edges, and it is quite close to the "compound graph model" of Sugiyama and Misue [268] .

A *visual précis* is the drawing of a précis. This extended graph model, with an implied edge set, allows a variety of précis to be extracted from the hierarchical compound graph, such as "horizons".

Horizons are the simplest form of précis drawn from the hierarchical compound graph. A *horizon* is a précis where all the *clusters* are equidistant from the root of the inclusion tree $\mathcal{T}$ and all the *nodes* are less than or equal to the same distance from the root as the clusters.

A **hierarchical compound graph method** creates a hierarchical compound graph from an underlying graph. Typically these are recursive clustering methods which produce a nested series of clusters and implied edges. A hierarchical compound graph method may ignore the edges and implied edges and take only the attributes of the nodes and clusters into consideration, in which case it is equivalent to a hierarchical point data clustering method.

3.4 Quality Measures for Graph Clustering

A *graph clustering quality measure* gives a quantitative measure to the "goodness" of a particular clustering. Such a measure is a function that returns a value, that may be used to compare the relative quality of two different clusterings $\mathcal{C}$ and $\mathcal{C}'$ of a graph $\mathcal{G}$.

Ideally, given the optimal clustering $\mathcal{C}_{opt}$ of a graph, it is relatively easy to formulate a quality measure. Such a measure should compare the optimal clustering with a solution $\mathcal{C}$

from any given clustering method. One such measure is the *Minkowski measure* $\mathcal{Q}_m$ which is based on the normalised $\mathcal{L}_2$ distance between representative matrices for the clusterings $\mathcal{C}_{opt}$ and $\mathcal{C}$ [117, 192]. To compute this measure, we first describe each of the clusterings $\mathcal{C}_{opt}$ and $\mathcal{C}$ with an adjacency matrix $\mathcal{M}$, where

$$
\mathcal{M}_{i,j} =
\begin{cases}
0 & \text{if } i \text{ and } j \text{ are in } \textit{different} \text{ clusters, and} \\
1 & \text{if } i \text{ and } j \text{ are in the } \textit{same} \text{ cluster}
\end{cases}
\tag{3.2}
$$

Denote the adjacency matrix for $\mathcal{C}_{opt}$ by $\mathcal{O}$ and the adjacency matrix for $\mathcal{C}$ by $\mathcal{S}$ and the number of elements in the matrix as $\| \mathcal{O} \|$.
. Then,

$$
\mathcal{Q}_m(\mathcal{O}, \mathcal{S}) = \frac{\sqrt{\sum_{i,j}(\mathcal{O}_{ij} - \mathcal{S}_{ij})^2}}{\| \mathcal{O} \|}
\tag{3.3}
$$

Another measure which uses the optimal clustering is the *all-pairs measure* $\mathcal{Q}_{ap}$ [156]. This quality measure determines a value for *overall* difference between the matrices.

$$
\mathcal{Q}_{ap}(\mathcal{O}, \mathcal{S}) = \frac{| \{(i,j) \mid \mathcal{O}_{ij} = \mathcal{S}_{ij} = 1\} | - | \{(i,j) \mid \mathcal{O}_{ij} \neq \mathcal{S}_{ij}\} |}{| \{(i,j) \mid \mathcal{O}_{ij} = 1\} |}
\tag{3.4}
$$

The primary drawback of such measures is the *a-priori* need for $\mathcal{C}_{opt}$, the optimal or "true" clustering. In general there may be a set of optimal clusterings rather than just one. Further, with real application data it is not practical to expect that an optimal clustering is computed before the clustering takes place. However, a test bed of graphs, for which an optimal clustering is known, might be generated to perform a comparative study of the qualities produced by different graph clustering methods. In practice, this *test bed* approach is often used to measure the performance of new clustering methods against existing techniques, such as Koschke's framework [163] described in Chapter 5.

A different approach is to formulate measures based on the desirable properties one might expect a good clustering to have. Such measures then allow the relative quality of two different clusterings to be compared, without having to know the optimal clustering. This is the general approach taken in this thesis, that is, to measure and validate the relative quality of the clustering strategies across different relational information domains. These

quality measures can be dichotomized as "graph theoretic" or "geometric", depending on the nature of the attributes used to determine the measure.

Such measures, although classically applied to a single level of clustering, can be extended or generalized to produce quality measures for hierarchical compound graphs. This thesis presents two heuristic sets of measurements, called the "clustering quality measures for hierarchical compound graphs" and the "clustering quality measures for geometric graph clustering".

Here we specify several measures, graph theoretic and geometric, related to both graph drawing and hierarchical compound graphs. The goal is to then evaluate several of these measures on the drawing, clustering and abstract representation of different large graphs from real world domains. This evaluation is described in the case studies in Chapters 5 and 6. These measures, for hierarchical compound graphs, can either be *graph theoretic*, which are outlined in Section 3.5 or *geometric*, which are outlined in Section 3.8.

Briefly these two sets of measures include:

- IEP Measure: Implied Edge Precision - *graph theoretic*.

- LCA Measure: Lowest Common Ancestor - *graph theoretic*.

- CoCo Measure: Coupling and Cohesion - *graph theoretic*.

- NNS Measure: Node Neighbourhood Similarity - *graph theoretic*.

- SoSE Measure: Sum of Squared Error - *geometric*.

- MV Measure: Minimum Variance - *geometric*.

3.5 Clustering Quality Measures for Hierarchical Compound Graphs

A *clustering quality measure for hierarchical compound graphs* is a function which returns a number for an entire hierarchical compound graph. This measure, as with all absolute or ordinal clustering measures, allows the relative quality of a particular characteristic of
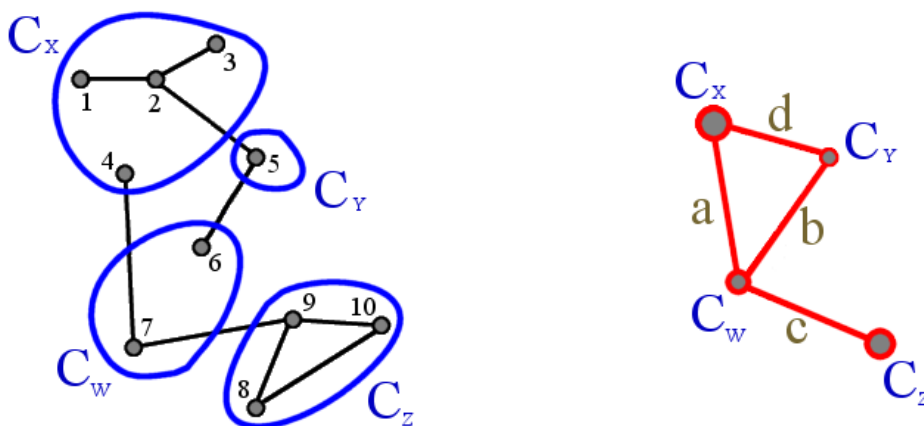
Figure 3.3: Graph with cluster regions and the clustered graph with implied edges *a,b,c,d*

two different clusterings to be compared. Typically, such a measure is used to evaluate the quality of one characteristic of a clustering. In this sense these clustering measures can be thought of as a set of *desirable properties of the clustering*. In graph drawing, different application domains can have different types of graphs, which require different sets of properties of the drawing to be considered to ascertain their relative qualities. Likewise clustering can have different constraints, which require different properties of the resultant clusterings to be considered to ascertain their relative level of quality. In Chapters 5 and 6 we use application domain data and highlight the usefulness of certain measures, due to the combinatorial properties of the graphs.

IEP Measure: Implied Edge Precision

The *implied edge precision measure* $\mathcal{IEP}$ gives a value for how well the implied edge set $\mathcal{I}$ of a hierarchical clustering, expresses the underlying connectivity of the graph $\mathcal{G}$. This is a connectedness measure, and is defined as follows: suppose that you have a hierarchical compound graph defined on $\mathcal{G}$ with a set of clusters $\mathcal{C}$, a rooted tree $\mathcal{T}$ and an implied edge set $\mathcal{I}$. Suppose that C and C' are connected by an implied edge e. The number of possible pairs of nodes that could give rise to the implied edge e is $|C||C'|$. For each pair of nodes $u \in C$ and $v \in C'$, there may be a path in $C \cup C'$ from u to v. Implied edges arise from such paths. In a specific example, for some pair $u \in C$ and $v \in C'$, there may be no path from u to v. For example, nodes $1 \in C_X$ and $6 \in C_W$ in Figure 3.3 are not connected by a

path in $C_X \cup C_W$. However, the clusters C_X and C_W are connected by the implied edge a. We can measure the "precision" of an implied edge e as the ratio of the number of actual paths to possible paths.

This can be formalized as follows: given an implied edge e between two clusters C and C', the *precision* φ of e is

$$\varphi(e) = \frac{1}{|C||C'|} \sum_{u \in C, v \in C'} \xi_{uv} \tag{3.5}$$

where

$$\xi_{uv} = \begin{cases} 1 & \text{if there is a path in } C \cup C' \text{ from } u \text{ to } v \\ 0 & \text{if not,} \end{cases} \tag{3.6}$$

and

$$\mathcal{IEP} = \frac{1}{|\mathcal{I}|} \sum_{e \in \mathcal{I}} \varphi(e) \tag{3.7}$$

Consider the example shown in Figure 3.3.

$$\varphi(a) = 0.125$$
$$\varphi(b) = 0.5$$
$$\varphi(c) = 0.5 \tag{3.8}$$
$$\varphi(d) = 0.75$$
$$\mathcal{IEP} = 0.475$$

There is a difference between the precision of the implied edges c and d, although appearing to have similar connectivity properties, they are actually quite different. The implied edge c suggests that the nodes in cluster C_W are connected to the nodes in cluster C_Z; this connectivity suggestion only applies to half the nodes in C_W. Whereas, the implied edge d suggests a connection which is more precise, since only one node in C_X does not have a path as suggested. Clearly, the clustering shown in Figure 3.3 is not a good clustering in that the implied edgeset does not seem to abstract the underlying connectivity accurately. The implied edge precision of the clusterings in Figures 3.7, 3.8 and 3.9 is 1. The connectedness

suggested by all the implied edges shown is valid as there exist possible paths for all node pairs.

Although these examples represent idealized clusterings, in practice a good clustering method should produce a very precise implied edge set, if it is to be useful for abstract visualization or measurement. The $\mathcal{IEP}$ is measured on the *absolute* scale and can be computed in linear time, using a breadth first search approach.

LCA Measure: Lowest Common Ancestor

The *lowest common ancestor measure $\mathcal{LCA}$* gives a value for how well the implied edge set $\mathcal{I}$ of a hierarchical clustering represents the coupling and cohesion of the clustering. The intuition for this measure is as follows: an edge of the graph that is deep inside a series of clusters but yet causes an implied edge connection high up in the cluster tree, has a larger effect on *coupling* than an edge that results in an implied edge only between clusters deep in the tree. A good clustering should have the vast majority of its implied edges deep in the cluster tree. Here, due to the normalization, the $\mathcal{LCA}$ is measured on the *ordinal* scale. This is a coupling and cohesion measure, and is defined as follows, suppose you have a hierarchical compound graph clustering defined on $\mathcal{G}$. Given an edge e then the *lowest common ancestor* of e is the deepest internal node t of the cluster tree $\mathcal{T}$ which is an ancestor of both end nodes of e. The *pre-leaves* of e are the parents of the end points of e.

The $\mathcal{LCA}$ measure can be formalized as follows: for each edge e of $\mathcal{G}$ denote the depth of the lowest common ancestor of e by $\delta(e)$ and the maximum depth of a preleaf of e by $\lambda(e)$. The $\mathcal{LCA}$ measure is defined by:

$$\mathcal{LCA} = \frac{\sum_{edges} \delta(e)}{\sum_{edges} \lambda(e)} \qquad (3.9)$$

Alternate formulations include using the *average* and *minimum* pre-leaf depth.

For example, consider the hierarchical clustered graph[3] $\mathcal{G}$ shown in Figure 3.4 and the associated cluster tree $\mathcal{T}$ shown in Figure 3.5. If the schema for creating the implied edges

---

[3]Without the implied edges this is not a hierarchical compound graph.
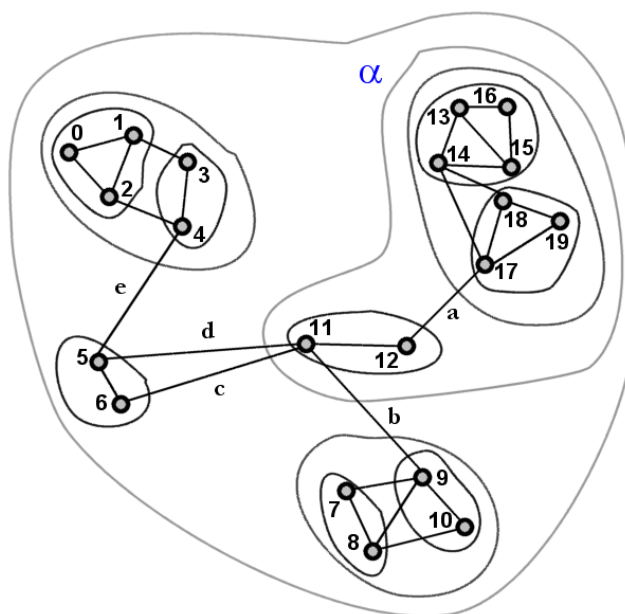
Figure 3.4: Hierarchically clustered graph $\mathcal{G}$, with four of its edges labeled a, b, c, d.
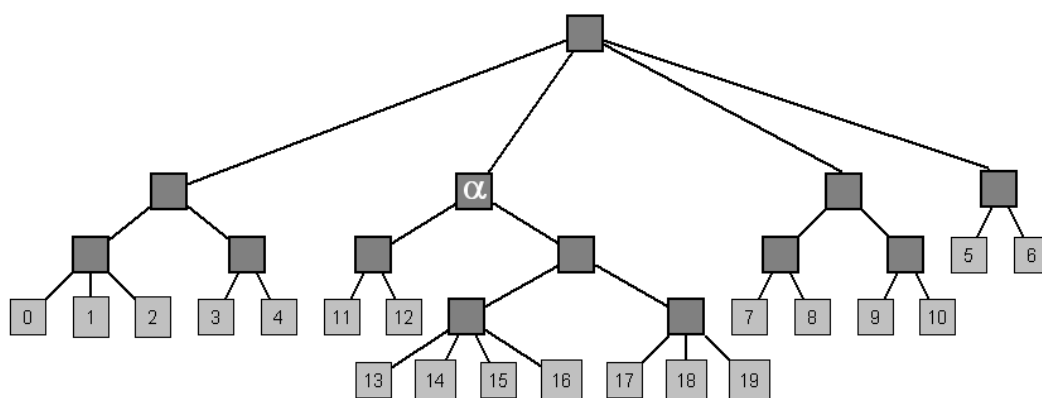
Figure 3.5: The cluster tree $\mathcal{T}$ for the hierarchically clustered graph shown in Figure 3.4

of a compound graph is simply the basic connectivity, then the four graph edges labeled a, b, c, d cause implied edges, between various clusters. The outer cluster, which contains all the nodes and edges of the graph is at level 0 and the cluster containing nodes $17, 18, 19$ is at level 3. Hence the edge a between nodes 12 and 17 first causes an implied edge between a cluster on level 2 and one on level 3. The nodes of a are contained in a level 1 cluster (that is, the internal node α shown in Figure 3.5, is the lowest common ancestor of the edge a.)

$$\delta(a) = 1 \quad \lambda(a) = 3$$
$$\frac{\delta(a)}{\lambda(a)} = 0.33333 \tag{3.10}$$

Clearly, if an edge e causes an implied edge all the way up the cluster tree to the root (that is, the nodes of e have the root node as their lowest common ancestor) then this ratio returns 0 for e since e causes maximal coupling and no real cohesion. The edges b, c, d, e have the root node as their lowest common ancestor, so each contributes 0 to the overall measure.

Of course this must be considered with the fact that sixteen edges of the graph have a value of 1, five of the edges $((1,3),(2,4),(7,9),(8,9),(8,10))$ have a value of 0.5, two edges $((14,18),(14,17))$ have a value of 0.666, and one edge a has a value of 0.333. Giving an overall $\mathcal{LCA}$ measure for this hierarchical graph clustering of:

$$\mathcal{LCA}(\mathcal{G}) = 0.7202 \tag{3.11}$$

This indicates that the cluster tree $\mathcal{I}$ shown in Figure 3.5 has a moderately good cohesion factor. A simple inspection shows that if the cluster with nodes $5, 6$ were included in the cluster with nodes $11, 12$ this measure would improve to 0.8 which is a quality improvement, according to this ordinal measure. Although useful for suggesting places where the clustering can be improved, we are simply interested in this measure to shown how the implied edges can be used to quantify the overall coupling and cohesion in a hierarchical compound graph.
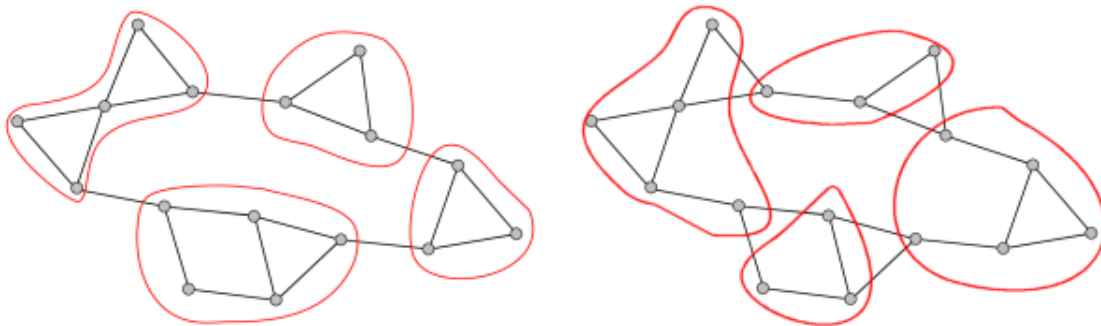
Figure 3.6: Two single level graph clusterings with four clusters each. Where $\kappa = 4, 8$ respectively
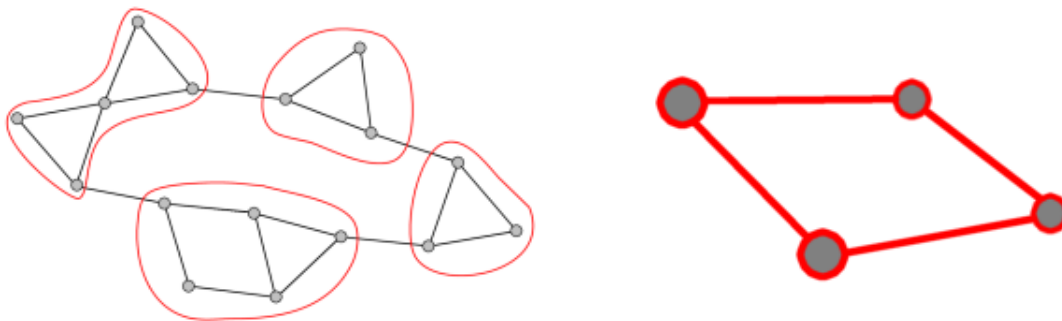
Figure 3.7: First level clustering $\mathcal{L}_1$ of the graph and its induced clustered graph $\mathcal{G}_1$

CoCo Measure: Coupling and Cohesion

One of the most basic absolute scale measures of the quality of a clustering, is the direct coupling and cohesion quality measure CC. This measure compares the number of *inter*-cluster edges $\mathcal{E}_{inter}$, that is, those inside a cluster boundary, with the number of *intra*-cluster edges $\mathcal{E}_{intra}$, that is, those edges crossing a cluster boundary. $\mathcal{E}_{intra}$ is also called the cut size κ of the clustering.

$$CC = \frac{|\mathcal{E}_{inter}| - \kappa}{|\mathcal{E}|} \tag{3.12}$$

For example, from Figure 3.6, then measured on an absolute scale $\kappa = 4$ is $2\frac{1}{3}$ times better than $\kappa = 8$. The maximum value for CC is 1, when all the edges are internal and $\kappa = 0$. The minimum value for CC is -1, when all the edges are external and $\kappa = |\mathcal{E}|$, that is, $|\mathcal{E}_{inter}| = 0$. It is possible to scale the measure so that measurements lie in the range $(0, 1)$, where 0 is a *bad* clustering and 1 is a *good* clustering, according to the notion of coupling and cohesion.
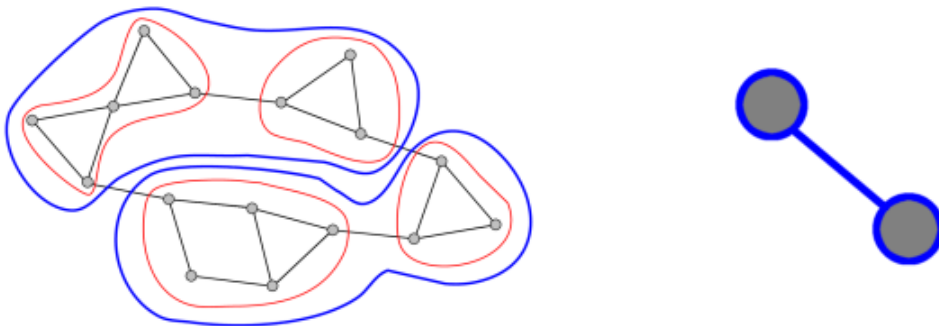
Figure 3.8: Second level clustering $\mathcal{L}_2$ of the graph and its induced clustered graph $\mathcal{G}_2$

The CC is a single level measure which can be extended to be a hierarchical average weighted measure $CoCo$, which measures the coupling and cohesion across the levels of the cluster tree. Given a cluster tree with levels $\mathcal{L}$, an equal weighting ω for each level (which sum to 1) and a single level coupling and cohesion measure CC, then we compute the hierarchical coupling and cohesion as follows.

$$\sum_i \omega_i = 1 \tag{3.13}$$

$$CoCo = \sum_i \omega_i * CC(\mathcal{L}_i) \tag{3.14}$$

$|\mathcal{L}|$ is the height of the cluster tree. For the two level cluster tree ($|\mathcal{L}| = 2$) shown in Figure 3.8, with a weighting of 0.5 for each ω then

$$CoCo = 0.727272$$
$$CC(\mathcal{L}_1) = 0.6363 \tag{3.15}$$
$$CC(\mathcal{L}_2) = 0.8181$$

If the measure is used with no differentiation between levels then $\omega = \frac{1}{|\mathcal{L}|}$.

For the two level cluster tree shown in Figure 3.9, with a weighting of 0.5 for each ω then

$$CoCo = 0.45668 \tag{3.16}$$

Any intuitive comparison is dependent on an absolute scale and the fact that each level is
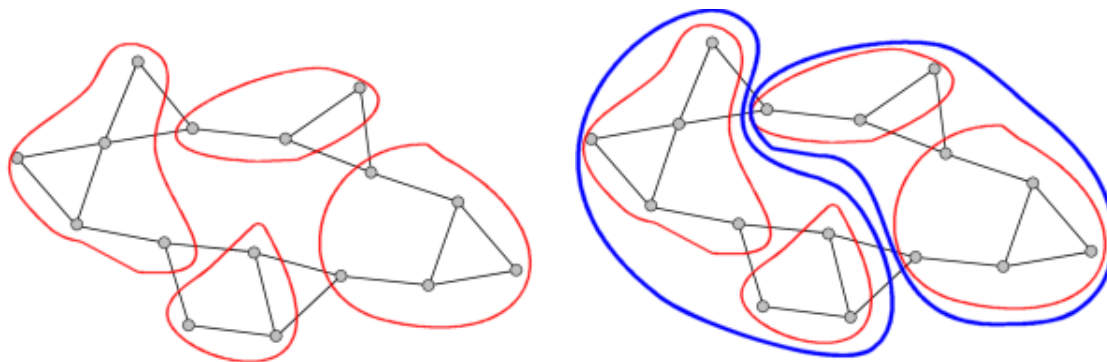
Figure 3.9: Different first and second level clustering $\mathcal{L}_1 \kappa = 8, \mathcal{L}_2 \kappa = 8$ of the graph shown in Figure 3.8

assigned an equal ω weighting. A different way to formulate the ω values is to assign the lowest and highest levels of the tree values of zero and the rest of the levels are assigned a normal distribution of ω about the median level.

NNS Measure: Node Neighbourhood Similarity

The *node neighbourhood similarity measure* $\mathcal{NNS}$ gives a value for how similar the nodes of a given cluster are, in terms of their relative neighbouring nodes. The $CoCo, \mathcal{IEP}$ and $\mathcal{LCA}$ measures use *between-cluster* comparisons whereas this can be thought of as a more traditional *within-cluster* measure of clustering quality. The intuition for this measure is as follows, a given set of nodes in a cluster should be as similar to each other as possible. One way to determine their relative similarities is to consider their adjacent nodes. The nodes of a good cluster should have many of the same neighbours, whereas a poor cluster has few. This is a node similarity measure on the ordinal scale, and is defined formally as follows: given a set of nodes $\mathcal{N}$ and edges $\mathcal{E}$, a cluster $\mathcal{C}$ and $u, v \in \mathcal{C}$, then the neighbourhood of u is $\{v \in \mathcal{N} | (u, v) \in \mathcal{E}\} \cup \{u\}$. Note that the node u is included in its own neighbourhood. The *joint neighbourhood* of u, v is $\eta(u) \cup \eta(v)$. The *shared neighbourhood* of u, v is $\eta(u) \cap \eta(v)$. The neighbourhood similarity $\varpi(u, v)$ of nodes u and v is s/j where s is the size of the shared neighbourhood and j is the size of the joint neighbourhood.

Where the number of clusters is $\mathcal{S}$, then the node neighbourhood similarity measure

$\mathcal{NNS}$ can be expressed as:

$$\mathcal{NNS} = \frac{1}{\mathcal{S}} \sum_i^{\mathcal{S}} \frac{2}{|\mathcal{C}_i| * |\mathcal{C}_i| - 1} \sum_{u,v \in \mathcal{C}_i} \varpi(u,v) \qquad (3.17)$$

To maximize this measure, a cluster should contain a complete sub-graph, where each node is connected to every other node hence all the neighbourhoods are equal.

3.6 Methods for Graph Clustering

A *graph theoretic clustering* is one formed by considering the non "geometric attributes", and the structural information of the graph. These structures include biconnected components, paths, triangles, and circles of cliques [33, 79, 244, 63] . Graph theoretic clustering typically requires some *a priori* decisions to be made before clustering can begin, such as the number of clusters, the degree of uniformity, the maximum cluster size, or the required cut-size [8, 64, 79, 192, 223]. This form of the clustering problem is closely related to the "graph partitioning" problem [30, 80, 83, 145, 154, 274]. Broadly speaking *graph partitioning* is a simpler form of graph clustering that does not consider any attributes of the nodes, and has two main objectives:

- to split the graph into a number of sub-graphs with low coupling

- to achieve a degree of uniformity in sub-graph size.

Graph partitioning is often used as a pre-processing step in the field of parallel computation, when the computational problem can be modeled in terms of a graph [223, 275]. For parallelization, the partitioning effort is generally two-fold, first to identify areas of concurrency in the problem and second to divide the problem so as to "load balance" the effort across a number of processing units. Communication between processing units is expensive in terms of how long it takes, regardless of the architecture, so the first objective is to reduce the overhead in communication between processing units [61]. The second objective aims to distribute the computational effort evenly across the processing units so as to maximize throughput and hence reduce processing time [223].

Examples of graph theoretic clustering, to partition a large computation modeled as a graph include; matrix ordering [153], VLSI design [149], computational mechanics [10], parallel partial differential equation solvers [58, 285], parallel Euler equation solvers [56], physical mapping of DNA [152, 153], and graph drawing [113, 115, 249, 283]. This area of clustering is rich with methods and techniques as described in a book on decompositions of graphs by Bosak [30]. There are two excellent survey papers, although not entirely disjoint, on graph clustering from Elsner [80] and Pothen [223]. Falkner *et al*. [83] give a comparative analysis of the upper bounds for several graph partitioning algorithms, for a variety of graph types, both simulated and domain specific. This approach can be taken to extremes and the usefulness of parallel partitioning solvers, to aid in partitioning problems for parallelization has been questioned by Hendrickson [120].

Point data can also be modeled in terms of a graph, by inducing edges between otherwise independent data elements. One such approach is based on creating a "minimum spanning tree" of the point data, which induces a graph. This graph can then be used with a variety of graph theoretic clustering methods, to cluster the underlying point data set (see [8, 30, 65, 140, 192].

Currently, most graph clustering is a form of basic graph partitioning which does not consider attributed nodes. Some of the most important graph theoretic clustering methods include:

- Node similarity based algorithms

- Kernighan-Lin and variant algorithms

- Graph growing algorithms

- Spectral σ-section algorithms

- Multilevel partitioning algorithms.

Most graph theoretic clustering methods can be specialized with domain specific information, typically this information is used by similarity measures in the clustering process. These domain specific specializations are further discussed in [80, 223] and are outside the scope of this thesis.

We now briefly describe the relevant graph theoretic clustering methods which relate to the formation of hierarchical compound graphs.

3.6.1   Graph Growing Methods

Graph growing is a clustering approach that forms sub-graphs based on expanding node sets. *Graph growing methods* typically select a starting node and add nodes to it until the sub-graph (cluster) is big enough. The addition of nodes is generally achieved by walking the graph to selectively add nodes to the current cluster. These methods, although conceptually quite simple, have very fast running times and can often find good clustering solutions, or at least provide good starting points for other more sophisticated methods. A breadth first graph growing method is incorporated into MeTiS [151] and ParMeTis [154] which are libraries of multilevel methods similar to those described in Section 3.6.4.

Basic *greedy graph growing* methods, add nodes based on information immediately at hand without worrying about the effect of these decisions later. Typically these methods start by randomly selecting a node which is assigned to a cluster. The next adjacent node to get added to this cluster, should be in some sense, the most promising choice based on some "selection function". A *selection function* simply gives a score to a node based on a rule such as, has the lowest degree, maximizes the decrease in cut-size, or minimizes the increase in cut-size. Once the cluster size reaches some *a priori* limit or no suitable node can be added to the current cluster, a new cluster is formed and a starting node is selected. This process continues until all the nodes are in clusters, resulting in a clustering of the nodes of the graph.

A good example of a greedy graph growing clustering method (*GGGC*) is that of Gerlhof *et al.* [104] which is an agglomerative method derived from Kruskal's algorithm for finding a minimum spanning tree [35]. This method initially assigns each node to a single cluster and inserts all the edges into a list sorted by weight. The weight is derived from the sum of the degrees of the end point nodes of the edge. Each edge $e = (u, v)$ is visited in descending order with $u \in \mathcal{C}_1$ and $v \in \mathcal{C}_2$. If the end point nodes of the edge are in separate clusters, that is, $\mathcal{C}_1 \neq \mathcal{C}_2$ and $|\mathcal{C}_1| + |\mathcal{C}_2| \leq \sigma$, where σ is the *optimal cluster size*

decided *a priori*, then the two clusters are joined, otherwise the edge is discarded [105]. This approach can be further improved by a "bounded look-ahead" and a "new-chance" edge list. The *bounded look-ahead* is used to detect situations where it is advantageous to reject the current edge and to consider other edges first. The use of a *new-chance* edge list results in edges which are rejected in the look ahead step, always being re-considered before the clustering finishes. These heuristics give the basic method a limited *foresight* and *memory*. Empirical evaluations, of these heuristics, have shown then can produce high quality clusterings [35].

These methods are more sophisticated than the Farhat-algorithm [84], which selects the starting node in a greedy fashion and then proceeds to grow until the required cluster size is formed. A greedy region growing method is also incorporated into ParMeTiS [154]. A variant of graph growing is the Markov cluster algorithm which simulates flow in a graph by first relating it to a Markov graph. In this method, graph flow is alternately expanded and contracted based on the expectation that flow between dense regions which are sparsely connected evaporate, resulting in a high quality clustering [63].

3.6.2 Kernighan-Lin variant Methods

No discussion of graph theoretic clustering methods would be complete without mentioning the well known Kernighan-Lin (KL) graph partitioning algorithm [159]. This is one of the earliest graph partitioning algorithms and although now over thirty years old, this method is still important and widely used due to the quality of the partitions produced. Given one of the areas this thesis concerns itself with is program comprehension for reverse engineering, it is worth noting that the KL algorithm was pre-dated by Kernighan's seminal work on graph partitioning problems related to program segmentation [158].

The KL algorithm is not directly applicable to clustering large graphs due to its inherently large run time complexity which is approximately $\mathcal{O}(n^2 \log n)$. KL has spawned a large number of variants [41, 87, 145, 267] and is often used by other graph clustering methods, such as the multilevel method described in Section 3.6.4 which are suited to large graphs.

The clustering method developed by Kernighan and Lin [159] was motivated by the difficulties involved in optimizing the placement of circuits onto a set of printed circuit cards. Each card contains a sub-set of circuits and the goal is to minimize the number of connections between circuits on different cards. Clearly this is just a form of clustering as described in Section 3.3, however until the development of the KL algorithm, the optimization of circuit card layout was a manual task requiring thousands of man hours which was a costly, tedious and error prone effort.

The KL algorithm starts with an initial set of clusters and iteratively improves them. In their original paper the initial clustering was random but nowadays the algorithm is used to improve the quality of clusterings found by other computationally inexpensive methods [80, 151, 154, 223], such as those described in Sections 3.6.4 and 3.6.1. The KL algorithm has been extensively surveyed in both research papers and books. We propose only to give a brief description here and refer the reader to [80, 159, 223] for further details.

This description is based on having two initial clusters C_1 and C_2. The KL algorithm makes iterative improvements by swapping sub-sets of equal numbers of nodes between the two clusters to reduce the cut-size. At the beginning of each iteration the *diff-value* for each node is computed. This is the amount the cut-size decreases if the node is moved to the other cluster. The *gain-value* for a pair of nodes $u \in C_1$ and $v \in C_2$ is the amount the cut-size changes if we swap u into C_2 and v into C_1. The *gain-values* determine what particular sub-set of nodes should be swapped. As described, this is basically a greedy algorithm, that attempts to maximize the reduction of edge cut size at each iteration. The power of the KL algorithm comes from the fact that it also contains an inter loop that specifies *some* swaps must be made a number of times, even if these swaps result in negative gains, that is, an increase in edge cut size. The hope is that a few negative gains avoids local minima in the cluster space by reaching a state where more significant edge cut reductions are possible, thereby reducing the overall edge cut size in the long run. At each iteration, the best clustering found so far is recorded and if a series of steps with such negative gains don't improve the cut size, the best clustering can be restored. The acceptance of negative gains means that the KL algorithm isn't a straight forward greedy method. It typically results in good quality clustering albeit at the cost of $\mathcal{O}(n^2 \log n)$ which has since been refined

to $\mathcal{O}(E)$ by Fiduccia and Mattheyses [87]. As noted by many authors [80, 87, 151, 152, 153, 223], the KL algorithm is useful in a local post-processing phase, to further reduce the cut-size of good initial clustering produced by another faster method.

3.6.3   Spectral Bisection Methods

Spectral σ-section algorithms, for example a spectral *bi-section* method, operate on an augmented mathematical representation of the graph, not directly on the graph itself. Spectral σ-section algorithms take the following approach:

   I Consider the graph undirected so it has a symmetric matrix representation.

   II Create the Laplacian $\mathcal{L}(\mathcal{G})$, which is almost the same as the adjacency matrix $\mathcal{A}(\mathcal{G})$, except the diagonal entries are equal to the degrees of the nodes.

   III Compute the second eigenvector i of $\mathcal{L}$. This is used in a relaxed problem which approximates the NP-complete bisection problem.

   IV Partition the nodes into two sub-graphs $\mathcal{P}_1$ and $\mathcal{P}_2$ based on the median eigenvector component.

   V The two sub-graphs need not be of equal size. Then, $|\mathcal{P}_1| = m$ and $|\mathcal{P}_2| = n - m$. This is called the *m-partition*, where the mth largest or smallest component of the second eigenvector is used to determine the partitions [45].

This method can be extended to a *recursive* spectral bisection (RSB) which is a heuristic technique for finding a recursive minimum cut graph bisection [45]. For graph clustering and partitioning considerable study has gone into eigenvalues and eigenvectors [6, 47, 53, 166, 196, 224, 225]. Notable surveys on this area are by Mohar [196] and Merris [190, 191].

3.6.4   Multilevel Methods

A *multilevel method* clusters (partitions) a representative graph, called the "coarsened graph", which is much smaller than the underlying graph, called the "fine graph". The

clustering is then projected and modified back though a series of increasingly "finer", that is, more detailed graphs, until a clustering of the original underlying graph is formed, see Figure 2.25. The process of creating the smaller graphs, the *coarsened graphs* is referred to as *coarsening the graph*, such as the series of graphs $\mathcal{G}_0$ to $\mathcal{G}_4$ shown in Figure 2.25. The coarsening typically halts when the coarse graph has a few hundred nodes. The *initial partitioning* is then formed from the smallest coarse graph, for example the partitioning of $\mathcal{G}_4$ in Figure 2.25.

To cluster the coarsest graph, a variety of clustering methods can be used; these methods include: spectral, Kernighan-Lin, breadth first region growing and greedy growing methods. The initial clustering of $\mathcal{G}_4$ is used to generate a clustering of $\mathcal{G}_3$ by reversing the coarsening, that is uncoarsening the graph. The approximate clustering of $\mathcal{G}_3$ is then refined, for example by moving or swapping nodes to reduce the cut size. This process is then repeated for each uncoarsening step, until a refined clustering of the original (finest graph) $\mathcal{G}_0$ is produced.

Although multilevel methods are relatively new, they are popular due to their simplicity, computational efficiency and ease of implementation. These methods have been successfully parallized to handle much larger graph partitioning problems. Different methods implement a range of coarsening steps such as neighbourhood growing, maximal matching [62], heaviest edge matching and random matching [150, 152, 153].

Recall that multilevel partitioning schemes have also recently been used in graph drawing to provide significant performance improvements to some classical layout methods [98, 116, 283], as we discussed in Section 2.2.7.

3.7 Models for Geometric Graph Clustering

Typically data analysis is concerned with the exploration of sets of point data. In such data sets each element has *multiple* attributes. Viewed geometrically these attributes allow the points to be represented in *n*-dimensional space. Thus a similarity measure for point data can be based on *Minkowski distances*; for example the Euclidean distance between points is often used to determine their relative similarities. Points that are closer in Euclidean

space are somehow more similar than those that are far apart. Generally, this similarity measure is well suited to aid in the exploration of patterns which are based on the distance relationships within a set of points. This approach implies that the n-dimensional space is isotropic which means such measures are not invariant to linear transformations.

A more general approach is to use a geometric *similarity function* $\mathcal{S}(x, x')$. Where the function $\mathcal{S}$ is symmetric and returns a large value if x and x' are similar. However this level of generality comes at a high cost; for example, formulating a robust similarity measure for abstract categorical data or data of high dimension is often impractical [178].

A graph with an assigned geometry is "spatial data" consisting of points, lines, polylines, and polygons in two dimensions and surfaces or volumes in three. The efficient and scalable representation of spatial data is important in a variety of applications in software visualization [12, 37, 142, 161, 211], reverse engineering [188, 263], graph drawing [219, 98, 237, 296], computer graphics [13, 22, 172, 252, 253], computer animation [13, 59, 266], image processing [253, 266], and computational geometry [20, 21, 185, 226, 252, 253].

Many applications in these areas rely on the ability to manipulate spatial data. As the storage capacity and speed of available computing technology increased dramatically over the past 30 years, applications that use larger models of spatial data were called for. Representing such large complex spatial data models using naive data representation techniques proved unsuitable. Similar to the difference between searching a file system based on flat files and one based on a database, often the primary weakness of a representation of a model is that it does not scale well to handle larger amounts of data.

One such model for spatial data representation is based on the recursive decomposition of space to form a hierarchy of regions. These divide and conquer methods are referred to as *hierarchical space data structures*. Examples of such hierarchical data structures include quadtrees [88, 299], kd-trees, bintrees, PR-trees [252, 253, 254], and binary space partitions [21, 66]. Hierarchical data structures allow an application to focus on subsets of data leading to a scalable representation with improved execution.

3.7.1 Hierarchical Space Decomposition

Space decomposition is the partitioning of geometric space into smaller regions. One such method for partitioning planar space is referred to as polygonal tiling [253, 254], or tessellation [21]. Each partition can be represented in terms of equations of lines, which define the polygonal perimeter of the partition. Each partition of space is referred to as an *atomic tile*, since it is the finest granularity representation of a *unit of space*. A *regular tiling* has polygonal tiles with edges of equal length and equal interior angles, see for example Figure 3.10.

A *hierarchical space decomposition* is a recursive partitioning of space, where the tiling used is an infinitely repetitive pattern and is infinitely decomposable into an increasingly finer tiling. These two constraints satisfy the earlier definition of a hierarchical clustering. If the space contains a point set, then a hierarchical space decomposition induces a hierarchical clustering of the point set. Further, if the geometric space contains the drawing of a graph, then the hierarchical space decomposition induces a hierarchical clustering of nodes of the graph and hence a graph theoretic clustering, this is discussed in more detail in Section 3.9 and Chapter 4.

A further constraint on the tiling is that of "similarity". A *similar tiling* exists when a tile at any level k of the hierarchical space decomposition, has the same shape as the root tile. For example you cannot have a similar tiling based on hexagons as in Figure 3.11, because tiles must either overlap or miss parts of space, thereby invalidating the partitioning constraint. Clearly other types of space decomposition are possible, such as a recursive Voronoi decomposition of space. Such a partitioning, is polygonal in nature but typically is neither regular nor similar. Voronoi diagrams take time $\mathcal{O}(n \log n)$ per level of the hierarchy to compute [91].

A very general class of hierarchical space decompositions are based on "quadtrees". In general a *quadtree* is a data structure used to recursively group pixels and a *octtree* is one that recursively groups *voxels*. Typically this grouping takes place in a top-down fashion rather than a bottom up agglomerative manner. To create a quadtree, a partitioning method recursively divides space in a regular, similar and polygonal manner.
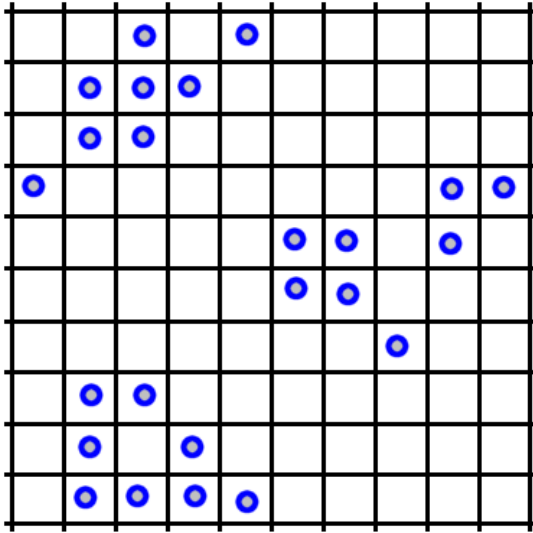
Figure 3.10: Regular polygonal tiling of a point set in the plane.
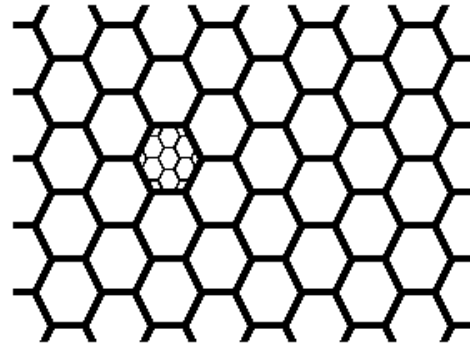
Figure 3.11: Hexagonal non-regular tiling.

We are concerned with defining quadtrees that are used for the storage of large geometrically attributed graphs, not images. Each node (vertex) has a location, and in its simplest form each edge route can be derived from its associated node's locations. In this thesis we consider a quadtree to be is a rooted tree where every internal cell, referred to as a *twig*, has a maximum of four daughter cells. Every internal cell represents a square of two dimensional space (a *tile*), with the root cell representing the entire space, as shown in Figure 3.12. Each daughter cell represents at most $\frac{1}{4}$ of the space of its parent, that is, a quadrant of space, hence the name quadtree. This differs slightly from the classical definition, where if a cell is classed as internal then it has exactly four daughter cells each of which is classed as empty, full or partially full. The regions of space which have been greyed-out in Figure 3.12, are not daughters of their parent cells, instead they simply do not exist in our quadtree structure. This definition is very similar to the definition of a *PR Quadtree* [254], except here all the nodes are initially known.

The classical definition of a quadtree stems from the image processing field, which uses quadtrees to aggregate data having identical or similar values, rather than point sets. In general the construction of a quadtree can be differentiated on the following bases:

- **Type of data:** Is the data in point format or is information based on regions or
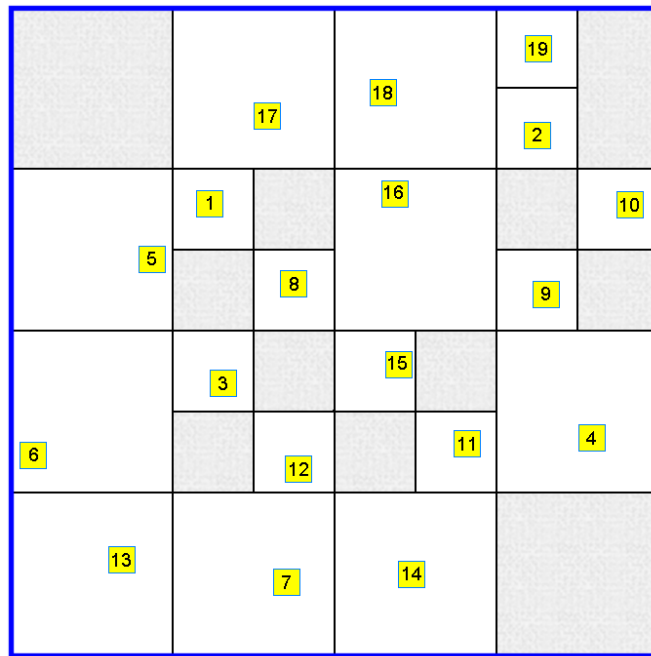
Figure 3.12: Quadtree decomposition of a point set in the plane.

volumes?

- **Decomposition rules:** Are the daughter cells regular polygons and is the decomposition a similar tiling, that is, are the polygons the same shape on each level?

- **Resolution:** How many times is the decomposition process performed? Is it based on an *a priori* selection or based on the input data?

We use quadtrees to form hierarchical clusterings of space which in turn are used to form hierarchical compound graphs. Therefore, our quadtree construction is based on:

- **Type of data:** A graph drawing where nodes have unique x- and y-point locations. (centroids of the labeled square as shown in Figure 3.12)

- **Decomposition rules:** A simple $\frac{1}{4}$ split of the parent square cell into smaller square cells that are regular-polygonal, thus inducing a similar tiling.

- **Resolution:** Nodes are spilt until each leaf cell contains only one node. This is a *maximal decomposition*.

The daughters of the root cell are labeled NE, NW, SW and SE to indicate which quadrant they correspond to; SW stands for the south-west quadrant (or the bottom left quadrant in simpler terms). The recursive splitting continues as long as there is more than one node (point location) in a square, so that each leaf cell contains at most one node (point location) of the graph. The split is based on the median x and y lines.

Suppose that $\mathcal{N}$ is a set of points. A quadtree for $\mathcal{N}$ consists of a rooted tree $\mathcal{T}$, a set $\mathcal{N}_u \subseteq \mathcal{N}$ and a region $\mathcal{C}_u$ for all nodes u of $\mathcal{T}$, such that:

- If u is the root of $\mathcal{T}$ then $\mathcal{N}_u = \mathcal{N}$ and $\mathcal{C}_u$ is defined as follows. Suppose that the minimum/maximum extent for the x/y values of the points in $\mathcal{N}$ are x_{min}, x_{max}, y_{min}, y_{max}. Then $\mathcal{C}_u$ is the minimum enclosing square centred at:

$$\frac{x_{min} + x_{max}}{2}, \frac{y_{min} + y_{max}}{2} \qquad (3.18)$$

- If u is a node of $\mathcal{T}$ such that $|\mathcal{N}_u| = 1$ then u is a leaf.

- If u is a node of $\mathcal{T}$ with $|\mathcal{N}_u| > 1$ then consider the quadrants $\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \mathcal{Q}_4$ of $\mathcal{C}_u$. If $\mathcal{N}_u \bigcap \mathcal{Q}_i \neq \emptyset$, then u has a daughter d with $\mathcal{C}_d = \mathcal{Q}_i$ and $\mathcal{N}_d = \mathcal{N}_u \bigcap \mathcal{Q}_i$

1 Lemma (Depth of a Quadtree)

The depth of a quadtree for a set of nodes $\mathcal{N}$ in the plane is at most $log(s/c) + \frac{3}{2}$, where c is the smallest distance between any two points in $\mathcal{N}$ and s is the width of the root cell.

2 Lemma (A quadtree of depth d storing a set of $\mathcal{N}$ nodes has $O(2\mathcal{N})$ nodes)

Each node leaf in the quadtree contains precisely one node. Further, twigs have at least two daughters. Thus the total number of nodes is at most $2\mathcal{N}$.

Classical quadtrees were developed by Finkel and Bentley in 1974 [88]. Although over ten years old, the surveys and books by Samet [252, 253, 254] still provide the most comprehensive review and analysis of tree based data structures and their application. A quadtree is the simplest form of a data structure based on a recursive decomposition of space. Due to its simplicity and ease of implementation it is used in a wide variety of the application domains that deal with spatial data. Quadtrees were motivated by the need to reduce the storage space required for two dimensional images and three dimensional
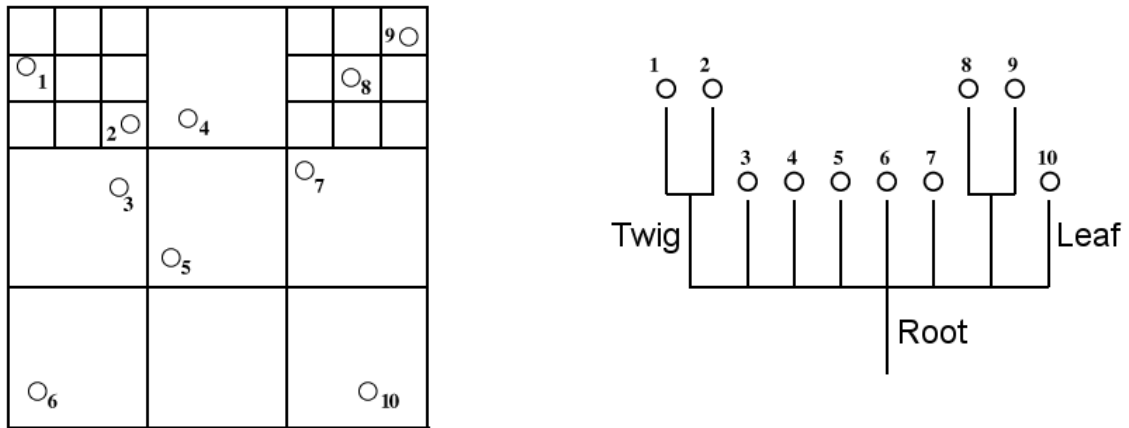
Figure 3.13: Nonotree space decomposition and data structure

models. However, as we show in Chapter 4, the reduction in computation and visual complexity are often more dramatic and of greater importance than storage space reduction for information visualization.

One of the major contributions of this thesis is the new application of such hierarchical data structures to the creation of hierarchical compound graphs and their use in the layout, abstract representation, and measurement of large graphs. This thesis also introduces several variants of the quad and octtrees, namely the general family of orthogonal n^2 or n^3 decompositions. For example, the *nonotree* shown in Figure 3.13 can be used in the abstract representation and drawing of large graphs.

3.8   Quality Measures for Geometric Graph Clustering

A *geometric graph clustering quality measure* gives a quantitative measure to the "goodness" of a particular clustering, in terms of it geometric attributes. Such a measure is a function that returns a value that can be used to compare the relative quality of two different geometric clusterings C_i, C_j of the drawing of a graph G. Graph theoretic clustering considers the non-geometric attributes of a graph. However, if the graph can be viewed geometrically, and each one of its geometric attributes can be mapped to an individual dimension, in n-dimensional space, then each node of the graph can be considered a point in n-dimensional space.

Classical geometric clustering is based on the statistical analysis of the distribution of such n-dimensional point sets. Based on this analysis, "clouds" or clusters of points are identified and grouped. However, determining exactly which type of statistic to measure *a-priori* is a difficult and highly domain dependent issue. Assuming the data has a simple normal distribution, may lead to the discovery of a good natural clustering of the data set but it may also lead to gross inaccuracies in the interpretation of that data [65]. The difficulty in this clustering approach is knowing the structure of the data before clustering begins.

Regardless of the clustering scheme used, quality measures based on measuring different geometric aspects of the clusters can be formulated. Typically, such measures are used to evaluate the quality of one geometric characteristic of a clustering. These measures can be extended to apply to hierarchical compound graphs. Examples of such measures include:

- SoSE Measure: Sum of squared error.

- MV Measure: Minimum variance.

A geometric cluster is typically described by a representative point in the cluster. The point is typically some common centre point, measured according to some distance function. The common point is typically the centre of mass of the point set but it may also be the most representative point. The distance measure used can be a simple Euclidean measure or a Manhatten distance.

SoSE Measure: Sum of Squared Error

The *sum-of-squared-error* measure is one of the most basic geometric measures used in determining the clusters of point data [140]. It can also be used to measure the relative quality of two different clusterings. This quality measure is based on the following intuition, a set of point data in a cluster should be uniformly close to the representative point used to describe this cluster in an abstract manner.

This is a geometric cohesiveness measure, and is defined as follows. Suppose you have a hierarchical graph clustering defined by $\mathcal{G}$, a set of clusters $\mathcal{C}$ and a rooted tree $\mathcal{T}$. Then

$|\mathcal{C}_i|$ is the number of graph nodes in a given cluster $\mathcal{C}_i$. Cluster $\mathcal{C}_i$ has a mean $\mathcal{M}_i$ for its node set. Summing the squared differences between the mean for each cluster and the points in the cluster, gives a measure for the overall level of cohesion in that cluster. One would expect a random grouping of nodes to result in a very poor quality measure, whereas dense localized groups give much better results.

This can be formalized as follows, given a cluster $\mathcal{C}_i$ the mean $\mathcal{M}_i$ is:

$$\mathcal{M}_i = \frac{1}{|\mathcal{C}_i|} \sum_{u \in \mathcal{C}_i} u \tag{3.19}$$

and the sum of squared errors is:

$$\mathcal{SOSE} = \sum_{i=1}^{|\mathcal{C}|} \sum_{u \in \mathcal{C}_i} \| u - \mathcal{M}_i \|^2 \tag{3.20}$$

This measure can be applied to the entire tree $\mathcal{T}$, either by considering only the actual node locations a cluster contains, regardless of the level in the cluster tree, or the clusters of clusters may be considered the elements u of the clustering. This is a measure on the ordinal scale, for use in cross comparison, it must be normalised since it simply represents the total squared error involved in representing this set of nodes by increasingly more abstract pseudonodes.

MV Measure: Minimum Variance

The $\mathcal{SOSE}$ measure is often modified to encompass a more general class of similarity measure $\bar{S}$. The *minimum variance* measure compares all the inter-node distances between points in a cluster. As with the $\mathcal{SOSE}$ it can also be used to measure the relative quality of two different clusterings. This quality measure is based on the following intuition, a set of point data in a cluster should be uniformly close to each other and not just a single representative, as in the $\mathcal{SOSE}$ measure.

This is a geometric cohesiveness measure, and is defined as follows, averaging the squared differences between all pairs of nodes in a clustering, gives a measure for the overall level of cohesion in the clustering. As with the $\mathcal{SOSE}$ measure, a random grouping

of nodes results in a very poor quality measure, whereas clustered points give better quality results. This measure can be formalized as follows:

$$\mathcal{A}_i = \frac{1}{|\mathcal{C}_i|^2} \sum_{u \in \mathcal{C}_i} \sum_{v \in \mathcal{C}_i} \| u - v \|^2 \tag{3.21}$$

and the sum of squared errors is:

$$\mathcal{MV} = \frac{1}{2} \sum_{i=1}^{|\mathcal{C}|} (|\mathcal{C}_i| - \mathcal{A}_i)^2 \tag{3.22}$$

Finally, as with the $\mathcal{SOSE}$ this measure is on the ordinal scale and must also be normalised for cross comparisons.

3.9 Methods for Geometric Graph Clustering

Geometric graph clustering is strongly related to "data clustering". *Data Clustering* is the grouping of *point data* into groups according to some labeled set of patterns. As such, a graph that has been drawn in two or three dimensions could be clustered as if the nodes were simply point data.

Some of the most important geometric clustering methods, most of which disregard the connectivity information include:

- Distance measure based algorithms

- Coordinate Bisection/Inertial Bisection

- Partitional algorithms/k-means

- Induced spanning tree of point set.

The area of data clustering typically deals with high dimensional data. Data elements having 150-200 attributes that can be included in a similarity measure are not uncommon. This thesis concerns itself with the general layout, clustering, and abstract representation of simple graphs. As such, further discussion of data clustering based on a data of high dimension is outside the scope of this thesis.

3.9.1 Coordinate/Inertial splitting

Coordinate splitting is a divisive method that employs the geometrical coordinates of the nodes in two or three dimensions to compute a clustering. Typically, it involves finding a plane parallel to the fixed coordinate *x-*,*y-* or *z-axis* that divides the graph into two sub-graphs. If the two sub-graphs should have an equal number of nodes then this is called *coordinate bisection*. Regardless of the relative size of the sub-graphs, the dividing plane is orthogonal to one of the axis planes. This method can be applied recursively to produce a hierarchical clustering in the form of a binary tree, which is a balanced tree in a coordinate bisection. Creating this type of balanced recursive node grouping with *x-* and *y-axes* is effectively the same as creating "bintrees" in image processing [253] and also relates to quadtrees for representing point data. Although in the worse case the quality of the partitions formed may be poor, this is a computationally cheap method that produces reasonable clusters in practice [80, 106, 223].

Different rotations of the node set in two or three dimensions can result in very different clustering than if fixed axes are used. This is a weakness of this clustering method but it can be over come by the "Inertial Bisection" method [212]. An *Inertial Bisection* method first determines the centre of mass and the principal axis of inertia, that is, the axis of minimum angular momentum. This forms the "adjusted axes" which are independent of overall node rotation. Another variant of this method, computes the separator of the node set using a circle rather than a straight line.

In this thesis we show how variants of this form of decomposition are useful not only for clustering the nodes of a graph but also for improving the performance of a classical graph layout method which results in drawings that can be viewed and explored across multiple levels of abstraction.

3.9.2 Partitional/k-means

The *k-Means* method is one of the most commonly used geometrical clustering techniques. In *k-Means*, the coordinates of the nodes, in n-dimensional space, are used to compute a clustering based on some similarity measure. Such a similarity measure is typically based

on squared Euclidean distances. The *k-Means* method is popular due to its simplicity, ease of implementation, time complexity of $\mathcal{O}(n)$ (where n is the number of clusters), and the fact that it often returns good clustering results.

The *k-Means* method proceeds as follows:

I First randomly select k nodes from the graph that are used as the initial cluster centres. Or select k random points within the hypervolume containing the graph drawing

II Assign each node of the graph, to its closest cluster centre

III Recompute the cluster centres, using the centroid of current membership

IV Repeat the assignment process if the "minimization criterion" is not met.

The *minimization criterion* is used to determine whether there has been a minimal or zero change in the state of the clustering. The state of the clustering can be measured in terms of a sum of squared error criterion ($\mathcal{SEC}$) which has a similar formulation as the $\mathcal{SOSE}$ measure shown in 3.20 . The centroid of a particular clustering $\mathcal{C}_i$ is $\mathcal{M}_i$ and $\mathcal{SEC}$ is typically expressed as:

$$\mathcal{SEC} = \sum_{u \in \mathcal{C}_i} \| u - \mathcal{M}_i \|^2 \tag{3.23}$$

The *k-Means* method is sensitive to the initial clustering and may converge to a local minimum if the initial clustering is poorly selected [140]. Numerous specializations to the basic *k-Means* algorithm have been developed [4, 8, 192, 259] to overcome the problems associated with selecting an initial clustering. Several of these methods also incorporate a thresholding scheme for the *split-push* and merging of clusters, such as ISODATA [192]. Recent methods, restrict the centroid $\mathcal{M}_i$ to be a node of the cluster, the so called *medoid* of the cluster [140].

3.10 Geometric Hierarchical Graph Clustering Methods

A graph does not constitute spatial data as it simply contains nodes and edges (no geometry). However, an attributed graph, where some of the attributes are geometric, does
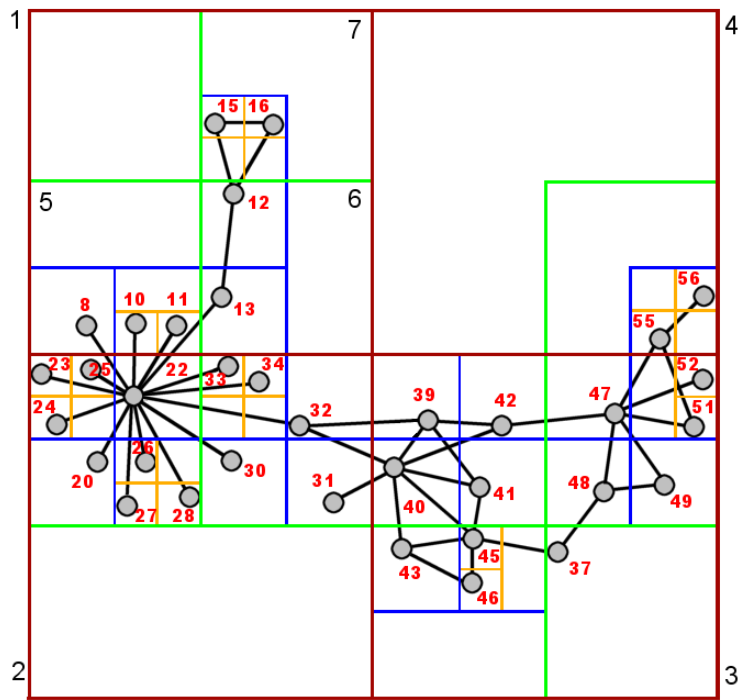
Figure 3.14: A graph drawing overlaid with a hierarchical space decomposition.
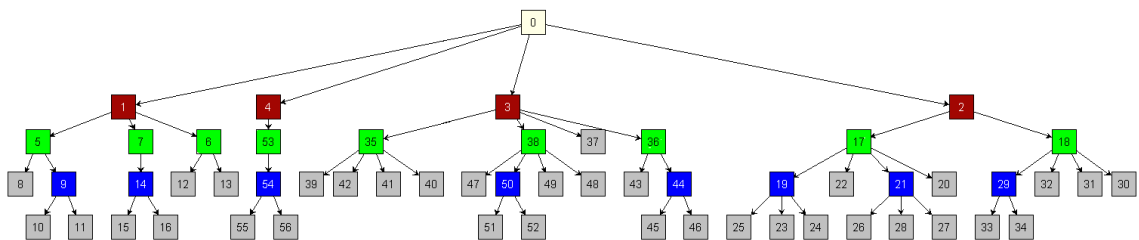
Figure 3.15: The inclusion tree $\mathcal{T}$ of a Hierarchical Compound Graph, of the graph and space decomposition shown in Figure 3.14

constitute spatial data. Graph drawing algorithms synthesize geometries for graphs without geometric attributes. Once the graph has an associated geometry, then a hierarchical geometric space decomposition method can be used to recursively cluster regions of space containing the node locations, such as that shown in Figure 3.14. This recursive clustering is a hierarchical clustering of regions of two or three dimensional space.

Given a graph drawing, then any hierarchical space decomposition induces a hierarchical graph clustering, such as that shown in Figure 3.15. This hierarchical geometric clustering, which can be measured with geometric measures, induces a graph theoretic clustering, which can be measured with theoretic measures. The graph theoretic clustering is also used to form the hierarchical compound graph associated with this hierarchical geometric space decomposition.

We aim to use computationally inexpensive hierarchical space decomposition methods, such as quadtrees as shown in Figure 3.14, to cluster the nodes of a large graph drawing. These hierarchical clustering are then used to create a hierarchical compound graph with implied edges. This compound graph is then used in our FADE paradigm to draw and abstractly represent large graphs using "visual précis".

3.11   Visual précis

Here we address the effective use of screen real estate and the computational effort involved in rendering large graphs. We use the hierarchical compound graph model to support our notion of a "visual précis", which is simply an abstract visual representation of the underlying graph. Our "visual précis" are related to mesh generation in the field of surface modeling [13, 170, 171]. This relationship comes about because the space decomposition methods, such as quadtrees, are used in both the formation of hierarchical compound graphs and approximate mesh generation. However, in surface modeling a mesh point is an approximation of some point in space. In a "visual précis", a region of space defines a cluster, which is an abstract representation of a set of relational data elements and their interrelationships.

Formally, a *visual précis* is a two or three dimensional projection of a précis extracted
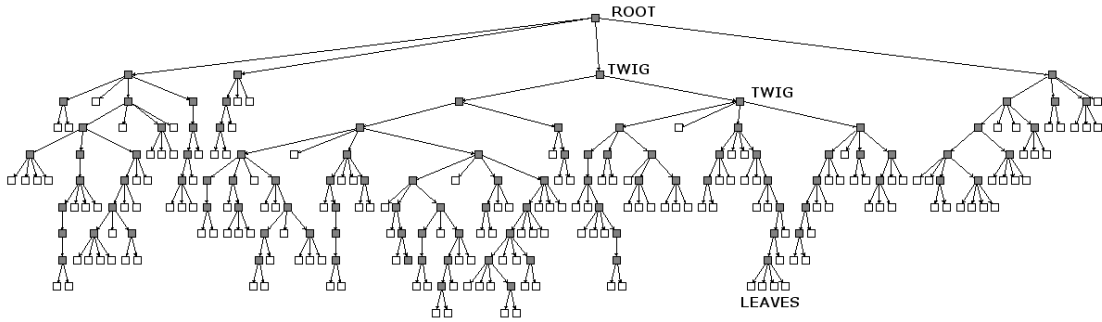
Figure 3.16: A hierarchical inclusion tree $\mathcal{T}$ of a 168 node graph.

from a hierarchical compound graph. Recall that, a *précis* consists of a set of clusters, implied edges, real nodes, and real edges. A précis may contain any combination of these, as long as it represents an abstract view of the entire underlying graph. Précis, which primarily contain clusters and implied edges are called *high level précis*. In a précis, the only graph edges are between nodes which are both included in the précis. All other edges are included as implied edges or are abstracted into clusters. The definition of a précis can apply to any type of inclusion tree regardless of its arity. As a result, regardless of the shape of space decomposition used to form the hierarchical compound graph, these visual précis drawing techniques can be applied.

Unless otherwise stated, an implied edge exists between two clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ when at least one real edge $e = (u, v)$ connects nodes in the two clusters, that is, $u \in \mathcal{C}_1$ and $v \in \mathcal{C}_2$. An implied edge can also exist from a cluster $\mathcal{C}$ to a single node v, where $e = (u, v)$ and $u \in \mathcal{C}$.

The hierarchical inclusion tree $\mathcal{T}$ shown in Figure 3.16 represents a possible clustering of the nodes of a graph. Here the root cluster contains four clusters with 33, 5, 107 and 23 nodes respectively. We will make reference to this inclusion tree in the description of our abstraction and viewing techniques. Note, this tree is not a hierarchical compound graph as it only represents the inclusion tree $\mathcal{T}$ of the hierarchical compound graph, without reference to edges $\mathcal{E}$ or the implied edges $\mathcal{I}$. An example of a hierarchical compound graph is shown in Figure 3.2.

The viewing methods introduced in this thesis, to visualize précis extracted from hierarchical compound graphs, include:

- Event Horizon views

- Horizon views

- Cut views/Multi-cut views/Distortions

- Surface Views

- Bell Curve Views.

An *event horizon* is simply a précis which consists of nodes and clusters from any level of abstraction. As such, an event horizon view is our general term for the clustered graph drawing of a précis. A *surface view* is our term for a three dimensional projection of an event horizon. Such surface views are three dimensional tree-maps. In a surface view the level of abstraction for a particular node or cluster, is typically rendered as a colour on a simple "hypsometric scale". A *hypsometric scale* is a a scale in the margin or cartouche of a map that shows which shades or colours represent which elevations.

The *visual weight* of a particular visual précis, is the percentage of clusters, nodes, implied edges, and edges drawn as compared to the total number of nodes and edges in the graph. *A visual weight cutoff* is simply a user defined percentage used in determining how many clusters, nodes, edges, and implied edges can be included in a given précis. Other more sophisticated methods for determining the visual weight cutoff can include measuring the processor speed, the graphics card capacity, the screen size, and monitor resolution.

Précis which include many high level clusters and implied edges, allow us to draw visual précis which reduce the visual weight, complexity, and rendering time for the visualization quite dramatically. This reduction does not come free since we are removing detail and showing only approximations. In discussing each type of view, we note the possible effects of these reductions.

Horizon views

Intuitively a horizon view is the extent to which the user can see into the hierarchical compound graph. It is a level based view, which is the simplest form of abstraction using context sensitivity, elision, and user control techniques. Hierarchical compound graphs
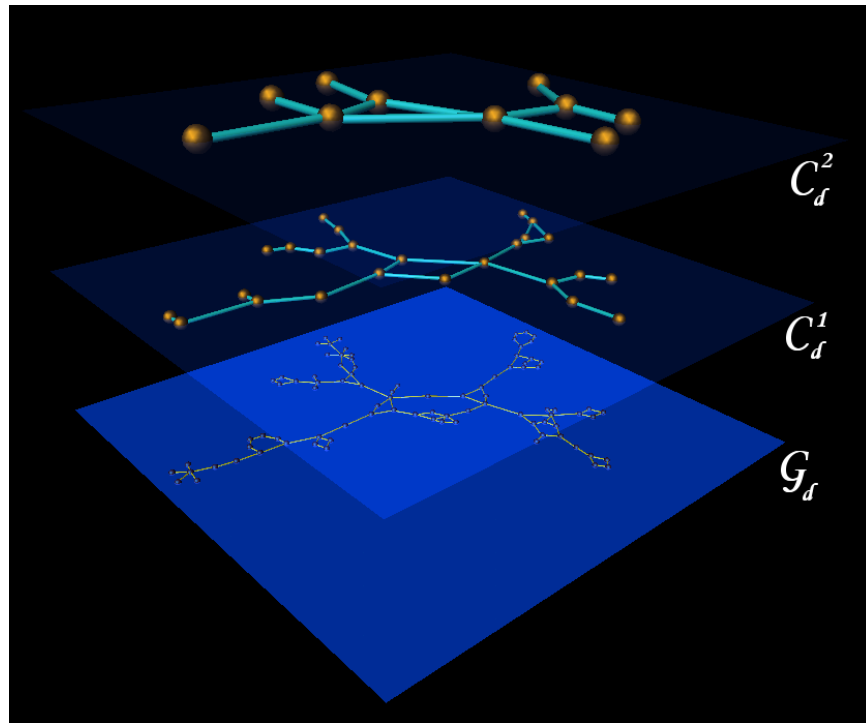
Figure 3.17: Two Horizons of a Graph, as a Multilevel Drawing.
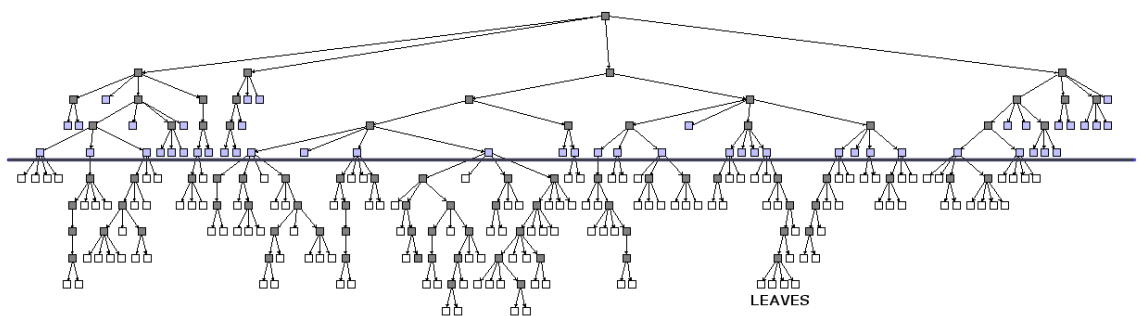
Figure 3.18: Nodes and clustered nodes (of $\mathcal{T}$) in a 4th level horizon view

formed using standard recursive space decomposition techniques, typically do not have a uniform node depth; that is, the distance from leaf node to the root, as shown in Figure 3.18. Such space decomposition techniques can be augmented to ensure that all the nodes of the graph exist only at the bottom of the tree, by repeatedly clustering shallow nodes until they are of the same depth as the deepest nodes.

Instead of this potentially costly algorithmic approach, we prefer to define our views in line with the existing recursive space decomposition methods. As such, our horizons are précis, which contain clusters at the same level of abstraction (depth) along with real nodes which exist at this level of abstraction or higher. Real edges between nodes in the précis are included. Edges between nodes in different clusters are also included as implied edges and edges between real nodes and clusters are included as implied edges.

The level of abstraction (depth) for a particular horizon view can be determined in several ways; these include: interactive selection, *a priori* user selection, or selection based on some visual weight cutoff. Typically, high level horizon views have a much smaller visual weight than the underlying graph drawing. Here, the cutoff can be used to determine the lowest level horizon drawing permissible. Figure 3.19 shows an underlying graph of 400 nodes overlaid with a hierarchical space decomposition to the third horizon level. The root cell is split into four quadrants (level 1). Each of these is split into a maximum of four quadrants (level 2). Each of these quadrants is again split to form level 3. This level defines a précis of clusters implied edges and nodes which can be extracted from the hierarchical compound graph.

There are several ways to draw a visual horizon defined by a précis. Figure 3.19 shows both a two dimensional "tree-map" and a "clustered graph drawing" of the horizon defined above. A *tree-map* consists of squares showing the relative densities of nodes in a particular region. A *clustered graph drawing* shows each node and cluster along with the implied edges, and edges between them.

Typically there are many visual horizons that can be extracted from a hierarchical compound graph, as shown in the three dimensional drawing in Figure 3.20. The multilevel representations of Feng *et*. *al* [73, 86], showed several ways to visualize the levels of a clustered graph in a single multilevel representation. Here, the précis extracted in the form
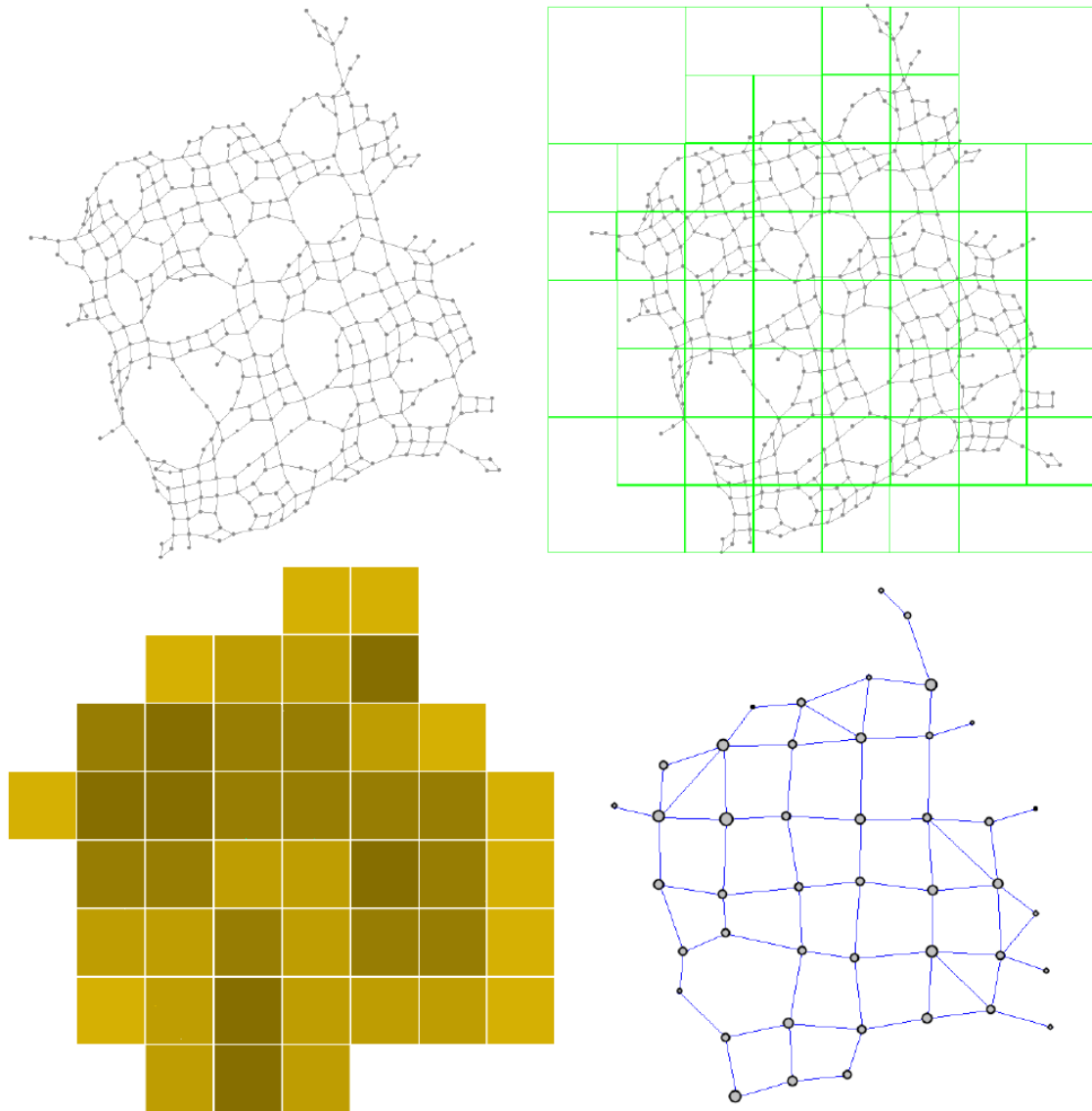
Figure 3.19: 400 node graph, and a horizon, drawn as a *tree-map* with hypsometric tints and as a *clustered graph*.
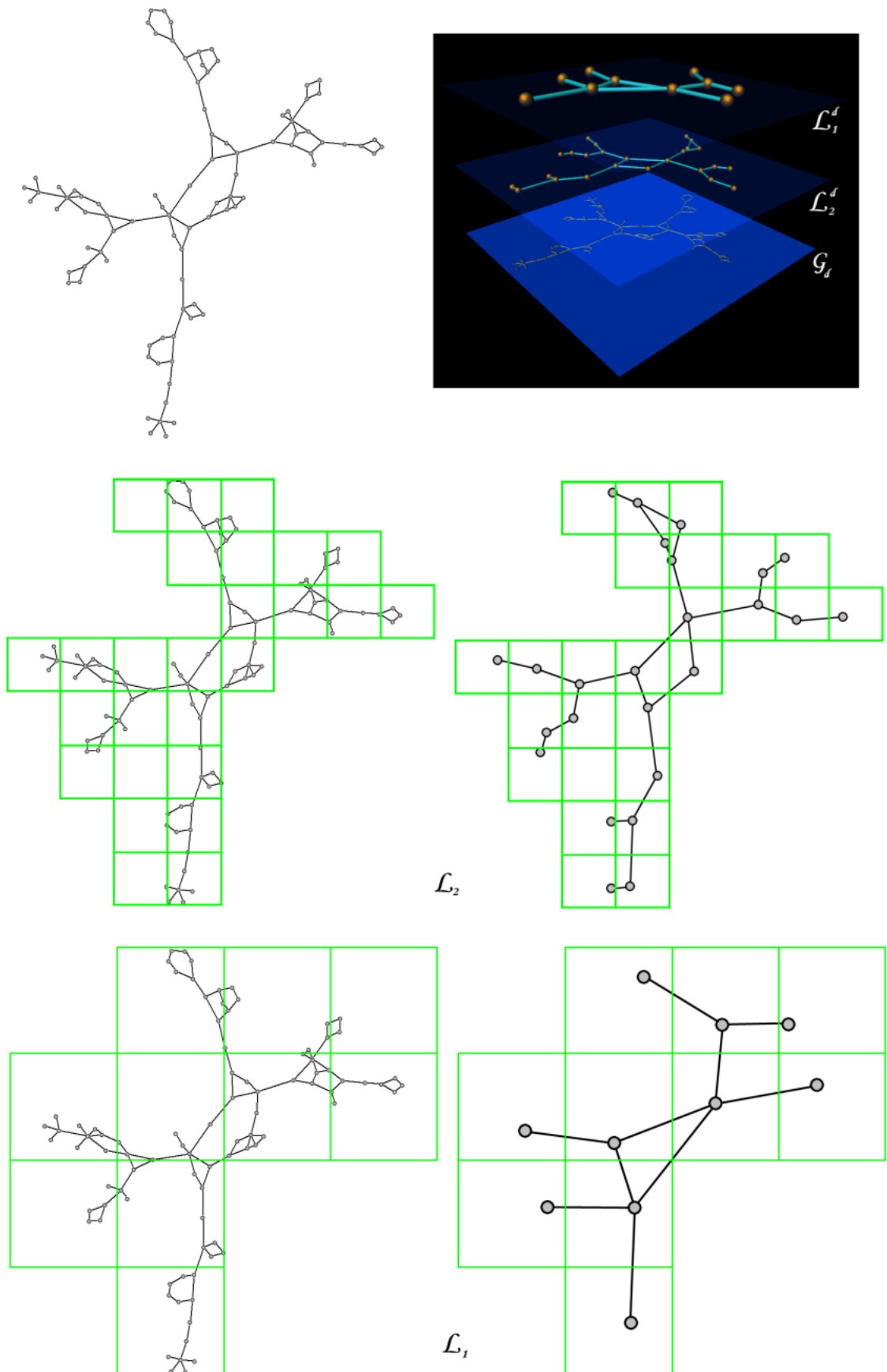
Figure 3.20: Different horizons and their drawings and a multilevel representation view.

of visual horizons are used to reduce, rather than increase, the amount of graphical information on display. The goal here, is to show how the visual horizon drawings can reduce the visual weight, thereby making them suitable for large scale graph drawing, rather than single multilevel representations.

The top left image in Figure 3.20 shows a graph of 93 nodes and 111 edges drawn in two dimensions. The top right image shows a multilevel representation of two horizons $\mathcal{L}_1$ and $\mathcal{L}_2$ along with the underlying graph drawing $\mathcal{G}_d$, similar to those described in [86]. In the multilevel view, each drawing is embedded in a transparent plane. Clearly $\mathcal{L}_1$ is more abstract than $\mathcal{L}_2$ as it uses fewer clusters and implied edges to represent the overall structure of $\mathcal{G}$.

The clustering that induces the horizon $\mathcal{L}_1$ is shown in the middle left of Figure 3.20. Our visual horizon drawing of $\mathcal{L}_1$ (overlaid with the space decomposition information) is at the top on the middle right. The clustering that induces the horizon $\mathcal{L}_2$ is shown on the bottom left of Figure 3.20 along with the visual horizon drawing on the bottom right.

Single high level horizon drawings are useful as they can provide an initial simplified view of the structure of large graphs. However, they can also be used in interactive systems where users can decrease the depth of the horizon. This in turn results in less detail being shown which provides a more abstract view of the data. Alternatively, they can increase the depth of the horizon to show a greater amount of uniform detail.

In an interactive system some form of animated visual transition effect is required as the user moves from one level to the next. We identify three possible transition techniques:

- **Fading**: The nodes, clusters, implied edges and edges are faded in or out as appropriate when moving between horizon levels.

- **Animation**: Moving to a deeper horizon, the nodes and clusters are animated as moving from their parents location to their new locations. Moving to a lower horizon has the opposite effect.

- **Combination**: Nodes/clusters are faded in and then animated to their final locations.

Using a level view, it is possible to reduce the visual weight of the visualization quite dramatically. Taking a horizon a few levels up from the deepest nodes, often reduces the

Figure 3.21: On the left a graph drawing of 1000 nodes and 1997 edges. On the right a high level view with 31 nodes and 33 edges.

visual weight by over 90%. As a technique for initial visual inspection of the overall structure of a large graph, horizon viewing seems promising. For example, the horizon drawing on the righthand side of Figure 3.21 represents a 98% reduction in visual weight, yet the overall structure remains clearly visible.

However, this drawing technique clearly has some drawbacks. If the user wants to inspect a node that happens to be at the very lowest level of the hierarchical compound graph, then the entire graph must be drawn according to this drawing technique. This brings us back to the original problems of rendering time and the effective use of screen space.

For approximate or overview drawings, horizon views clearly reduce the visual weight and computational effort in creating the picture. However, for interactive data exploration, this technique can be used initially and then in conjunction with other techniques described here.
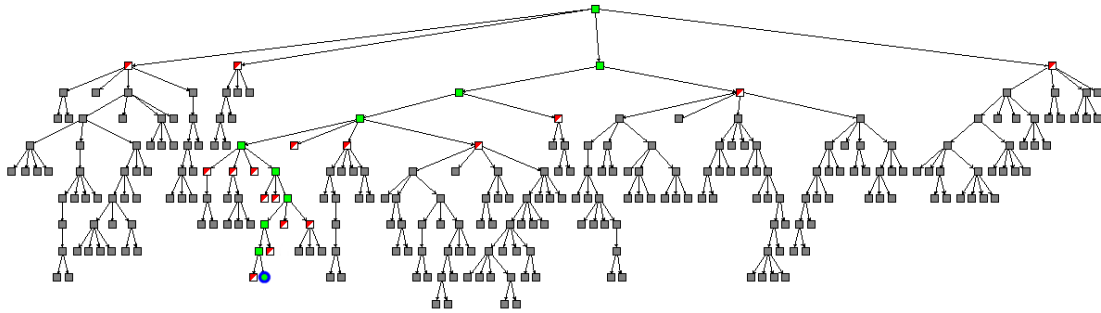
Figure 3.22: Simple cut from selected node to root of the tree in Figure 3.16
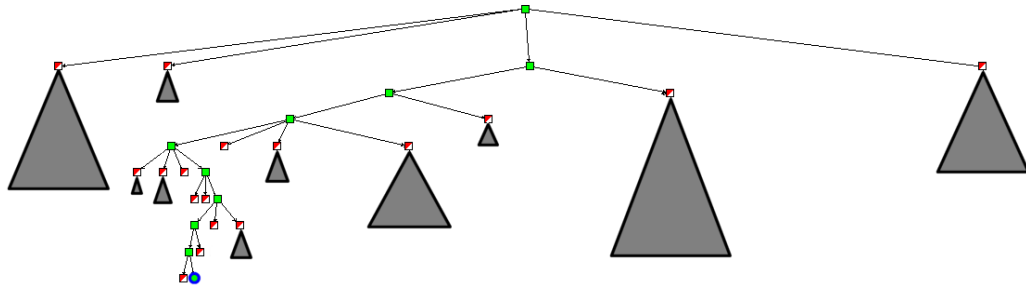
Figure 3.23: A précis view of the inclusion tree, with the sub-tress elided, of the cut in Figure 3.22

Cut views

Graph visualization systems are typically used in exploratory data analysis. A common operation is to search for, or query the location of a particular node in the graph. As noted above, to visualize a particular node the horizon must be deep enough to encompass it. This often results in the drawing of large amounts of graphical information, simply to see one node at the required level of detail. Here we introduce techniques, that use the inclusion tree to support the creation of visual abstractions for such queries and searches.

A *cut* is a précis through a hierarchical compound graph from a single node. A cut is used to shown the *detail*, *local-* and *global-context* of that node. The detail is given by the node itself, the local context is given by clusters towards the bottom of the inclusion tree (close to that node) and the global context is given by clusters towards the top of the tree.

A *cut method* creates a précis from a single node. Cut methods can be formulated in different ways, with the most basic cut method shown in Figures 3.22 and 3.23. Figure 3.22 shows a selected node and a cut from that node. Here the cut method forms the précis by
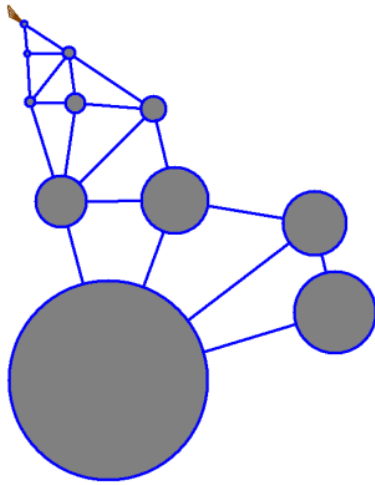
Figure 3.24: Straight cut from a single node to the root of a 5000 node graph
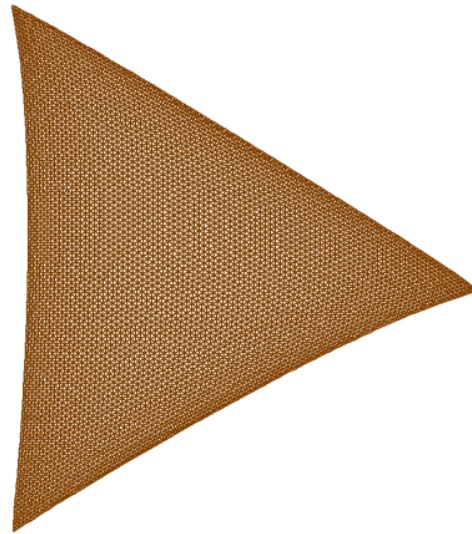
Figure 3.25: Underlying drawing of the 5000 node graph

marking the path from the node u to the root of the inclusion tree as *open*. Edges from u to other nodes, whose parent clusters are marked as open, are included in the cut. Nodes or clusters, whose parents are marked as open and implied edges between those clusters or nodes are included in the cut. This method is called a *straight cut* and Figure 3.23 shows an abstract view of the clusters and nodes of the cut in terms of the inclusion tree where sub-trees are elided away. The drawing of a cut is useful for showing one node along with the local and global-context for the rest of the graph.

Take for example a uniform distribution of nodes based on a k-way hierarchical decomposition of space with $\mathcal{N}$ nodes. Then a cut shows k nodes in detail and $k - 1(\log_k \mathcal{N})$ nodes or clustered nodes in a local and global context. For example, 40 nodes or clustered nodes are displayed when a single node is selected in a million node graph drawing, which has a uniform distribution, and is clustered according to a quadtree hierarchical space decomposition.

Figure 3.24 shows the drawing of a cut, extracted by the straight cut method from a hierarchical compound graph, based on the underlying graph shown in Figure 3.25. Clearly this is not a great visualization, as much detail and some structure have been lost, when compared with the original in Figure 3.25. However, if different projection techniques are used, then the nodes can be drawn larger and the clusters smaller, hence balancing the
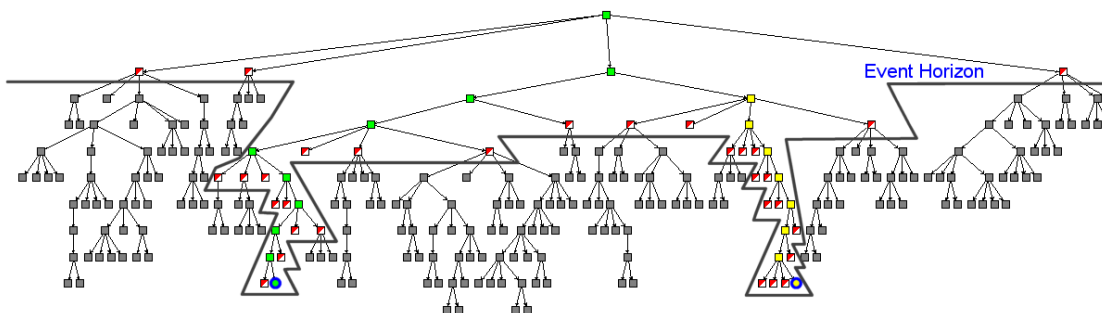
Figure 3.26: A multi-cut view of two selected nodes to root of the tree in Figure 3.16
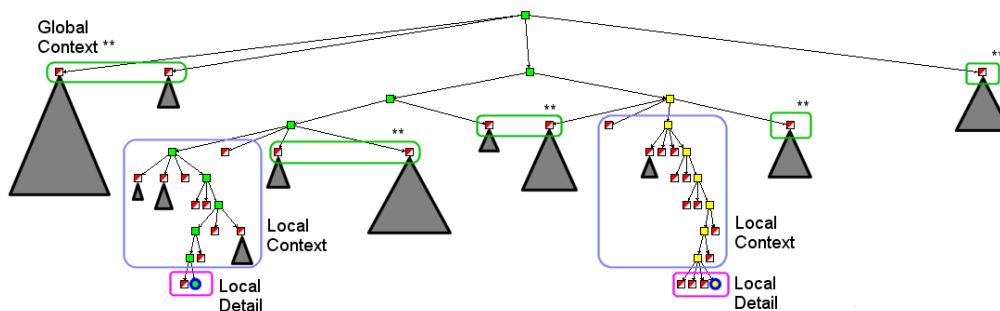
Figure 3.27: The different contexts and elided sub-trees for the multi-cut in Figure 3.26

overall picture and allowing the drawing of node labels.

The cut technique can be extended to show multiple cuts from a set of nodes; and this is called the *multi-cut* view. Figure 3.26 shows an inclusion tree overlaid with an event horizon, as a result of computing two straight cuts from two selected nodes. Figure 3.27 shows the areas of local detail along with the *local-* and *global-*context for each node. The nodes and clusters on different levels of abstraction, along with the edges and implied-edges form the event horizon of this précis. A drawing of this précis, which includes different levels of abstraction, is an event horizon drawing.

Here the local-contexts are disjoint but if the nodes were geometrically close they may well share a larger local-context area. This approach is useful for addressing the classical *detail-in-context* problem of showing nodes in detail along with their global connectivity. Figure 3.28 shows a multi-cut from 12 nodes in three groups on the periphery of the hierarchical compound graph. Figure 3.29 shows a multi-cut from 60 nodes in six groups. Where the individual cuts intersect they form an event horizon. In the drawing of a multi-
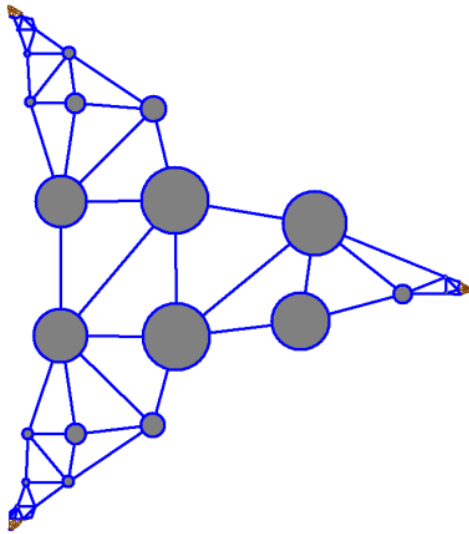
Figure 3.28: A multi-cut based on 12 nodes from the graph in Figure 3.25
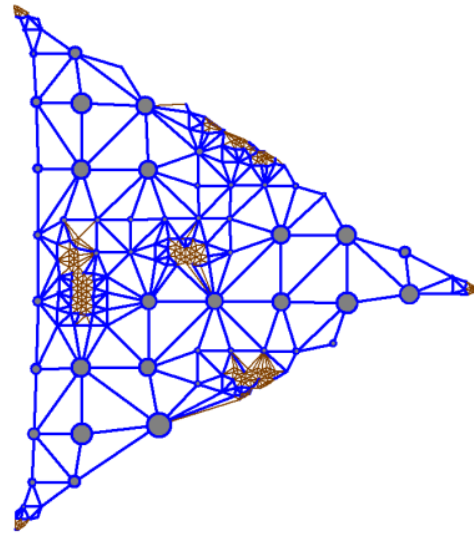
Figure 3.29: A multi-cut based on 60 nodes from the graph in Figure 3.25

cut, the local contexts become shared and the global context is shown at a lower level of abstraction. Figure 3.29, has a visual weight of 5%. This represents a 95% reduction in the number of nodes required to effectively visualize this node set while retaining a context view.

These cut views allow for an approximate drawing of a set of nodes, which reduces the visual complexity and computational effort in rendering the visualization. As noted above, different projections of the underlying event horizon drawings are possible. Figure 3.30 shows a simple single focus fisheye view of the drawing in Figure 3.29. Other more sophisticated projection techniques include multiple foci [208, 257] and intelligent zoom techniques [289].

Surface Views

As noted earlier, a surface view is a three dimensional projection of an event horizon précis extracted from a hierarchical compound graph. These visualizations clearly show the different levels of abstraction in multi-cuts from a hierarchical compound graph.

Figure 3.31 shows a graph drawing from our case study in Chapter 6. The event horizon drawing of a multi-cut is shown in Figure 3.32. The surface view of this event horizon is rendered as a three dimensional picture in Figure 3.33. This surface shows the nodes drawn
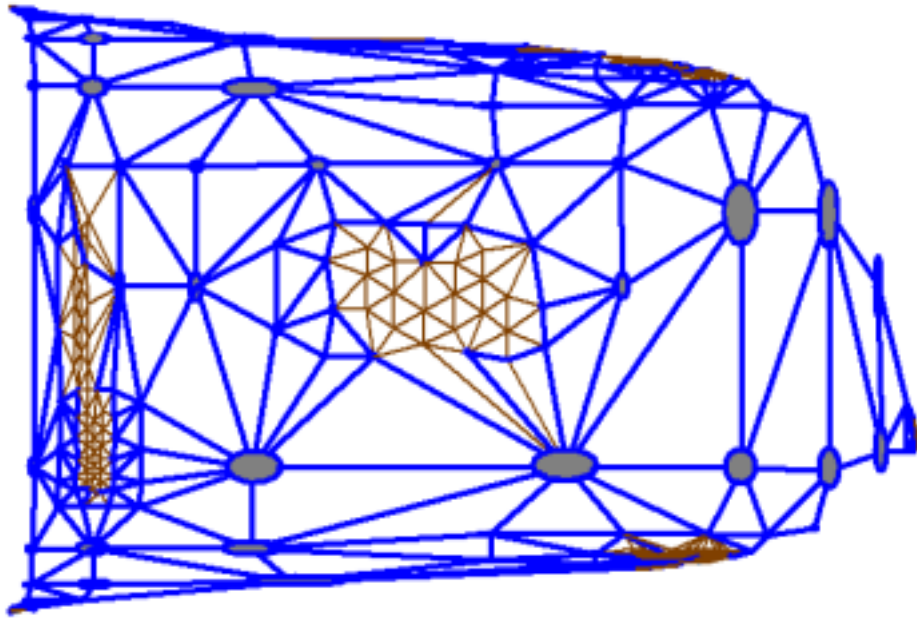
Figure 3.30: Simple fish-eye projection of the 60 straight cut-slice view from Figure 3.29

at the "highest level", the shared local context as the "middle level", and the global context at the "lowest level". Figures 3.34, 3.35, 3.36 and 3.37, show other multi-cuts and surface views of this graph (the peaks in the three dimensional surface views represent nodes in the précis).

Bell curve views

Cut-views are formed by well defined cuts through the cluster tree structure. This can result in the detail and local context of a cut view not including nodes and clusters that are geometrically close, as a result of them being far away in the inclusion tree structure. This problem can be overcome by the use of neighbourhood techniques such as "Bell curve" views. A *bell curve view* is a simple *quadric surface* through a volume defined by the layout and the height of the cluster tree. This can be extended to a surface through 4 dimensional space for three dimensional drawings, where the fourth dimension is defined by the height of the cluster tree. In three dimensions the curve is a *trace* which is rotated about the node from which the curve is defined. This rotation forms a surface cut through the volume defined. The précis is then formed as follows: nodes inside the volume are
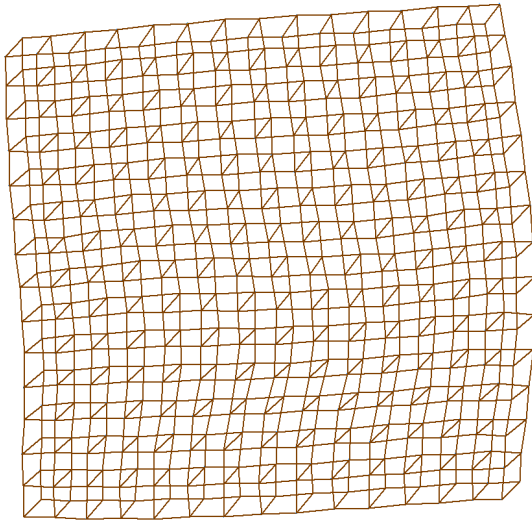
Figure 3.31: 2D drawing of the underlying graph rdb450
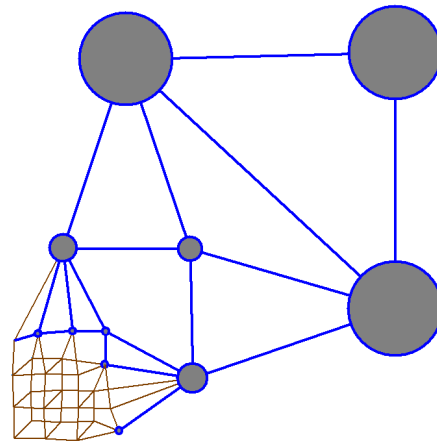
Figure 3.32: 2D drawing of a multi-cut (event horizon), from Figure 3.31
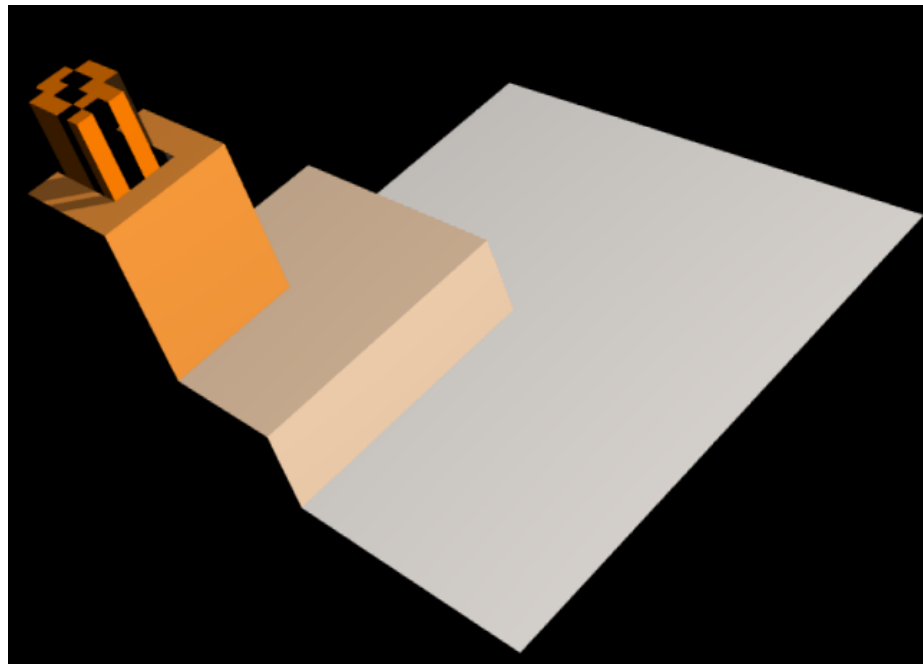
Figure 3.33: 3D drawing of an event horizon, showing detail, local- and global-context using hypsometric tints
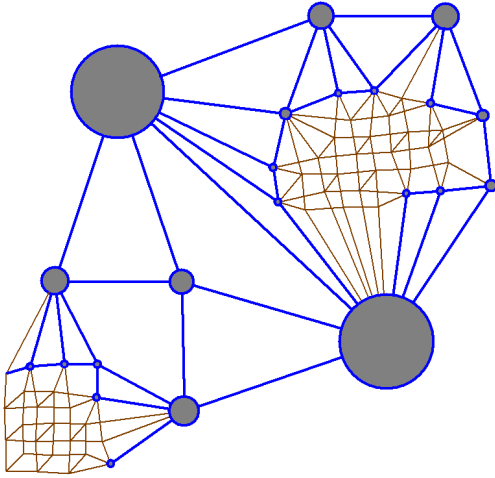
Figure 3.34: 2D drawing of an event horizon extracted from rdb450, showing two areas of detail
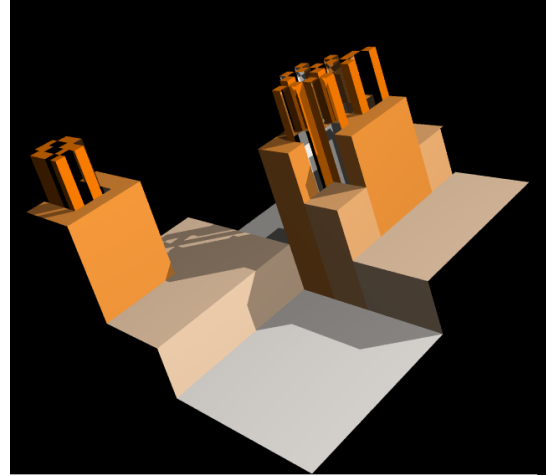
Figure 3.35: 3D drawing of an event horizon, showing two regions of detail
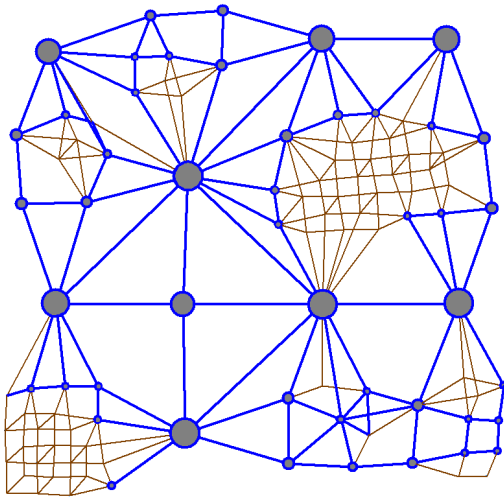
Figure 3.36: 2D drawing of an event horizon extracted from rdb450, showing multiple areas of detail
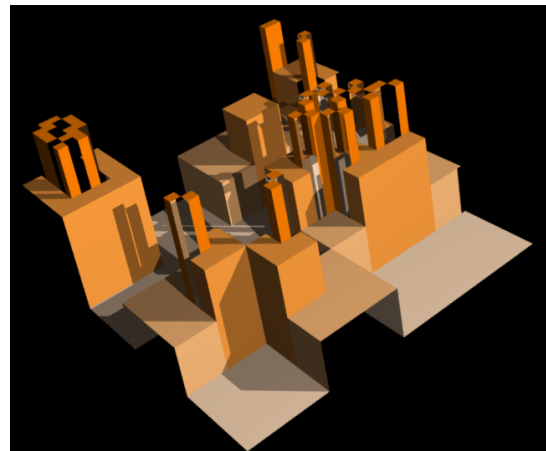
Figure 3.37: 3D drawing of an event horizon, showing multiple regions of detail

included in the précis. However, if all children of a parent inside the volume, are not inside the volume then that cluster is included in the précis. The highest clusters containing only nodes not covered by a cluster already in the précis are added to the précis. This surface generates the geometric *local-context* of a node by including geometrically close nodes and a *global-context* be including clusters far from the node. As with level views, these simple quadric surfaces define an event horizon which can be drawn. Nodes inside the volume are drawn at the lowest level of detail and the closest clustered nodes to these nodes are drawn at varying level of local and global context.

A multi-cut method which takes different "angles" through the cluster tree can approximate a surface. This technique may provide an elegant parameterized approach to the generation of detail-in-context views.

3.12   Remarks

The hierarchical compound graph, described in Section 3.3, provides the basis for the hierarchical compound graph quality measures introduced in this thesis. The geometric clustering approach taken here is based on a recursive decomposition of space. This clustering method, as we show in the next chapter, creates inclusion trees that can greatly improve the performance of classical force directed layout methods. The visual précis drawing techniques presented, dramatically reduce the visual weight and computational effort in creating the visualizations.

In this thesis we do not propose to study the effectiveness of any of these drawing techniques but rather the quality of the drawings of the précis extracted. Given the number of possible précis for any graph is large, we restrict our interest to the clustering and graph drawing aesthetic measurement of horizon views. There are typically $O(\log n)$ horizons in any given hierarchical compound graph.

Along with measuring computational performance, classical graph drawing aesthetic measures, and our quality measures the aim is also to study the quality of the horizons based on aesthetic measures in each of the case studies.

The FADE paradigm

"Art, like morality, consists of drawing the line somewhere." - Gilbert Chesterton

This chapter describes the FADE paradigm which is embodied in a suite of graph drawing algorithms. These algorithms are based on the hierarchical compound graph model introduced in Chapter 3 for the clustering, visual representation, and abstraction of large amounts of relational information. Each member of the FADE suite uses a fast geometric clustering of the locations of the nodes based on simple recursive space decomposition techniques. This geometric clustering induces a graph theoretic hierarchical clustering which is used to build a hierarchical compound graph for viewing and measurement. The hierarchical compound graph is then input to force-directed algorithms that improve the relation between Euclidean and graph theoretic distances in the underlying drawing. This improvement in the underlying drawing in turn improves the graph theoretic clustering and hence improves the quality of the hierarchical compound graph. Iterating this process further improves both the drawing and the hierarchical compound graph.

Force directed graph drawing methods, such as those outlined in Section 2.2.4, proceed by making small iterative changes to the layout of the graph; each change reduces some energy function for the system of forces. Although sensitive to the initial layout, these methods often produce aesthetically pleasant drawings due to the nature of the interacting forces. Groups of related nodes are drawn together, whereas unrelated nodes are typically drawn far apart. Several authors have commented that force directed layouts often exhibit the natural clusterings of the graph that one hopes to discover and visualize [136, 173,

249]. Our hypothesis, called the *progressive cycle* shown in Figure 4.1, marries graph-drawing, hierarchical compound graph creation, and visual précis. The progressive cycle suggests that a drawing with high energy has poor quality both as a visualization and as a hierarchical compound graph according to any geometric or graph theoretic measure used. As the forces are applied and the drawing improves and moves towards a lower energy state, the hierarchical compound graph improves.

We aim to show that the accuracy of the force calculation in our FADE paradigm is not the determining factor for the quality of the graph drawings produced. Section 4.2 describes how force directed graph drawing algorithms relate to the N-BODY problem in particle simulation. We also review the existing force directed algorithms which have derived methods from this area in Section 4.2 and give a specific example in Section 4.2.1. Section 4.2.2, introduces "tree codes" which provide the basis for the FADE suite of algorithms described later in this Chapter. Section 4.3 introduces the FADE paradigm, along with describing different "cell opening criterion". Section 4.3.4 describes the scaling and accuracy measure used to compare the performance and accuracy of the FADE suite of algorithms, with classical force directed methods. Section 4.4 describes the basic two dimensional FADE2D algorithm in detail, along with an overview of the geometric clustering algorithm used. Section 4.5 describes a three dimensional force directed layout algorithm called FADE3D. As we show here and in our case studies, the use of an initial random layout may not be suitable starting point for the drawing of large graphs using a force directed layout. Section 4.6 describes a "wave front" algorithm which may be used in conjunction with a two or three dimensional FADE method; this aims to produce a reasonable initial drawing (low energy state) from which further iterative improvements are possible.

4.1   Overview

The main idea of the FADE paradigm is the following. Given an initial layout method, such as random placement, a graph can be assigned geometric attributes. This process creates a graph layout, which can then be rendered to produce a graph drawing. This is the classical graph drawing "pipe-line" [146], as indicated in Figure 4.1.
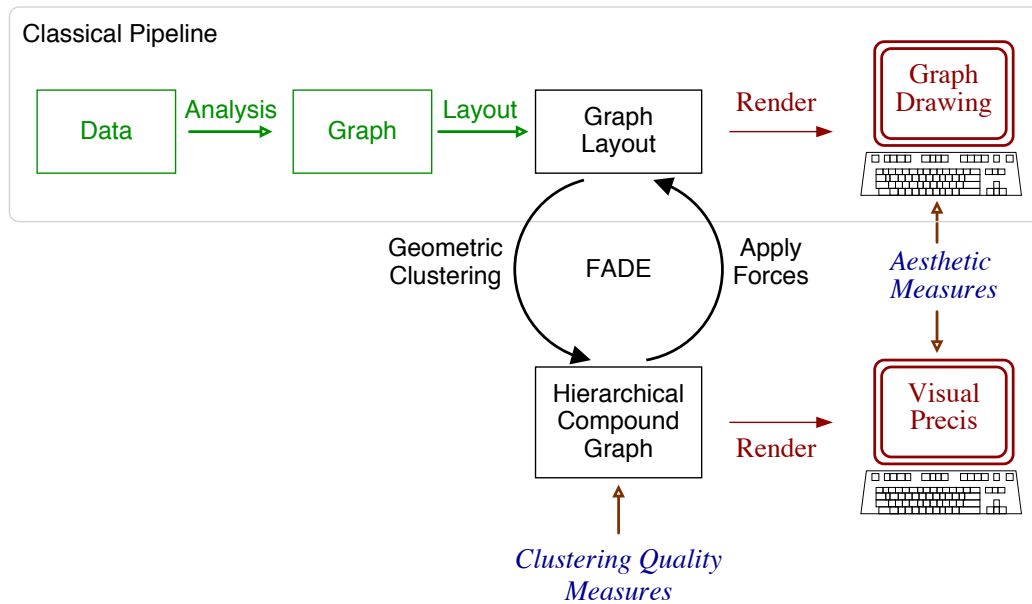
Figure 4.1: The *Progressive Cycle* of Drawing and Hierarchical Compound Graph Improvement

In the FADE paradigm, we take the graph layout and perform a geometric clustering (typically by recursive space decomposition) of the locations of the nodes. This process, along with implied edge creation, forms a hierarchical compound graph, as shown in Figure 4.1. The hierarchical compound graph, which includes the decomposition tree, allows us to approximate the nonedge forces in our force directed graph drawing algorithms. Using the decomposition tree, FADE computes forces; the nonedge forces may be approximately computed. After computing the forces, they are applied, and the underlying graph layout is updated. This in turn requires the recomputation of the hierarchical compound graph, which was formed on the previous locations. This process, if iterated, is called the *progressive cycle* where, as the graph drawing improves, the quality of the hierarchical compound graph also improves.

Roughly speaking, the progressive cycle of FADE proceeds as the repeat loop in algorithm 1. The stopping condition can be a simple counter, an evaluation based on an aesthetic measure, or a condition derived from the graph clustering measures presented in this thesis. Further discussion of possible stopping conditions is outside the scope of this thesis, instead we focus on a discussion of the nonedge force computation. The progressive cycle can be slightly modified for different platforms. Instead of computing the nonedge

---

Algorithm 1 FADE paradigm
begin
  Compute Initial Layout
  repeat
    Compute Edge Forces
    Move Nodes
    Construct Geometric Clustering
    for each node v **do**
      Compute Approximate Nonedge Forces on v
    end for
    Move Nodes
    Update Bounding Area/Volume
  until Stopping Condition
end

---

force for each node in turn; the nodes and pseudonodes may be added to an "interaction list" for each node. An *interaction list* is the list of twig nodes and leaf nodes of the inclusion tree that are involved in the calculation of the approximate nonedge force on v. Such lists can be used, for example, in a parallel workload distribution of force calculations.

The quality of the hierarchical compound graph can be evaluated by various clustering quality measures introduced in Section 3.4. The hierarchical compound graph is also used to extract précis which can be rendered as graph drawings. The quality of these drawings, along with the drawings of the underlying graph, can be evaluated using the graph drawing aesthetic measures described in Section 2.2.2. Evaluations with these measures, using data from two application domains, are discussed in Chapters 5 and 6.

In general, the performance improvement in our FADE paradigm comes from computing nonedge forces using a recursive approximation of groups of nodes, rather than all the *node-to-node* nonedge forces directly. To achieve this, we form a recursive space decomposition of the locations of the nodes in a graph drawing, to form a geometric clustering of the graph. Different space decompositions generate different recursive geometric clusterings of the nodes of the graph. The recursive clustering, represented as sub-trees, does not directly produce a high quality geometric clustering of the node positions, nor is it necessarily a high quality graph theoretic clustering. However, the clustering does facilitate a dramatic improvement in the performance of force directed algorithms, as shown in Sec-

tion 4.3 below. Further, it allows for multi-level viewing of huge graphs at various levels of abstraction. As the quality of the drawing improves (as it reaches a lower energy state of the force system), the quality of clustering, exhibited by the inclusion tree improves to a reasonable amount. This clustering, if it is of sufficient quality is appropriate for visual abstraction. Evaluations of the performance of FADE in application domains is discussed in Chapters 5 and 6.

The FADE layout and viewing paradigm provides a novel technique for clustering and drawing large graphs, along with a geometric viewing technique based on that clustering. This model draws together three difficult problems, that is, performance, abstraction, and clustering, and uses the same underlying model to tackle each. Moving from drawing, to viewing the graph in detail, to viewing a visual précis is integrated into one model. Basing the decomposition of the graph firmly in the geometric domain allows for a wealth of existing visual presentation and augmentation methods to be applied [38, 60, 121, 168, 263, 266, 292].

Our FADE paradigm is related to the clustering and visualization methods of Walshaw and Harel mentioned in Section 2.2.7. The main difference with the graph theoretic clustering approach of Walshaw [283] and Harel *et al.* [113, 115] is that we approximate forces based on actual geometric information, rather than preprocessed approximate forces. This difference allows our force-directed methods to be used in interactive drawings that show high-level intermediate results. The FADE method can also be applied to large graphs, without distorting the resultant drawing due to an approximated underlying graph clustering structure. All the performance and time measures in this chapter were computing using an implementation in Parlanse, running on a 500Mhz Pentium III processor with 512Mb of memory.

4.2   Particle Simulation

Modeling large numbers of interacting particles through long- and short-range forces has interested physicists for centuries (see [5, 125, 222, 275]). *Celestial mechanics* involves charting the motion and inter-play between heavenly bodies such as super-novas, black-
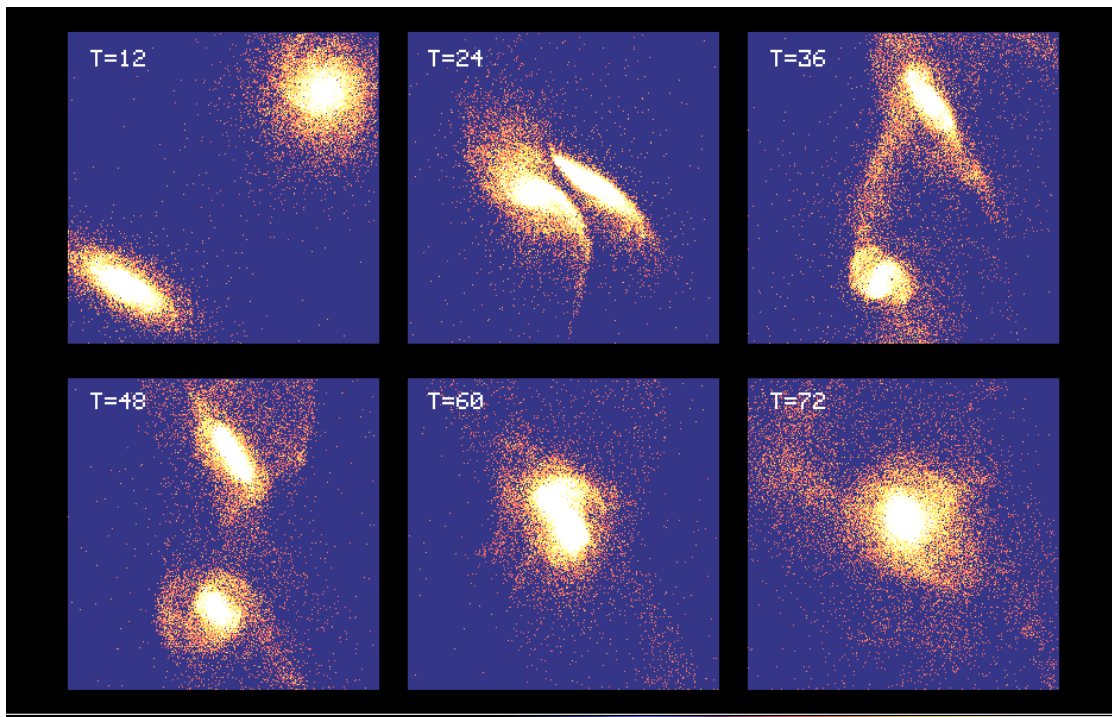
Figure 4.2: Fabry-Perot Image of the Simulation of two Interacting Galaxies. Reproduced by courtesy of Professor G. Bothun of The Electronic Universe Project [31]

holes, galaxies, stars, comets, and planets. Methods and techniques to model such interactions have been developed over the centuries but until recently modeling large particle sets was infeasible. Other fields of science also require modeling systems consisting of interacting particle sets, such as plasma physics, biological macromolecule simulation, and the vortex method in fluid dynamics [43, 125, 131, 222]. However, regardless of the domain, the same basic problem presents itself: the equation of motion of a system with more than two interacting particles defies an analytical solution.

As computers have become more powerful over the past few decades, particle researchers have developed simulators that plot the trajectories of multiple particles simultaneously. The simulation includes the position, mass, and velocity of each particle $i \in N$. As such, the force Φ on any one particle can be computed by considering the position and masses of all the other particles in the simulation along with any other external influences (such as magnetic fields in plasma simulations). For classical simulations this can be expressed as:

$$\Phi_{total} = \phi_{short} + \phi_{long} + \phi_{external} \tag{4.1}$$

Where,

- ϕ_{short} is a rapidly decaying function of distance, such as the Van der Waals force in chemical physics.

- ϕ_{long} is a long-range force, such as gravitational attraction or electrical repulsion.

- $\phi_{external}$ is an external force, typically independent of the position and number of particles, such as an external electric field.

Since $\phi_{external}$ is an independent force, it is calculated separately for each particle, which contributes $O(N)$ to the computation time for Φ_{total}. In such simulations ϕ_{short} is typically $O(N)$ as well, because the force decays rapidly and each particle interacts significantly only with a small number of its nearest neighbors. If only both of these force calculations were required to be taken together it would result in the development a very fast and scalable algorithm. However, the computation of ϕ_{long} presents a problem because if done directly, it requires $O(N^2)$ operations. So an algorithm that calculates Φ_{total} by direct particle to particle comparison for the ϕ_{long} force will be a *quadratic time algorithm* and as such will only be suitable for simulations of a few thousand particles using state of the art high performance computers. However, many simulations in physics, such as in astrophysics, and plasma physics, require large numbers of particles with long range forces. Such numerical simulations are referred to as the *N-body problem* and typically involve 10^{12} - 10^{23} particles. The cost of $O(N^2)$ is excessive in most cases, and prohibitive in others.

In certain simulations the ϕ_{long} force can be an *electrical repulsion* force. In the classical force directed layout technique, the force between every pair of unconnected nodes is often modeled as an electrical repulsion [60]. It is this realization that connected the n-body problem to graph drawing [236, 237, 296], so that:

$$\Phi_{total} = \phi_{short} + \phi_{long} + \phi_{external} \tag{4.2}$$

is a generalization of the nonedge forces in the force-directed layout. When using the classical force-directed approach for graph drawing, the same $O(N^2)$ scaling problem occurs.

In other n-body fields, approaches have been developed to reduce the computational effort in calculating the long range ϕ_{long} forces by means of *approximation* or *estimation*. Each and every method that attempts to approximate the ϕ_{long} force introduces errors. The domain of the simulation often dictates the acceptable level of inaccuracy. Typically the accuracy of the approximation is traded off against a faster running time for the simulation. We review two of these methods next.

4.2.1   PIC Codes

One such estimation method is the *particle-in-cell (PIC)* method. A regular mesh is placed over the simulation area and the particles contribute their masses to create a source density. These source densities are used in the estimation of the force field for the mesh. The forces are then interpolated from the mesh to the particle positions. PIC codes have been popular in particle simulations but have two primary drawbacks:

- PIC codes have difficulties dealing with non-uniform particle distributions, that is, where the particle distribution is clustered [222]. In Chapter 5 we show reverse engineering graphs, that exhibit highly non-uniform distribution in their layout. Chapter 6, includes graphs that are suitable for dynamic updates with such PIC code methods.

- PIC codes cannot accurately model the local correlations between particles in boundary cases, which can result in great inaccuracies in this method due to the imposition of an artificial grid which induces those boundaries.

The modified force-directed algorithm of Fruchterman and Reingold [97] follows the PIC approach. Unfortunately, PIC based methods prove unsuitable for approximating nonedge forces in graph drawings which do not exhibit a uniform node distribution. An initial random drawing of any graph will have uniform node distribution, however as the nodes are moved (due to edge and nonedge forces), their distribution becomes increasingly less uniform (in all but the most trivial cases).

Attempts to overcome some of the weaknesses of PIC approaches have employed finer grids to obtain better resolution in denser areas of the simulation and adaptive grid refine-

ment (see [222]); these refinements have not been adopted for graph drawing. A more sophisticated method that attempts to more accurately model local interactions is the *particle-particle particle-mesh* technique (P^3). It is an effective compromise between the possible number of particles and the spatial resolution if the particles are approximately uniformly distributed and relatively low precision is required. In the force-directed method for graph drawing, the required precision of the force calculation might be lowered; however a guarantee that as we iterate the process, each graph drawing has uniformly distributed nodes is unrealistic. As has been noted in other fields, if the particle distribution is clustered, (P^3) is not suitable.

4.2.2   Tree Codes

The simultaneous work of Appel [7], Jernighan and Porter (see [222]) exploit the realization that particles interact only strongly with their nearest neighbours and less detailed information is required to compute the interactions with more distant particles. This fact comes from the typical ϕ_{long} forces used in such simulations; for example gravitational attraction. Based on this realization, the early development focused on creating complex data structures with lists and pointers which attempted to model the "close-neighbourhood" and the "distant groups" for each particle in the simulation. These approaches proved successful but often resulted in large errors being introduced to the simulation due to unphysical grouping of particles. Another problem arose from the complexity of the data structures; these become more "tangled" as the simulation progresses, which makes it difficult to predict run-time or memory requirements *a priori*. Some data structures can require an arbitrary amount of space to hold change and update information. *Tangling* can occur when certain boundary conditions of the simulation are approached, resulting in large amounts of mostly redundant and interrelated additions to the data structure.

Barnes-Hut introduced a scheme to recursively group particles based on an oct-tree decomposition of space [14]. An oct-tree is a three dimensional extension of the quadtree described in Section 3.7.1. This scheme overcomes the tangling problem by rebuilding the decomposition tree for each timestep of the simulation. By rebuilding at each and
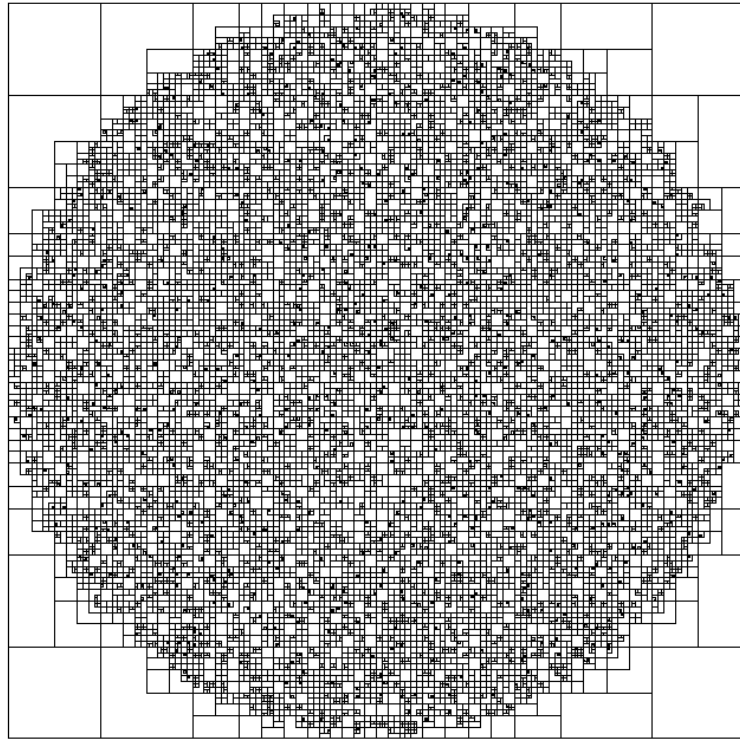
Figure 4.3: A two dimensional space decomposition of 10000 uniformly distributed bodies on a unit disk. Reproduced by courtesy of M. Warren from Warren and Salmon (1992) [290]

every timestep, this scheme ensures that the particle groupings are systematically updated, thereby avoiding the unphysical grouping problem of the prior methods. The previous schemes take nominal $O(n \log n)$ time but the arbitrary nature of the data structures meant this was difficult to analyze and typically not achieved in practice. The Barnes-Hut scheme has a complexity of $O(n \log n)$ which can be rigorously proven [14, 222] under some reasonable distributions of particles (see [222, 275]).

By using a tree data structure, the Barnes-Hut scheme, its derivatives and related schemes are collectively referred to as *tree codes* in the astrophysics literature [5, 7, 14, 28, 43, 222, 275]. Any method used to speed up N-BODY force calculations by the use of a systematic and recursive division of space into a series of *cells* can be considered a *tree code*. The Barnes-Hut scheme is based on an octary-tree code, whereas binary-tree codes were later developed by others such as Press and Benz (see [222]).

Determining the close neighbourhood, called a "near-field", of a particular particle requires $O(N^2)$ operations if calculated directly. The tree data structure provides a way
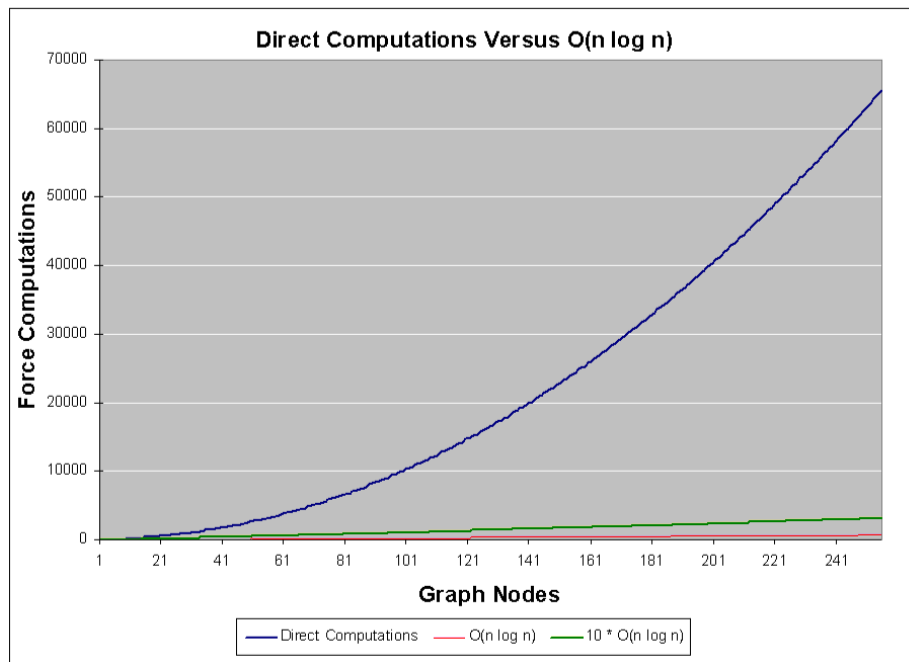
Figure 4.4: A comparison on the number of direct *node-to-node* calculations required upto 256 nodes, compared with a $O(n \log n)$ and with a constant of 10

of determining the degree of closeness without explicitly calculating the distance between every pair of particles. The force on an individual particle x from other particles in the "near-field" is, on average, evaluated by direct particle-to-particle interaction, whereas the force due to more distant particles in the "far-field" is included as a group contribution. In a tree-code method the force ϕ_N of "near-field" particles is computed exactly, while the ϕ_F of "far-field" is approximated.

Tree codes were first developed in the context of galactic simulations involving the study of the collision of galaxies. Tree codes may be based on regular (*quadtree*, *Oct-tree*) and irregular (*Voronoi*) space decompositions. Other trees, such as a nine-way recursive decomposition (*nono-tree*), which is a member of the *orthogonal* n^2 family of two dimensional decompositions, is illustrated in Figure 3.13. Regular trees such as this can be built in linear time.
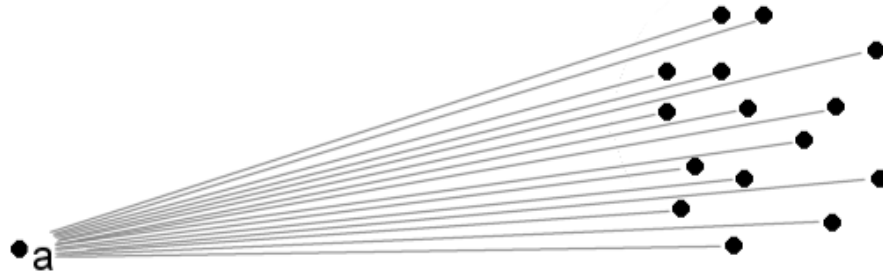
Figure 4.5: Fifteen nonedge force computations involving node **a**

Figure 4.6: A single nonedge computation on **a** with a *pseudonode* representing the 15 nodes, from Figure 4.5

4.3 Force-directed Algorithms by Decomposed Estimation

Classical force-directed methods are based on the direct computation of all *node-to-node* forces, which dramatically limits the number of nodes that these algorithms can handle, as shown in Figure 4.4. In FADE, the *node-to-node* force calculations are approximated based on the notion of *well-separated clusters*, as in other N-BODY based methods [5, 7, 14, 28, 43, 110, 125, 222].

As noted in [110, 222, 275], any force computation errors occur due to round-off, truncation, and discreteness effects, which makes it unreasonable to compute the forces to extremely high precision. Typically, for *particle based simulations*, in for example astrophysics, it is sufficient to have the approximation errors of the tree code be of the same order as these numerical errors. For visualization, this requirement is not the same because our goal instead is to reach a minimum energy state rather than the ensuring the accuracy of the forces. Using a two dimensional electrical repulsion as ϕ_{long}, then the x component

of this nonedge force is defined as follows:

$$\phi_{long.x} = \sum_{(u,v)\in\mathcal{N}x\mathcal{N}} \frac{\alpha}{(d(p_u,p_v))^2} * \frac{u_x - v_x}{d(p_u,p_v)} \tag{4.3}$$

Where $d(p_u,p_v)$ is the euclidean distance between points u and v and α is a weighting factor. The y component of a two dimensional layout is similarly defined. Each pair of nodes must be compared in the nonedge force calculation, in the classical definition. For example, consider just the nonedge force on node a in Figure 4.5. Direct calculation of the nonedge force on node a requires fifteen force computations, one for each (a, v) pair. However, the group of nodes can be clustered and represented as a new pseudonode with weight $w = 15$, as shown in Figure 4.6. If this pseudonode is used, then we save 14 force computations, at the expense of clustering the nodes and the accuracy in the force computation. However, if the node and cluster are "far apart", then the inaccuracies introduced by the approximation are minimal. Finding such clusters in an efficient manner, and determining whether the node and the cluster are sufficiently"far apart", are two keys to the FADE paradigm. As described, this is a *mono-pole* approximation of the forces, due to the single pseudonode representing all the nodes in the cluster. Higher order approximations, that is, p^{th}-*order multipole* expansion approximations, are briefly discussed in Chapter 7.

In the basic FADE method, called FADE2D, the clusters are created by generating a quadtree space decomposition of the locations of the nodes in the layout. This decomposition generates a canonical set of squares. These define a hierarchical geometric clustering (partitioning) of the locations of the nodes. This partitioning in turn induces a graph-theoretic hierarchical clustering of the nodes of the graph, that is (along with implied edges), a hierarchical compound graph.

The process of creating such quadtrees is described in Section 3.7.1. Figure 3.14, shows an example of a quadtree space decomposition for a given graph layout. In FADE, the recursive division of space generated by the quadtree is not used like a grid as in the PIC codes but instead as a bookkeeping structure and inclusion tree. The tree structure provides a systematic way to determine the degree of *closeness* between nodes without explicitly calculating the distance between each pair of nodes.

The root cell of the inclusion tree has a mass equal to the total number of nodes in the graph, that is $mass_{root} = |N|$. The centre of mass of the root cell is the average of both the x and y-locations of the nodes in the graph layout. So, as the tree is constructed, the mass of each cell is set to be the sum of the masses of its daughters. As a result each cell contains a centre of mass, which is the average of the centres of mass of its daughter cells.

It is the distance between this centre of mass and an individual node that is used to determine the closeness between a cluster (cell) and a node. If the centre of a cluster is far enough away, according to a "cell opening criterion", then the *node-to-pseudonode* nonedge force is computed. If the cluster is too close, then its daughter cells are resolved and the process continues. This approach means that the contribution of close nodes is computed directly, as per the classical method, whereas the contribution of distant nodes is taken into account only by including *node-to-pseudonode* forces, representing many node pairs. So overall the force on a node u is

$$f(u) = \sum_{v \in \mathcal{E}(u)} f_{uv} + \sum_{v \in \mathcal{N}(u)} g_{uv} + \sum_{v \in \mathcal{C}(u)} c_{uv} \qquad (4.4)$$

- f_{uv} is due to the edge forces

- g_{uv} is due to the *node-to-node* nonedge forces and $\mathcal{N}(u)$ is the set of close nodes.

- c_{uv} is due to the *node-to-pseudonode* nonedge forces and $\mathcal{C}(u)$ is the set of distant nodes.

Overall, there is a time saving by introducing approximations into the force calculation (as with tree codes). By approximating the forces, FADE introduces an error into the force calculation. Measuring, quantifying, and constraining this error is the subject of considerable study in other N-BODY fields (see [5, 14, 222]) where such errors can cause catastrophic inaccuracies in the simulations [290]. Whereas in FADE, in general, the greater the error in the force calculation, the more iterations of FADE are required to reach a low energy state.

Although we restrict our interest in this Chapter to the inclusion tree of the hierarchical compound graph, it is worth noting that our overall goal is more than just particle sim-

ulation. Unlike particle simulation, FADE addresses the measurement and formation of abstract representations of the data. The hierarchical compound graph allows measures of clustering quality to be formulated, as described in Section 3.4, and provides means to extract précis of the underlying graph. Drawing these précis, allows high level abstract views of the underlying large graphs to be visualized.

4.3.1   Barnes-Hut Cell Opening Criterion

After computing the edge forces, FADE proceeds by determining the nonedge forces on each node of the graph in turn. The *cell opening criterion* is a measure that determines whether a cluster is *far enough* away from the current node to be considered a pseudonode in the nonedge force calculation. The simplest cell opening criterion (also called a *multipole acceptability criteria*), formulated by Barnes and Hut [14] is:

$$\frac{S}{d} \leq \theta \tag{4.5}$$

where S is the width of the cell, d is the Euclidean distance between current node, and the centre of mass of the cell and θ is the "fixed accuracy parameter". The *fixed accuracy parameter* allows the overall runtime of FADE and hence the accuracy of the force computation to be controlled.

The value of θ is chosen *a priori* and not changed during the computation of the forces. For astrophysical simulation, a compromise of $\theta \sim 0.1$ - 1.0, often proves to be a practical choice (see [222]). For FADE the value of θ is typically in the range 0.8 - 1.8, since the edge forces tend to dampen the larger errors introduced in the nonedge force calculation.

Using this cell opening criterion then for each *node-to-pseudonode* criterion check; if $S/d \leq \theta$, then the cell is distant, the internal nodes of the cell are ignored and the *node-to-pseudonode* force, is added to the cumulative force for that node. Otherwise, the cell is resolved into its daughter cells, each of which is recursively examined. Cells are resolved by continuing the decent through the tree until either the opening criteria is satisfied or a leaf node is reached.

Consider the example shown in Figure 4.7. Here a quadtree space decomposition is

(a) Space Decomposition   (b) $\mathcal{BH}$ cell opening criterion
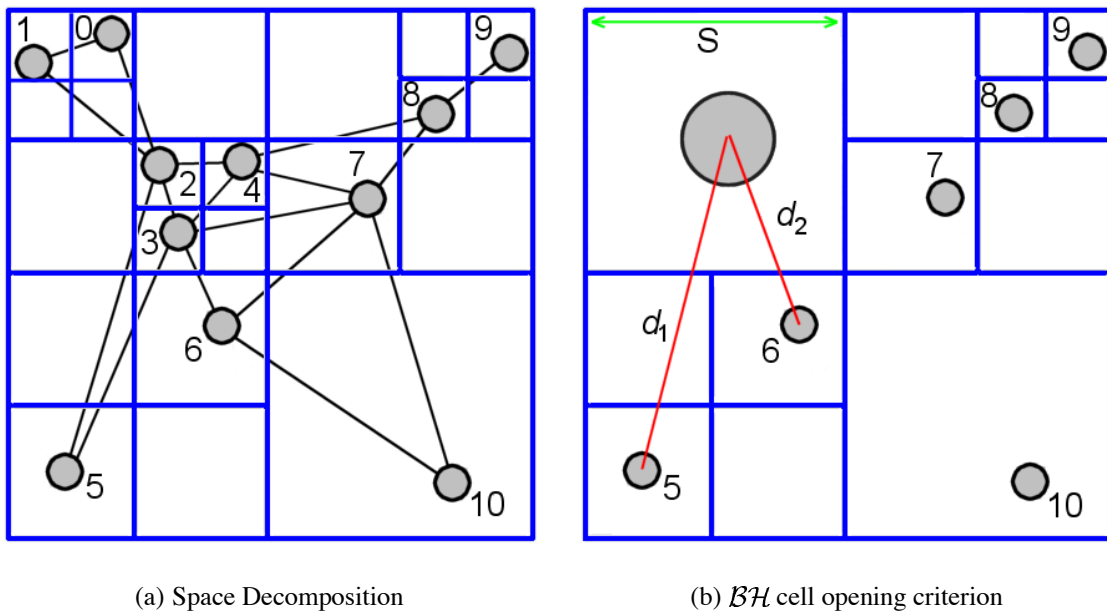
Figure 4.7: Comparing node 5 and 6 with the North-West pseudonode, where $S/d_1 = 0.80$ and $S/d_2 = 1.46$.

used to geometrically cluster the nodes of this graph layout in Figure 4.7(a), as indicated by the quadtree drawing over the graph drawing. The picture in Figure 4.7(b) graphically depicts the use of the Barnes-Hut cell opening criterion for two nodes 5 and 6 with a value of 1.0 for the fixed accuracy parameter θ.

In Figure 4.7(b), the leaf node 5 is compared with the pseudonode for the largest north west cell, which is one of the daughter nodes of the root cell. In the simplest case, the weight of the pseudonode is determined directly by the weight of the cell. The weight of the cell is the number of nodes that it contains, or the number of nodes that its daughter cells contain. The weight of this cell is 5 so the pseudonode has a weight of 5. The value for S is the width of the cell that contains the pseudonode and d_1 is the Euclidean distance from the centre of the node to the centre of the pseudonode. Thus S/d is 0.80 in this case. Given that $\theta = 1.0$, FADE computes an approximate nonedge force between this node and the weighted pseudonode and adds this force to the cumulative nonedge force for node 5. This approximate computation has resulted in a saving of four *node-to-node* force computations.

Again in Figure4.7(b), the leaf node 6 is then compared with the pseudonode for the
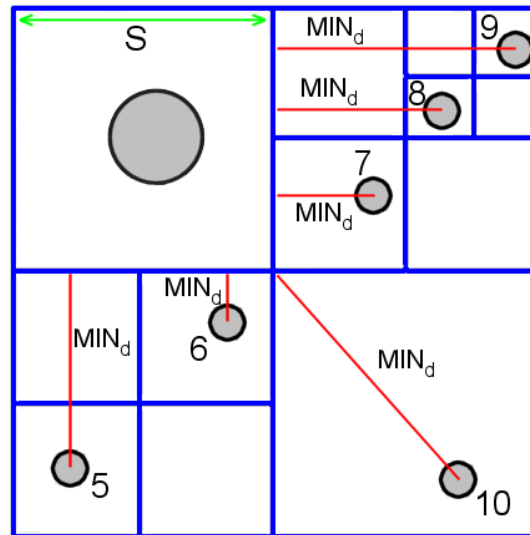
Figure 4.8: The minimum distance from nodes 5,6,7,8,9,10 to the edge of the north-west cell.

largest north west cell. Here d_2 is the distance between node 6 and the pseudonode in the north-west cell. Thus S/d is 1.46 in this case. Here the value does not fulfill the criterion $S/d \leq \theta$, hence the pseudonode is resolved into its daughter nodes, which are the pseudonodes representing (0,1) and (2,3,4). The pseudonode (0,1) does fulfill the criterion and hence the *node-to-pseudonode* force is added to the cumulative nonedge force for node 6. FADE continues by testing $S/d \leq \theta$ for the pseudonode (2,3,4). Again the criterion is not met, so each of the daughter cells are resolved and direct *node-to-node* forces are computed, the results of which are added to the cumulative nonedge force for node 6.

In this example, the nonedge forces due to nodes close by are included as direct *node-to-node* force contributions, whereas more distant nodes are included as group contributions; this example is typical.

The accuracy of a tree code, using this cell opening criterion, has been shown to admit potentially unbounded errors unless $\theta < 1/\sqrt{3}$ [251]. This can be catastrophic in particle simulation, so three alternate cell opening criteria, which address this unbounded error problem, are described below.
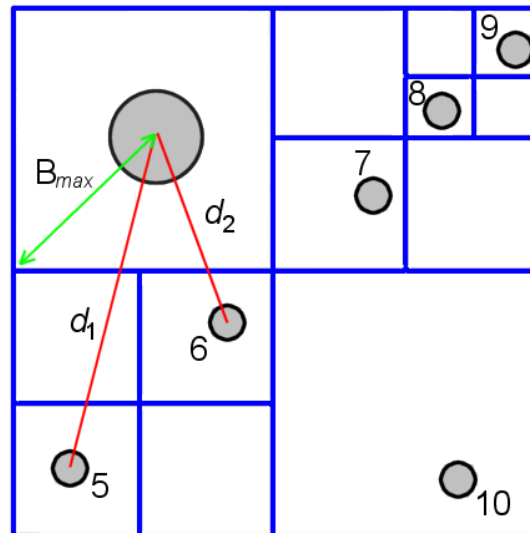
Figure 4.9: The maximum distance from the centre of mass of the north-west cell to the edge of the cell.

4.3.2 Other Cell Opening Criteria

The *minimum distance* cell opening criterion (also called the Min-distance acceptance criteria), formulated by Salmon and Warren [251] aims to avoid gross errors, in certain pathological cases, where the centre of mass is near the edge of the cell. This cell opening criterion is:

$$\frac{S}{MIN_d} \leq \theta. \tag{4.6}$$

where S is the width of the cell and MIN_d is the minimum horizontal or vertical Euclidean distance between the current node and the edge of the cell. This criterion measures the minimum distance from the node to any point in the cell, which is independent of the contents of the cell. Hence, this measure can be applied without computing the exact locations of the nodes inside the cell. Figure 4.8, shows the minimum distance from the centre of each node to the edge of the north-west cell.

The *Bmax* cell opening criterion is motivated by the largest possible error in a distribution of nodes in a cell. For any distribution of nodes in a cell, the largest error is proportional to the magnitude of the single approximation, and a monotonically increasing function of the distance from the centre of mass to the edge of the cell, divided by the

distance to the node [15, 251]. This cell opening criterion is:

$$\frac{B_{max}}{d} \leq \theta. \tag{4.7}$$

Where B_{max} is the minimum distance from the centre of mass of the cell to the edge of the cell. This is illustrated in Figure 4.9.

Finally, other cell opening criteria refinements, such as that by Barnes, have attempted to address the unbounded error problem of relatively off-center mass distributions without introducing extraneous computation (see [15]). Barnes 1995 cell opening criterion, illustrated in Figure 4.10 is:

$$d \geq \frac{S}{\theta} + \delta^2. \tag{4.8}$$

Here δ is the Euclidean distance from the centre of mass of the cell, to the *middle* of the cell. If the centre of mass is reasonably close to the middle of the cell, that is, not close to the edge, then the δ^2 term does not contribute greatly. However, as the centre of mass tends towards the edge of the cell, the δ^2 term contributes much more, thereby avoiding the pathological unbounded error case.

For FADE, the accuracy of the force calculation is not the determining factor for the quality of a graph drawing. Our case studies indicate that, a few very large errors in a given force calculation are typically dampened by the edge forces and are averaged out over many runs of FADE. As such, the unbounded error problem is not as significant as with particle simulations and we typically adopt the Min−distance cell opening criterion in our case studies. In Chapter 7 we suggest possible refinements to our FADE paradigm with the use of other cell opening criterion.

4.3.3 Fixed Accuracy Parameter

For a reasonably uniform distribution of nodes with a nonzero fixed accuracy parameter θ the sum of *node-to-node* and *node-pseudonode* interactions is $O(n \log n)$. This was demonstrated by Hernquist [125] using a simplified geometry, with a single particle in the centre of a homogeneous spherical particle distribution. However, in contrast with this idealized
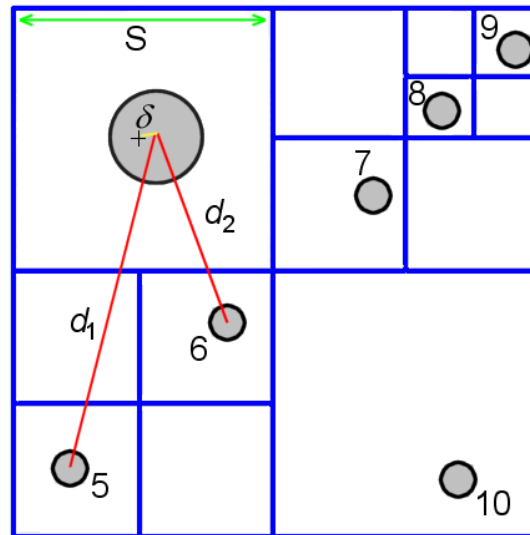
Figure 4.10: The off centre cell opening criterion for the north-west cell.

model, the computation required and the errors introduced also depends strongly on:

- the choice of cell opening criterion

- the value of the fixed accuracy parameter

- the ratio of the average inter particle distance to the width of the root cell.

The reliance on θ is evident. Take, for example, the case of $\theta = 0$; this is obviously equivalent to computing all *node-to-node* nonedge forces directly, as per the classical method of force directed computation. In fact, having $\theta = 0$ is slower than the classical method, as the time required to build and navigate the tree is unnecessary effort. Choosing a very large θ results in very few *node-to-node* computations. This reduction in direct computation produces a very fast run time for FADE since the number of force calculations approaches N. As θ is increased, progressively larger pseudonodes are included in the approximate force calculation for each node, until only the root node is included in the approximation. However, steadily increasing θ results in the force calculation becoming extremely inaccurate.

The trade off between accuracy and speed is very important for FADE, as it allows large graphs to be drawn with relatively inaccurate forces. Studying the relationship between θ

and the inaccuracies introduced requires measures to compare the approximate forces with some benchmarks. A technique for measuring the errors is described in Section 4.3.4.

FADE example

Here we provide an annotated example of how changing the value of the fixed accuracy parameter affects the force computation in our FADE paradigm. In this example, starting in Figure 4.11, FADE computes the nonedge forces on the node ρ, indicated by a star. Here the edge forces are computed first, so we omit the edges from further drawings, to emphasize the *node-to-node* and *node-to-pseudo-node* force contributions. Figure 4.11 shows a graphical depiction of the interaction list, that is, the nodes and pseudonodes that contribute to nonedge force calculation for ρ when $\theta = 0$. This is the classical force directed case, with only *node-to-node* force calculations, and every node in the graph is included in the interaction list for ρ.

As the value for θ increases, the number and relative weight of the pseudonodes in the interaction list also increase. The weight of a particular pseudonode is depicted as the area of the pseudonode image. This is illustrated in Figures 4.12 to 4.16. This increase in *node-to-pseudonode* interactions causes a resultant decrease in the number of *node-to-node* interactions thereby reducing the run-time of FADE. It is also important to note that, on average, the largest pseudonodes first appear on the periphery of the interaction region of ρ. This is the reason for the previously stated observation that, on average, nodes interact by direct *node-to-node* nonedge forces with close-by nodes, whereas approximate force contributions are included for distant groups of nodes. The precise determination of *close* or *distant* nodes is dependent on the choice of θ and the distribution of the nodes.

FADE treats each pseudonode as a single point mass that approximates a set of nodes within that cell. This approach follows the *monopole* tree code methods from particle simulation. However, it is also possible to increase the accuracy of the FADE paradigm without increasing the size of the interaction list for a given θ. *Multipole* moments treat the pseudonodes as having multiple centres of mass, rather than treating them as single point masses. *Di-pole* and *quad-pole* methods are popular refinements to the basic tree code method. Clearly, if the multipole expansion is carried out to a high enough order, it

contains the total information of the node distribution in the cell. As noted earlier, however, accuracy in the force calculation is not the driving factor for the use of these methods in force directed layout. As such, we leave further discussion of these multipole methods to Chapter 7.
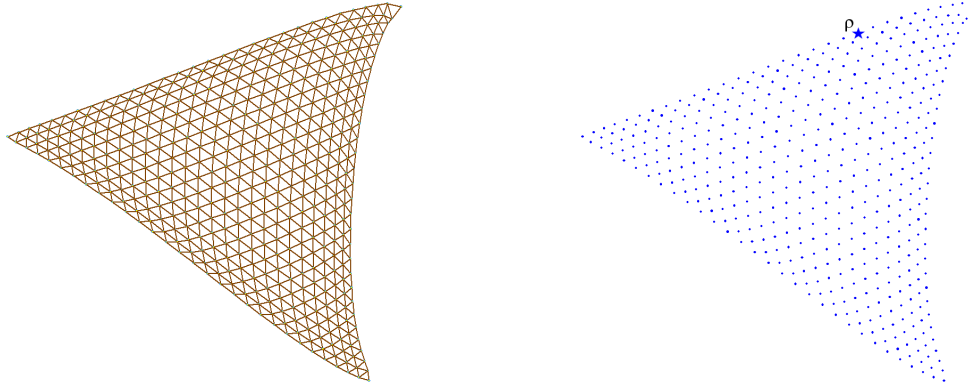
Figure 4.11: *node-to-node* interaction list, with and without edges, for node ρ with $\theta = 0$, that is, the interaction list as in the classical force-directed method
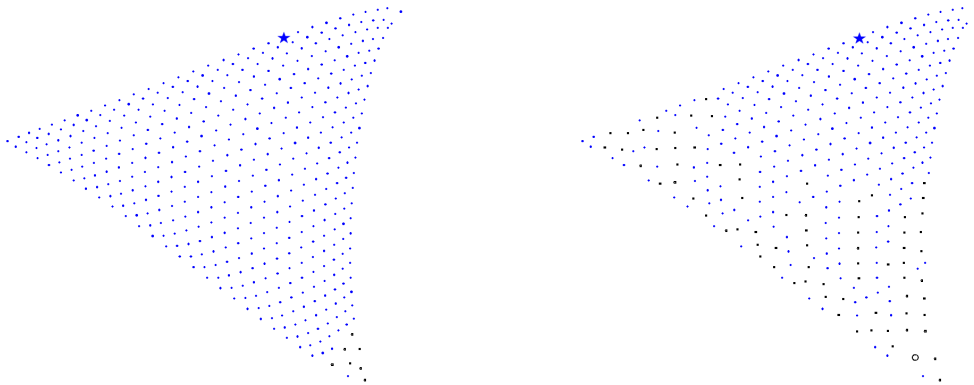
Figure 4.12: Interaction list for node ρ with $\theta = 0.05$ and $\theta = 0.1$
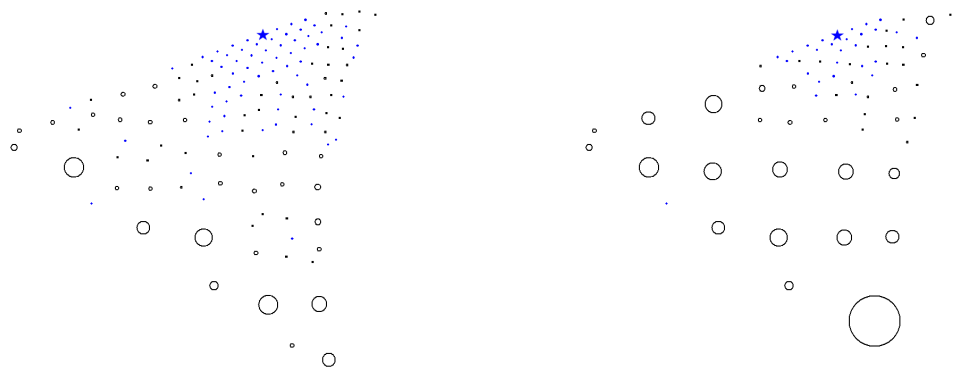
Figure 4.13: Interaction list for node ρ with $\theta = 0.3$ and $\theta = 0.5$
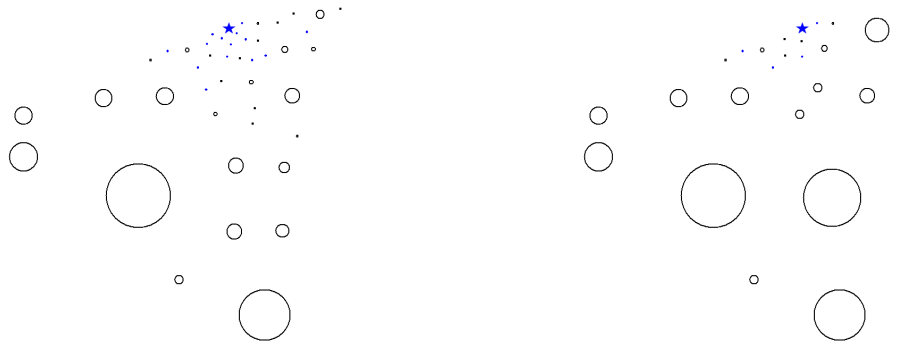
Figure 4.14: Interaction list for node ρ with $\theta = 0.7$ and $\theta = 1.0$
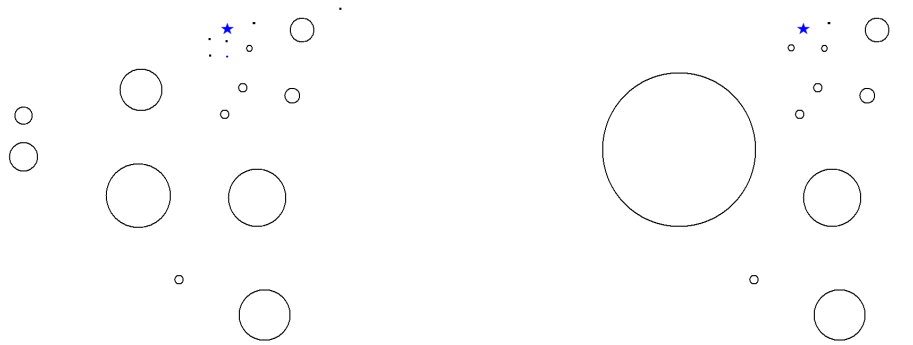
Figure 4.15: Interaction list for node ρ with $\theta = 1.2$ and $\theta = 1.5$
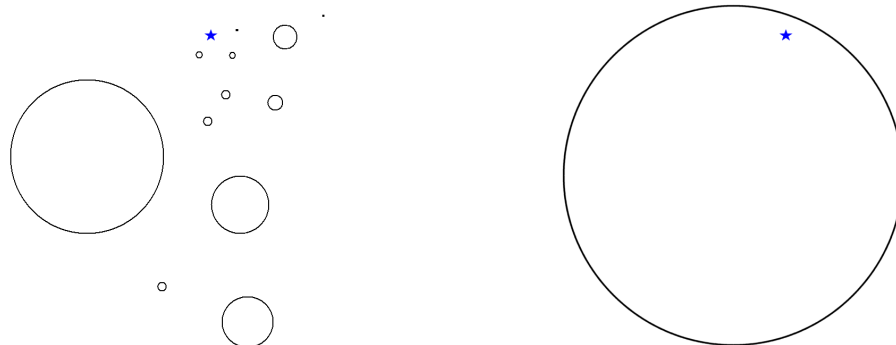
Figure 4.16: Interaction list for node ρ with $\theta = 2.0$ and $\theta = 4.0$

4.3.4 Error Measure

Although accuracy in the force computation is not the determining factor in the quality of the final drawing produced, it is an important factor that must be addressed and measured in our case studies. The question of trading off accuracy against performance is related to the initial layout, the structure of the graph, the choice of θ, the shape of the decomposition, and the density of the node distribution. Extensive analysis of tree-code performance and accuracy, in the particle simulation context, has been performed since the development of tree based methods [5, 28, 124, 125, 222]. Typically these analyses have focused on how *accurately* a given tree-code calculates the forces, since this is often the limiting factor for good energy conservation. These studies have resulted in error measures that chart how accurately a tree code approximates the true path of a given particle from start state to end state over a given number of timesteps.

FADE uses the same class of error measure as in the classical N-BODY simulations. To give a comparative analysis, only the nonedge forces are factored into the error calculation here and in our case studies. In reality, the edge forces tend to dampen out the errors in the nonedge forces, resulting in very accurate overall forces, even with relatively high nonedge force computation errors. In FADE, the error measure is used to see how closely the nodes of the layout follow the path as prescribed by the nonedge forces in the direct *node-to-node* computation method. This error measure gives a value for the global conservation rather than an absolute accuracy measure. The error measure (ϵ) is given in terms of a three dimensional layout (the force component in the z direction, can be safely ignored for a two dimensional error measure). More precisely the error measure ϵ is defined as follows:

$$\epsilon = \frac{\epsilon_x + \epsilon_y + \epsilon_z}{3} \tag{4.9}$$

Where for $k = x, y, z$

$$\epsilon_k = \left\{ \frac{\sum_{i=1}^{N} \left(F_{ki}^{tree} - F_{ki}^{direct} \right)^2}{\sum_{i=1}^{N} \left(F_{ki}^{tree} \right)^2} \right\}^{\frac{1}{2}} \tag{4.10}$$

where the x,y,z components of the approximated forces on a node i are $F_{xi}^{tree}, F_{yi}^{tree}, F_{zi}^{tree}$ and the x,y,z components of the directly summed forces on i are $F_{xi}^{direct}, F_{yi}^{direct}, F_{zi}^{direct}$.

The results of applying this error measure, to a set of example graphs, which have been given a random layout in a fixed bounded area are shown in Table 4.1. Here the error is averaged over a selection of runs of a basic implementation of FADE. In this example, we randomly select $\sqrt{i/2}$ iterations from the first $i = 700$ iterations of FADE2D; the algorithm described in Section 4.4. Table 4.2 shows the total number μ of force calculations (both *node-to-node* and *node-to-pseudonode*), averaged over the same iterations, as θ increases. Table 4.3 shows both the error measures and the force calculations μ for the opening iteration of FADE, when the nodes are initially randomly placed. The same trends are evident in all tables. Some remarks on this data should be made:

- The ϵ measure shows that as θ increases, the error climbs, as predicted by other N-BODY work.

- An increase in density of nodes in the unit square in the initial drawing causes the errors to steadily increase, as the optimal inter-node distance is based on a uniform spring length, regardless of node density.

- An initial random layout, in a scaled unit square, provides a uniform distribution of nodes which achieves the $O(n \log n)$ time as shown by Hernquist [124, 125]; this is shown in Table 4.3.

One can also see, from the difference between Tables 4.2 and 4.3, that FADE performs better on the initial random distribution than subsequent iterations. This is primarily due to two factors.

- The runtime of FADE is dominated by the force calculation but is not the same for every iteration. FADE rebuilds the tree *ab-initio* before each iteration which takes a different amount of effort depending on the distribution of the node locations.

- If the graph has natural clusters, then as the drawing improves these underlying structures become apparent and hence the uniformity in the layout decreases.

Nodes	$\theta = .01$	$\theta = .05$	$\theta = .1$	$\theta = .3$
1277	.00000000004	.0000008380	.0000138081	.00047509631
1832	.00000000038	.0000009288	.0000220585	.00048515007
2487	.00000000032	.0000014832	.0000232423	.00053358670
3242	.00000000047	.0000016704	.0000328129	.00063418773
5052	.00000000072	.0000026256	.0000319591	.00066698126
7262	.00000000126	.0000041525	.0000379348	.00068908789
9872	.00000000234	.0000048062	.0000449909	.00074681910
12092	.00000000259	.0000044238	.0000487464	.00075540835
Nodes	$\theta = .5$	$\theta = 0.7$	$\theta = 1.0$	$\theta = 1.2$
1277	.0019500795	.00539801978	.0150509131	.025228532713
1832	.0019884802	.0048182771	.0143546497	.026913635200
2487	.0021842746	.00505085568	.0153926954	.026407794374
3242	.0022511564	.00578494003	.0166393935	.029829451384
5052	.0025214306	.00678366025	.0180891907	.034157615132
7262	.0025843103	.00651497615	.0193562692	.035122402799
9872	.0027210468	.00680969862	.0198439982	.037554443325
12092	.0026423973	.00676360044	.0190950007	.036336674694
Nodes	$\theta = 1.5$	$\theta = 2.0$	$\theta = 3.0$	$\theta = 4.0$
1277	.04870902448	.09047355384	.15279595472	.1984136239
1832	.05031801678	.09611990368	.14670438657	.183907956324
2487	.05278841594	.10133313388	.16345486364	.22532728982
3242	.06925755366	.11768116360	.21565771798	.27166379940
5052	.06653620606	.12022535238	.20829714237	.27506811254
7262	.07176602972	.14557621557	.23911949148	.32061095159
9872	.07563862996	.15569051335	.28185172218	.40812468249
12092	.07631118917	.16711961048	.29151599631	.39025153426

Table 4.1: Errors in the nonedge force computation FADE2D different values of θ

Nodes	$\theta = .01$	$\theta = .05$	$\theta = .1$	$\theta = .3$
1277	1626234	1566126	1201892	423883
1832	3349876	3172705	2196379	646963
2487	6175171	5689191	4003452	1228431
3242	10497905	9976598	7122082	2018581
5052	30283941	29384859	11939403	4039201
7262	52710204	45329015	28552538	6583002
9872	97383575	81553666	49370679	10293893
12092	146095891	122938983	73202939	14354588

Nodes	$\theta = .5$	$\theta = 0.7$	$\theta = 1.0$	$\theta = 1.2$
1277	215991	134256	80293	61401
1832	318063	193133	113807	86004
2487	576627	338519	190656	142405
3242	900897	507110	278911	207081
5052	1928381	902910	569929	440039
7262	2710759	1463403	769205	558272
9872	4051174	2143683	1092473	779698
12092	5614529	2922024	1472483	1050497

Nodes	$\theta = 1.5$	$\theta = 2.0$	$\theta = 3.0$	$\theta = 4.0$
1277	44040	28330	15106	10126
1832	60289	36887	16960	9264
2487	100390	61135	30182	19772
3242	142071	85264	41651	27893
5052	293882	189293	801920	69930
7262	376987	219244	108758	71079
9872	513238	288505	130067	79922
12092	691404	384630	175169	111860

Table 4.2: Averaged sum of *node-to-pseudonode* and *node-to-node* interactions for various graphs with varying fixed accuracy parameter of θ, with errors as shown in Table 4.1

Nodes	μ	$\theta = 0.0$	μ	$\theta = .01$	μ	$\theta = .05$
1277	1629452	.0	1626900	.00000000	1618085	.000000060
3830	14665070	.0	14657297	.000000001	14416952	.000000062
5052	25517652	.0	25506310	.000000001	24902915	.000000077
12092	146204372	.0	146083109	.000000004	138336227	.00000013

Nodes	μ	$\theta = .1$	μ	$\theta = .3$	μ	$\theta = .5$
1277	1548402	.0000001	834649	.00000027	390064	.00000028
3830	12947392	.0000001	4382696	.00000026	1631375	.00000027
5052	21703741	.0000002	6439792	.00000032	2292324	.00000032
12092	107119059	.0000002	21158111	.00000028	6751967	.00000029

Nodes	μ	$\theta = 0.7$	μ	$\theta = 1.0$	μ	$\theta = 1.2$
1277	196752	.00000028	97337	.000000288	56756	.00000028
3830	755402	.00000027	342516	.000000273	196829	.00000027
5052	1051930	.00000033	478822	.000000330	277517	.00000033
12092	2995143	.00000029	1305468	.000000291	755903	.00000029

Nodes	μ	$\theta = 1.5$	μ	$\theta = 2.0$	μ	$\theta = 3.0$
1277	25769	.00000029	12711	.000000293	10185	.00000029
3830	84656	.00000027	38881	.000000279	30800	.00000028
5052	116414	.00000033	51241	.000000335	40536	.00000033
12092	308811	.00000029	124983	.000000297	97034	.00000029

Nodes	μ	$\theta = 4.0$	μ	$\theta = 5.0$	μ	$\theta = 6.0$
1277	9130	.00000028	7611	.000000271	6071	.00000025
3830	27580	.00000027	22946	.000000256	18487	.00000023
5052	36824	.00000033	30545	.000000316	24420	.00000030
12092	88860	.00000029	73775	.000000274	58893	.00000025

Nodes	μ	$\theta = 7.0$	μ	$\theta = 8.0$	μ	$\theta = 9.0$
1277	4853	.00000026	4034	.000000301	3348	.00000036
3830	14728	.00000024	11830	.000000291	9870	.00000035
5052	19702	.00000030	16020	.000000339	13283	.00000039
12092	46566	.00000026	38047	.000000301	31950	.00000036

Nodes	μ	$\theta = 10.0$	μ	$\theta = 11.0$	μ	$\theta = 12.0$
1277	2865	.00000044	2515	.00000052	2249	.00000060
3830	8589	.00000042	7483	.00000050	6790	.00000058
5052	11491	.00000046	10063	.00000054	9139	.00000060
12092	27421	.00000043	24173	.00000051	21653	.00000059

Table 4.3: The count of *node-to-pseudo-node* and *node-to-node* $= \mu$ interactions and the ϵ error for the first iteration of FADE2D for various graphs with varying fixed accuracy parameter
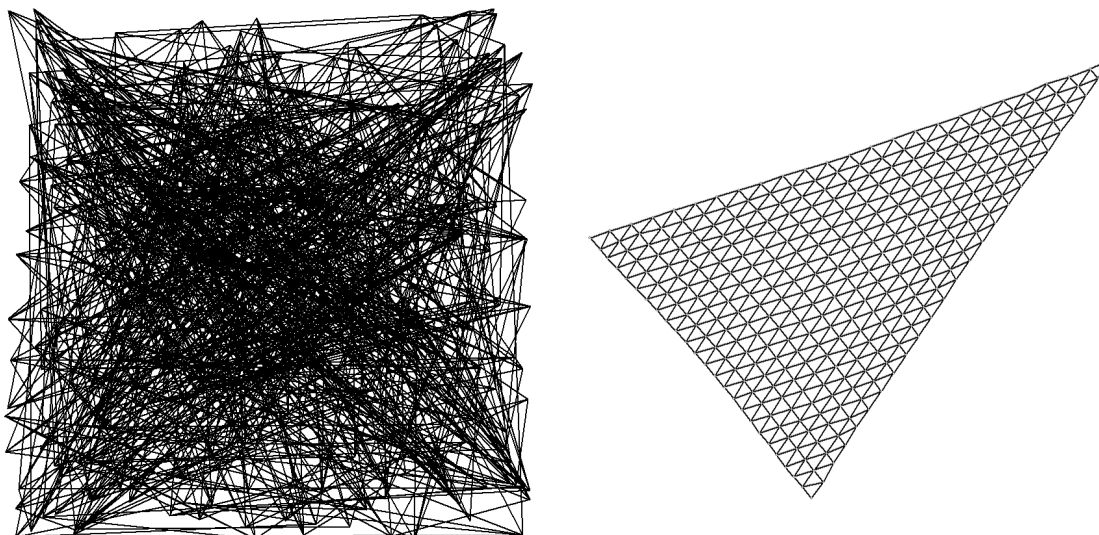
Figure 4.17: Initial random layout of a 326 node triangular-mesh and its final drawing

4.4 FADE2D

FADE2D is our basic drawing algorithm for undirected graphs, in two dimensions, using a simple force model and a quadtree space decomposition to create the hierarchical compound graph. The FADE2D algorithm directly computes the *node-to-node* forces for edges and approximates the nonedge forces using a tree-code approach, where the tree is reconstructed *ab-initio* for every new iteration of FADE2D. Roughly speaking, FADE follows the general schema described in **Algorithm 1**. In this section we describe in more detail how FADE2D works.

For FADE2D, the initial layout merely assigns random locations to the nodes. This is a simple technique to give the graph an initial layout. This layout can then be rendered to produce a graph drawing (albeit a very bad one by almost every aesthetic and clustering measure). In Section 4.6, we suggest an alternate initial layout strategy which is suitable for the graphs in both of our case studies. For now, we restrict our interest to the initial random placement as it provides a consistent starting quality.

For example, Figure 4.17 shows the opening and final drawing of a triangular mesh of 326 nodes. Initially the 326 nodes are assigned random x- and y-locations and the route for the edge is defined by its start and end node locations. This layout is then rendered in the image on the left of Figure 4.17.

Algorithm 2 describes the computation of the edge forces in FADE2D. The damping parameter $\mathcal{T}$ decreases the value and hence effect of the attactive force. If the damping parameter is 2 then the force is directly calculated and applied to each node.

Algorithm 2 Compute 2D Edge Forces

Input: EdgeSet $\mathcal{E}$
Input: $\mathcal{T}$, which is a damping parameter
Require: $\mathcal{T} \geq 2$
begin
  for $i = 1$ to $|\mathcal{E}|$ **do**
    $f \Leftarrow 0$
    $\mathcal{K} \Leftarrow \mathcal{E}[i].resilience$
    $\mathcal{L} \Leftarrow \mathcal{E}[i].idealLength$
    $V_x \Leftarrow \mathcal{E}[i].endNode.x - \mathcal{E}[i].startNode.x$
    $V_y \Leftarrow \mathcal{E}[i].endNode.y - \mathcal{E}[i].startNode.y$
    $d \Leftarrow \sqrt{V_x^2 + V_y^2}$
    if $d \neq 0$ **then**
      $f \Leftarrow \dfrac{(\mathcal{L} - d) * \mathcal{K}}{\mathcal{T}}$
    end if
    $f_x \Leftarrow f * V_x/d$
    $f_y \Leftarrow f * V_y/d$
    if $\mathcal{E}[i]$ is not fixed **then**
      $\mathcal{E}[i].endNode.x \Leftarrow \mathcal{E}[i].endNode.x + f_x$
      $\mathcal{E}[i].endNode.y \Leftarrow \mathcal{E}[i].endNode.y + f_y$
      $\mathcal{E}[i].startNode.x \Leftarrow \mathcal{E}[i].startNode.x - f_x$
      $\mathcal{E}[i].startNode.y \Leftarrow \mathcal{E}[i].startNode.y - f_y$
    end if
  end for
end

Algorithm 3 and **Algorithm 4** describe how a regular geometric clustering, which includes quadtrees and nonotrees, is constructed in a top-down manner. Here the nodes are simply added to the tree one at a time. Note that an agglomerative tree construction may give some performance improvement; however, tree construction takes only a tiny fraction of the total and the improvement would be negligible.

Algorithm 4 describes the process for creating twigs and leaves of the inclusion tree, regardless of orthogonal decomposition used. The node and the current cell (the root) are passed into this method. If the cell is empty, then it is updated to contain the details for this

---

Algorithm 3 Build 2D Orthogonal Geometric Clustering

Input: Nodes $\mathcal{N}$, Clustering Arity Width $\mathcal{A}$
Input: Bounding Box $\mathcal{BB}$ of $\mathcal{N}$
Require: $\mathcal{N} \neq \emptyset$
Require: $\mathcal{A}$ is a positive integer value
  $root \Leftarrow \text{newCell}(\mathcal{BB}, \mathcal{A})$
  for $i = 1$ to $|\mathcal{N}|$ **do**
    $\text{addNode}(\mathcal{N}[\text{i}], \text{root}, \mathcal{A})$
  end for

---

node. If the cell is full, then the node details it currently contains are removed and stored and the cell is split. This node, and the original node, are then reintroduced at this cell's parent level by a recursive call to this method. Finally, if the node is spilt then the centre of mass of this cell is updated and its weight incremented. References to the daughter cells are stored in an array and the index to this array is determined by the `indexOfDaughter` method. For example, in a quadtree, where there are a maximum of four daughter cells, this method determines whether the node location is in the *north-west*, *south-west*, *north-east* or *south-east* quadrant and selects the index accordingly. If there is no daughter cell for the current index, then one is created. Once a reference to the correct daughter cell is determined, then this method is recursively called with the node and this daughter cell.

The crux of the FADE2D method is the "Compute Approximate Nonedge Force" step, shown in **Algorithm 5**. It should be noted that $\mathcal{K}$ in **Algorithm 5** is the same class of weighting factor as the edge resilience $\mathcal{K}$ in **Algorithm 2**. These weights are typically set at 1 for direct force calculation. In an interactive setting these values can be altered, likwise when the value of θ is large then the overall non-edge force calculation is inaccurate so the edge forces can be made to dominate. If non-edge forces between nodes with an edge between them are not computed, then the the attractive forces on an edge should balance when the edge is at it's ideal length (if $\mathcal{K}$ is 1). In Section 3.7.1 we described how the internal nodes of the inclusion tree structure of the hierarchical compound graph, the *twigs*, contain equivalents for the centre of mass and weight. The weight is typically the number of nodes that the cell contains. Once these have been defined for each twig node, then they can be used by the force approximation routine.

FADE2D has been implemented in a prototype graph drawing tool, for the layout, draw-

---

Algorithm 4 addNode to a Geometric Clustering

Input: $node, cell$

Input: Clustering Arity Width $\mathcal{A}$

Require: $cell \neq \emptyset$

Ensure: There are at most $\mathcal{A}^2$ daughters of this cell

begin

  if cell is empty **then**

    $cell.contents \Leftarrow node$

    $cell.x \Leftarrow node.x$

    $cell.y \Leftarrow node.y$

    $cell.count \Leftarrow 1$

    $node.parent \Leftarrow cell$

    $cell.status \Leftarrow full$

  else if cell is full **then**

    $cell.status \Leftarrow split$

    $\mathcal{P} \Leftarrow cell.parent$

    $\mathcal{LN} \Leftarrow cell.contents$

    $\mathcal{LN}.parent \Leftarrow \emptyset$

    $cell.count \Leftarrow 0$

    $cell.x \Leftarrow 0$

    $cell.y \Leftarrow 0$

    $addNode(node, \mathcal{P}, \mathcal{A})$

    $addNode(\mathcal{LN}, \mathcal{P}, \mathcal{A})$

  else {Cell is split}

    $cell.x \Leftarrow \dfrac{(cell.count * cell.x) + node.x}{cell.count + 1}$

    $cell.y \Leftarrow \dfrac{(cell.count * cell.y) + node.y}{cell.count + 1}$

    $cell.count \Leftarrow cell.count + 1$

    $index \Leftarrow$ indexOfDaughter(cell,node,$\mathcal{A}$)

    if $cell.daughters[index] = \emptyset$ **then**

      $\mathcal{BB} \Leftarrow$ boundBoxOfDaughter(cell,node,$\mathcal{A}$)

      $cell.daughters[index] \Leftarrow$ newCell($\mathcal{BB}, \mathcal{A}$)

    end if

    addNode($node$, cell.daughters[index], $\mathcal{A}$)

  end if

end

Algorithm 5 Compute 2D Approximate Node Forces: 2daproxforce

Input: $node, cell$, Force Intensity $\mathcal{K}$,
Input: Criterion Type, Cell Opening Criterion θ
Require: $cell \neq \emptyset$
begin
$\quad V_x \Leftarrow node.x - cell.x$
$\quad V_y \Leftarrow node.y - cell.y$
$\quad d \Leftarrow \sqrt{V_x{}^2 + V_y{}^2}$
$\quad$ **if** $d \neq 0$ **then**
$\quad\quad$ **if** Criterion Type $= \mathcal{BH}$ **then**
$\quad\quad\quad \mathcal{M} \Leftarrow \dfrac{cell.width}{d}$
$\quad\quad$ **else if** Criterion Type $= \mathcal{MAX}$ **then**
$\quad\quad\quad \mathcal{M} \Leftarrow \dfrac{cell.max}{d}$
$\quad\quad$ **else if** Criterion Type $= \mathcal{NEWBH}$ **then**
$\quad\quad\quad C_x \Leftarrow cell.x - cell.cx$
$\quad\quad\quad C_y \Leftarrow cell.y - cell.cy$
$\quad\quad\quad \delta^2 \Leftarrow C_x{}^2 + C_y{}^2$
$\quad\quad\quad \mathcal{M} \Leftarrow \dfrac{cell.width}{d - \delta^2}$
$\quad\quad$ **else** {Criterion Type is Orthogonal Distance}
$\quad\quad\quad$ **if** $|V_x| > |V_y|$ **then**
$\quad\quad\quad\quad \mathcal{M} \Leftarrow \dfrac{cell.width}{|V_x| - 0.5 * cell.width}$
$\quad\quad\quad$ **else** {Use Horizontal Distance}
$\quad\quad\quad\quad \mathcal{M} \Leftarrow \dfrac{cell.width}{|V_y| - 0.5 * cell.width}$
$\quad\quad\quad$ **end if**
$\quad\quad$ **end if**
$\quad\quad$ **if** $\mathcal{M} \leq \theta$ **OR** cell is a leaf **then** {Use Approximation}
$\quad\quad\quad f \Leftarrow cell.count * \mathcal{K}/d^2$
$\quad\quad\quad node : dx \Leftarrow node.x + (f * V_x)$
$\quad\quad\quad node : dy \Leftarrow node.y + (f * V_y)$
$\quad\quad$ **else** {Resolve for the daughter cells}
$\quad\quad\quad$ **for** $i = 1$ to $|cell.daughters|$ **do**
$\quad\quad\quad\quad$ 2daproxforce (node, cell.daughters[i], $\mathcal{K}$, Criterion Type, θ)
$\quad\quad\quad$ **end for**
$\quad\quad$ **end if**
$\quad$ **end if**
end

ing and visual précis of a variety of graphs and hierarchical compound graphs from different domains. Application case studies for FADE2D, along with performance measures, error measure, aesthetic measures, and clustering measures (described previously) are detailed in Chapter 5 and Chapter 6. These case studies include large software engineering graphs extracted from a code analysis and matrix market graphs, from a range of application domains.

For FADE2D with the $\mathcal{BH}$ cell opening criterion, the effect of having different values for the fixed accuracy parameter θ is shown in Figures 4.11 to 4.16. Of course the resultant inaccuracies of this approximation method also increase; see for example Table 4.1 and Table 4.3.

A visual précis of `qh1484` (from Chapter 6) is shown in Figure 4.18, which is typical of the high level horizon drawings that the FADE paradigm constructs. A drawing of `add32` produced by FADE2D, a graph of 4960 nodes and 9462 from the Graph Partitioning Archive [283, 284], is shown in Figure 4.19.

4.5 FADE3D

FADE3D is a straightforward three dimensional extension to FADE2D. The original treecode method of Barnes-Hut [14] was designed for astrophysical simulations, of planets, stars, and other heavenly bodies in the real world. This required modeling a three dimensional environment which they achieved with an extension of the quadtree space decomposition called an "octtree". An *octtree* is a recursive decomposition of space into eight *octants* instead of four quadrants as in the two dimensional quadtree. As is shown in Figures 4.20, 4.21, 4.22 and 4.23.

Figures 4.20, 4.21 and 4.22, show the top three levels of the space decomposition of a drawing of `nos7`, from the case study in Chapter 6. The first figure, the root cell, shows the smallest cube enclosing all the nodes of the graph drawing. The second figure, demonstrates the eight way division that occurs in a octtree decomposition of space. Figure 4.22 represents the decomposition to the second level of granularity in the inclusion tree. Figure 4.23 is a representation of the entire octtree of a space decomposition of `bfw782a`,
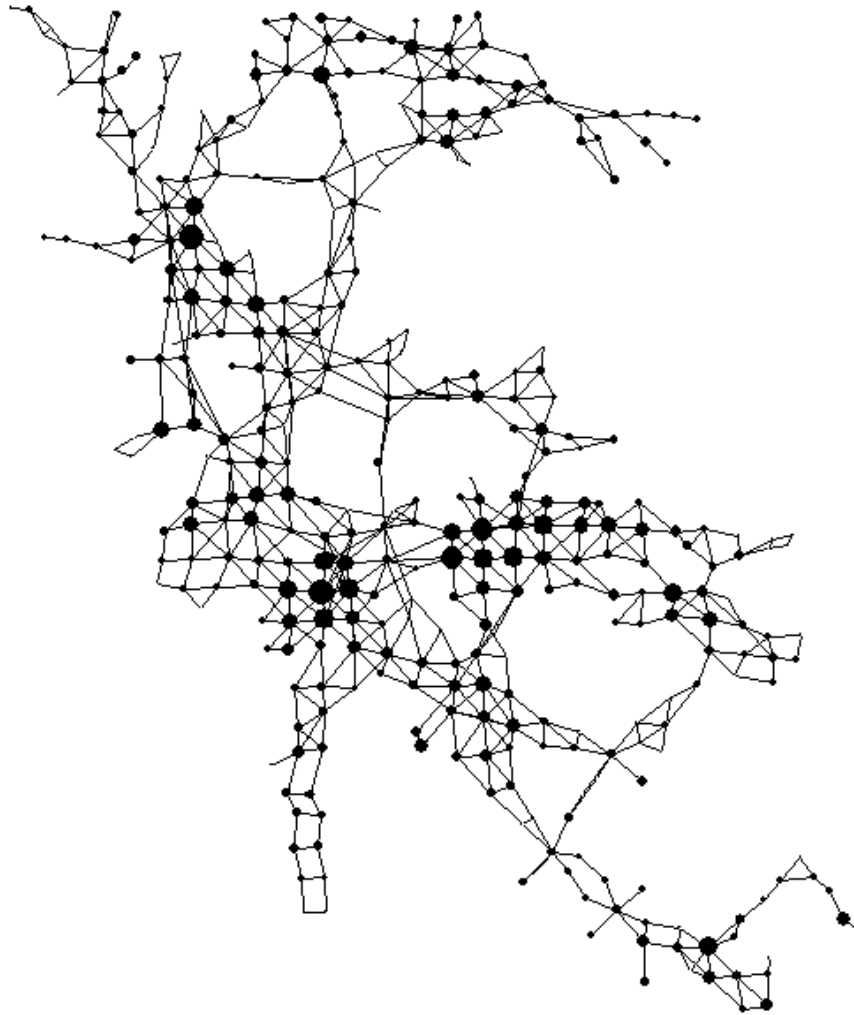
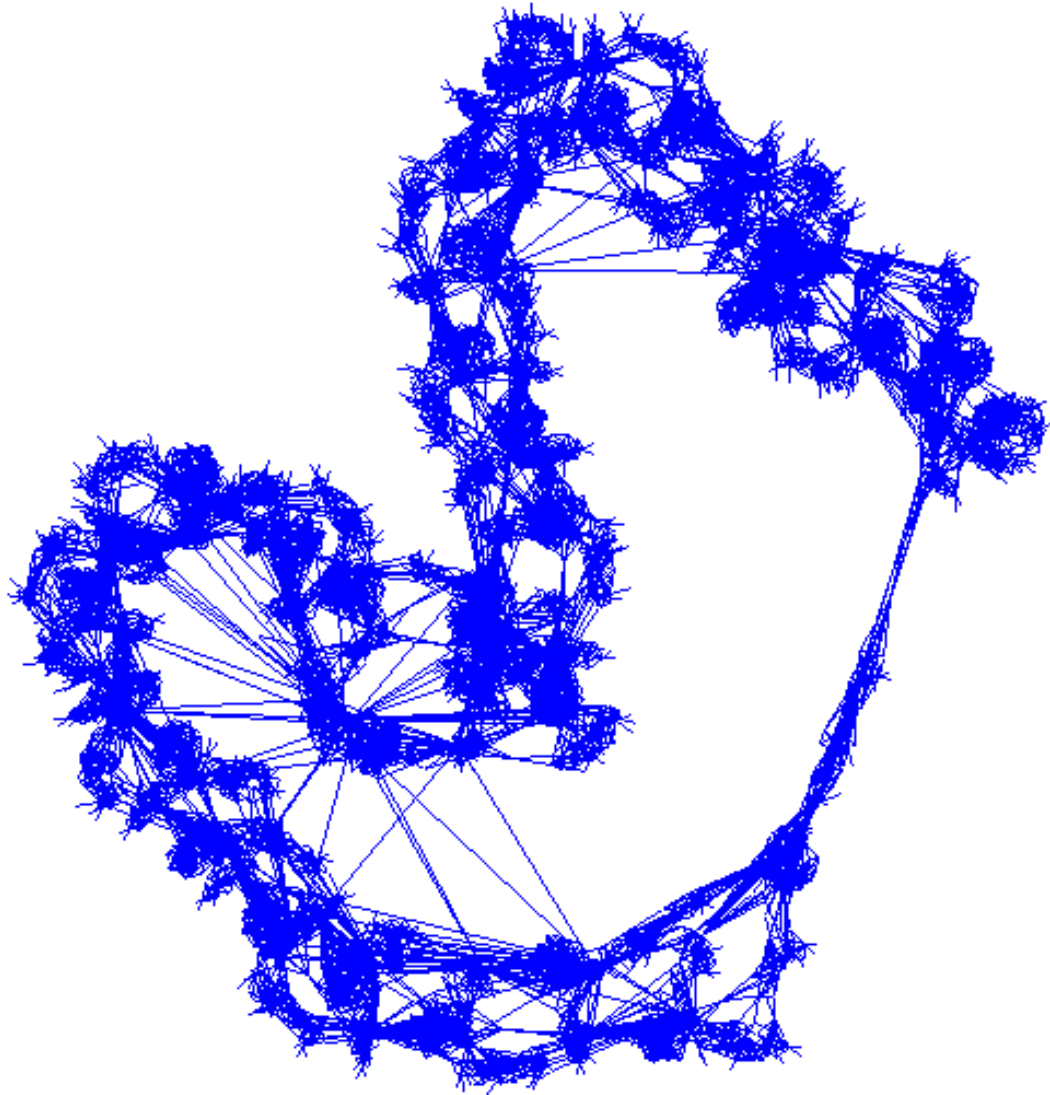**Figure 4.18:** A visual précis of `qh1484`, from the matrix market (see Chapter 6), drawn with FADE2D

Figure 4.19: FADE2D drawing of `add32` (4960 nodes) from the Graph Partitioning Archive [283, 284]
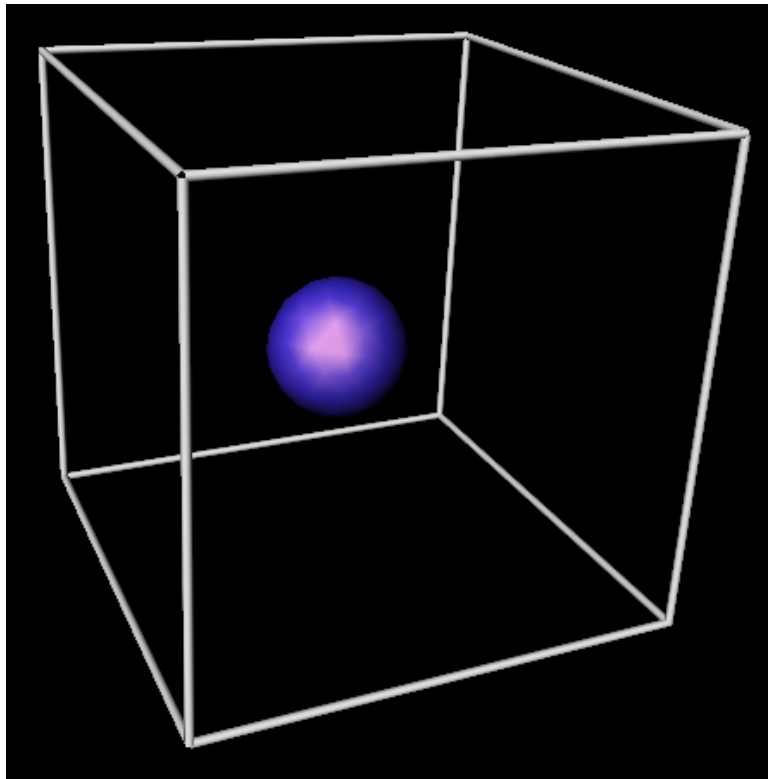
Figure 4.20: Root cell of octtree decomposition of `nos7`

which is a graph with two large disconnected sub-graphs.

The FADE paradigm is based on the hierarchical compound graph model, which is independent of any particular decomposition method; thus it is independent of dimension. The FADE3D method uses a cubical division of space for graph layout, clustering, and the generation of visual précis. The primary advantage of the three dimensional variant of the FADE2D algorithm, as with all three dimensional force directed placement, is that the extra dimension allows greater freedom in placing the nodes. This can result in a more natural representation of the graph and hence the information that it represents.

Figure 4.24, shows the results of applying both FADE2D and FADE3D to a graph from the Matrix Market case study in Chapter 6. The picture in the upper left corner of Figure 4.24 is produced by FADE2D, described in the previous section. The rest of the drawings in Figure 4.24 are produced by FADE3D. We include the three dimensional model and a video simulation, for this data, in Appendix A.

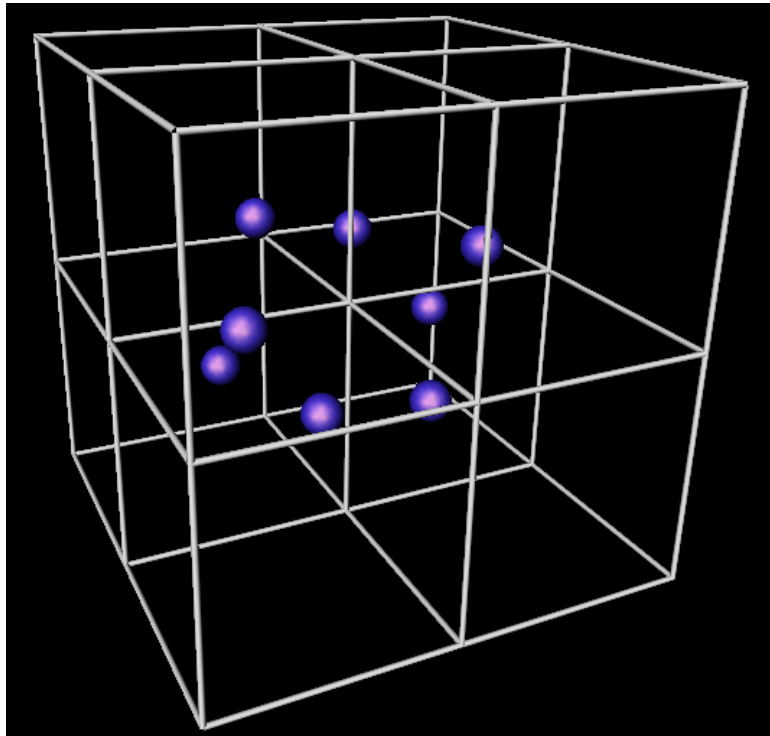The formulation of FADE3D differs little from its two dimensional counterpart. How-

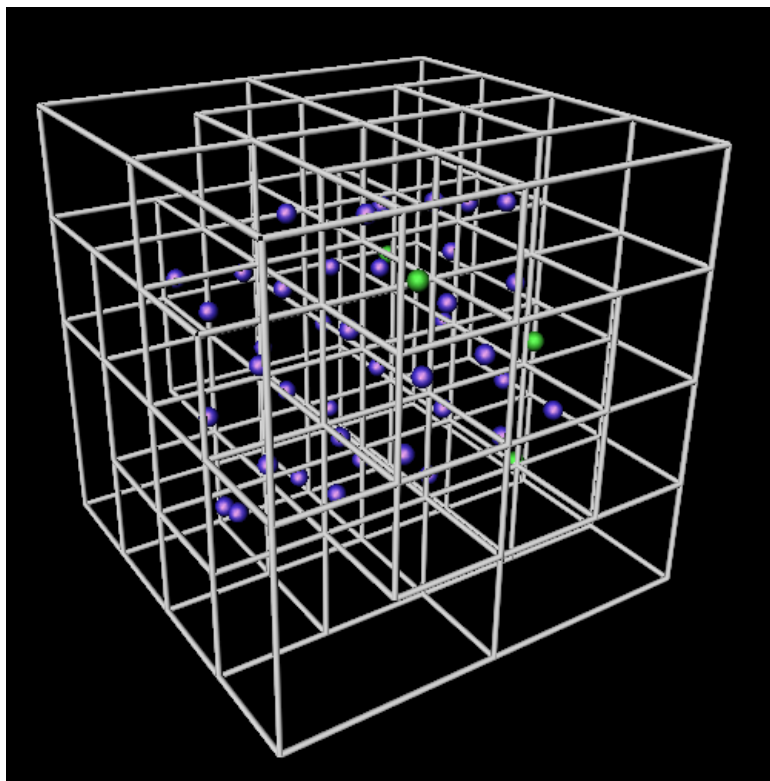**Figure 4.21:** Eight daughter cells of the root cell of an octtree decomposition of nos7

Figure 4.22: Second level of an octtree decomposition of nos7
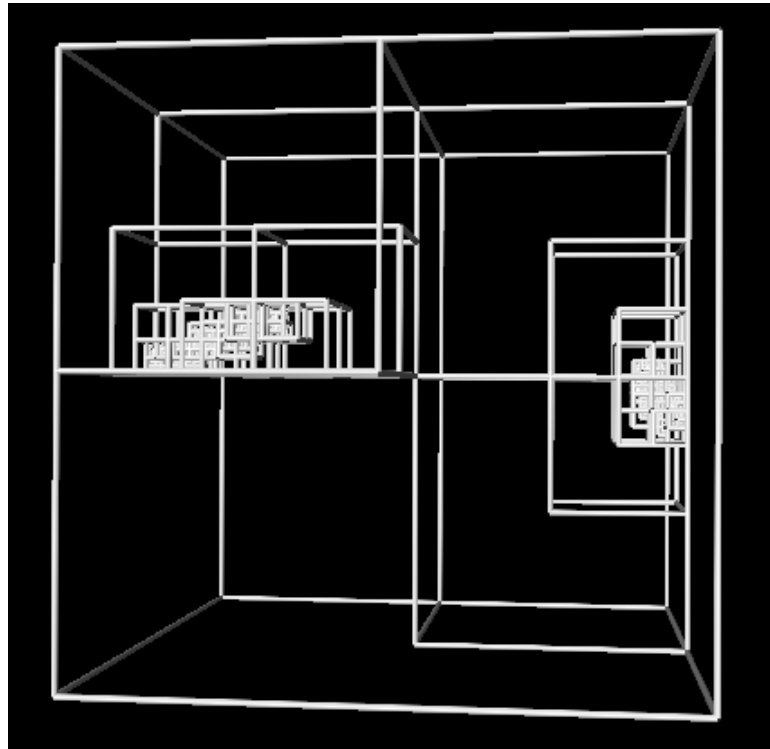
Figure 4.23: Complete Octtree of `bfw782b`

ever, it is important to note that the formulation of the *cell opening criterion* remains the same.

The extra dimension roughly doubles the computational effort in building the tree, which is still negligible compared to the effort involved in the nonedge force calculation. The force computation involves a few extra floating point operations, and the total overhead in moving from two dimensions to three dimensions is around 25%; see for example Table 4.4 for a performance comparison of the two and three dimensional WAVE-FRONT algorithms.

Figures 4.25 and 4.26 are alternate two dimensional projections of a three dimensional layout of the graph `qh1484`. Figures 4.27, 4.28 and 4.29 are visual précis of a third, fourth and fifth level horizon précis of `qh1484` (see Appendix A).

Figures 4.30 and 4.31 are alternate two dimensional projections of a three dimensional layout of the graph `dwa512`. Figures 4.32, 4.33 and 4.34 are visual précis of a second, third and fourth level horizon précis of `dwa512` (see Appendix A).
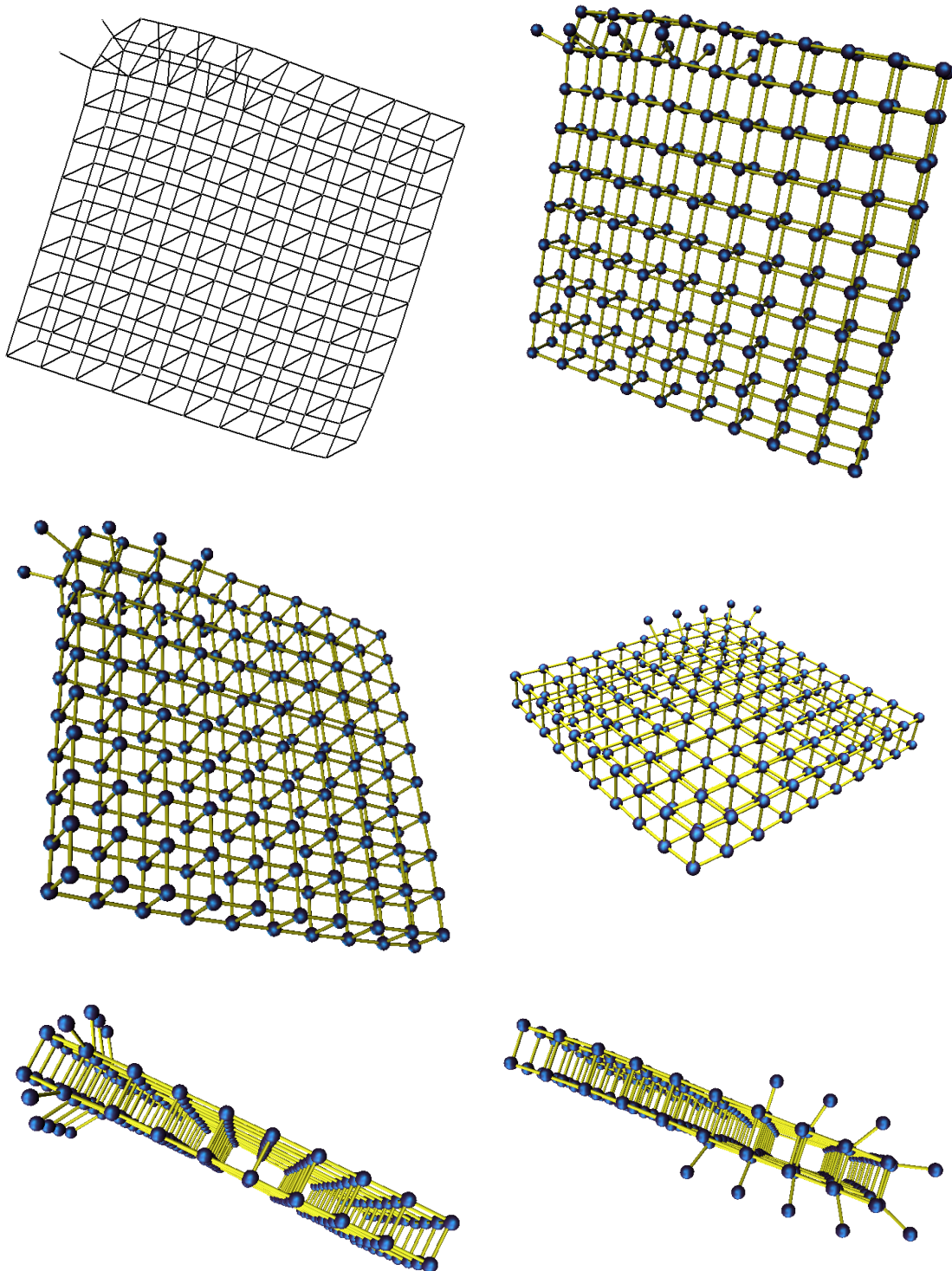
Figure 4.24: *RDB209* a graph of a Reaction-diffusion Brusselator Model, from Chemical Engineering, drawn with FADE2D then with FADE3D and viewed from different 3D viewpoints
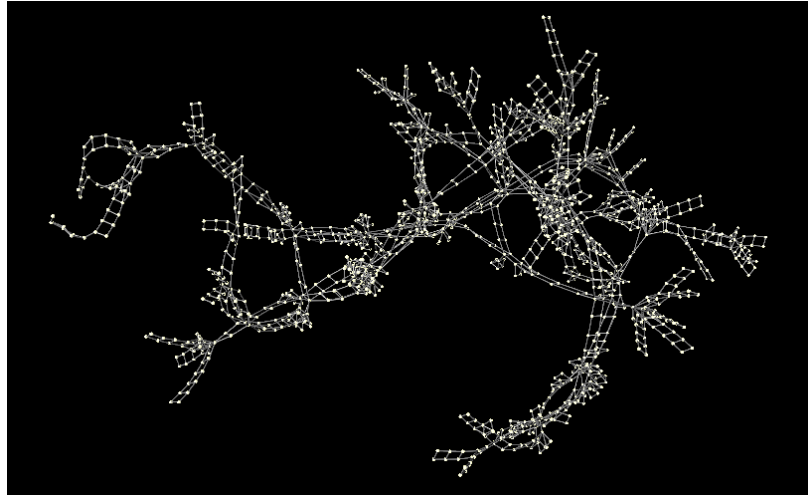
Figure 4.25: FADE3D drawing of `qh1484`
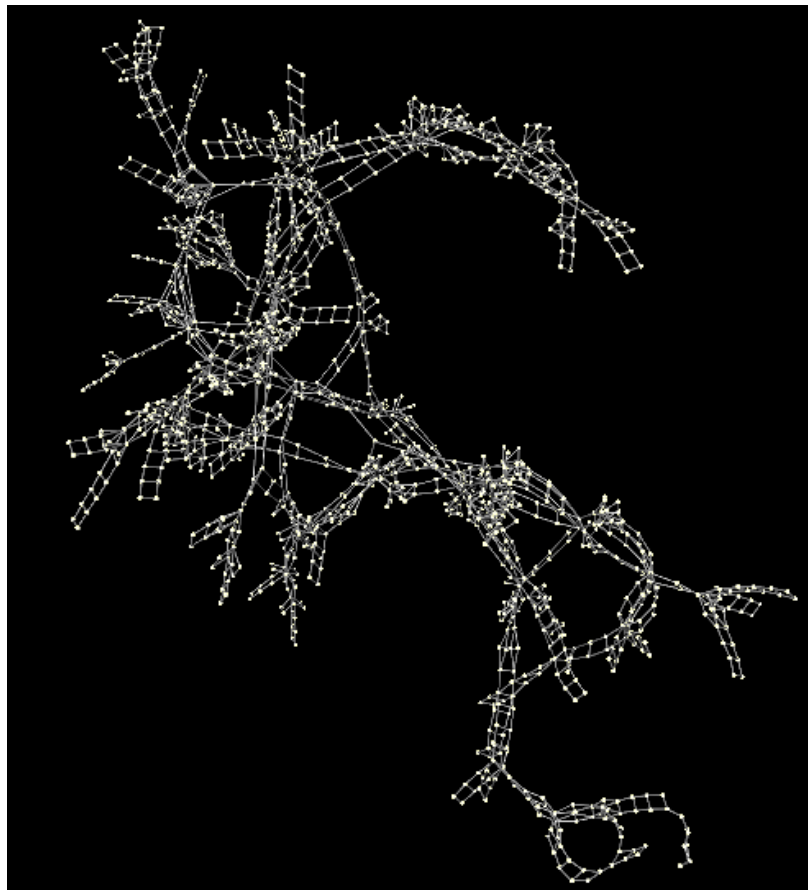
Figure 4.26: FADE3D drawing of `qh1484`
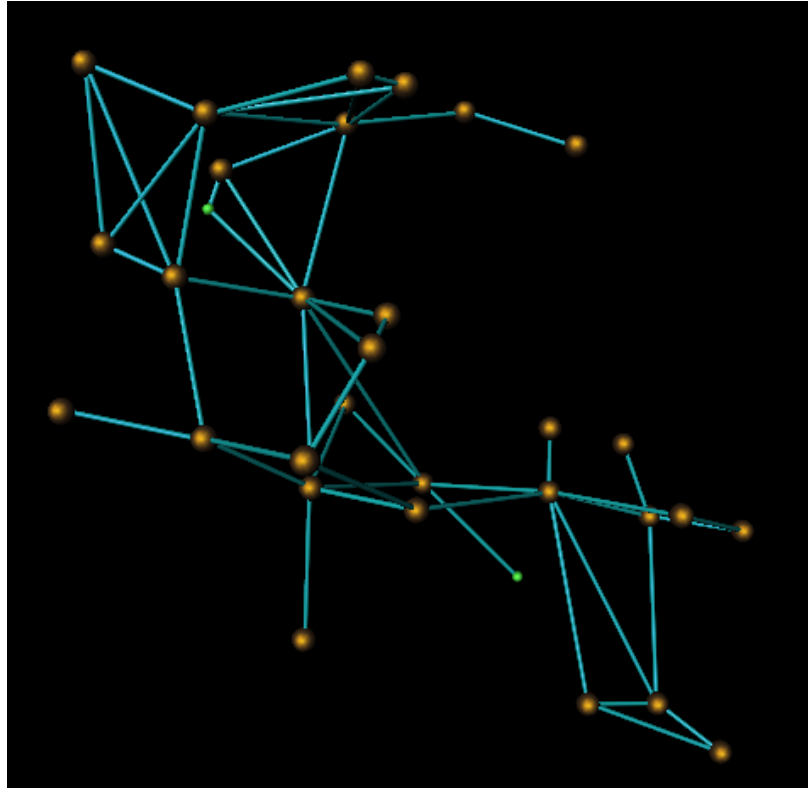
Figure 4.27: Third level horizon drawing, of `qh1484`, with a visual weight of 2% drawn with FADE3D.
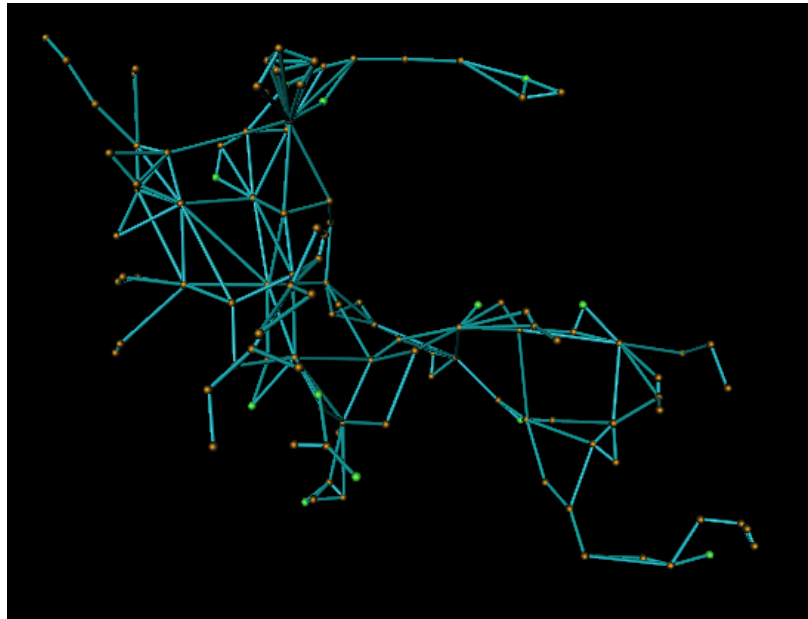
Figure 4.28: Fourth level horizon drawing of `qh1484`, with a visual weight of 7% drawn with FADE3D.
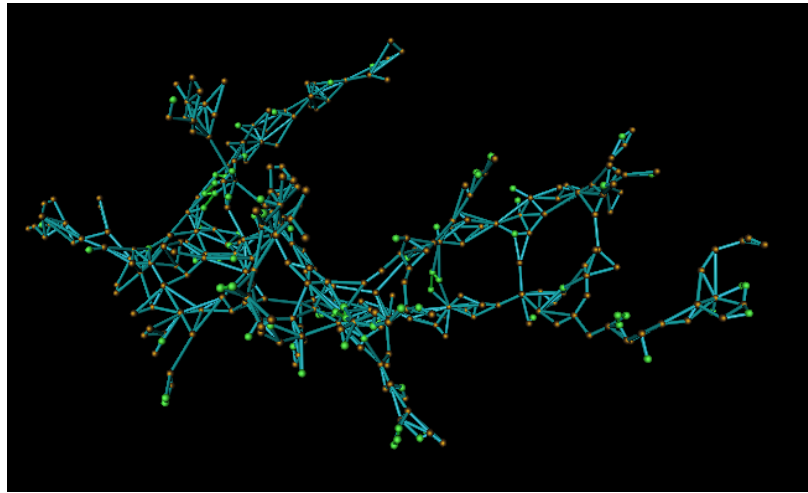
Figure 4.29: Fifth level horizon drawing of `qh1484`, with a visual weight of 24% drawn with FADE3D.
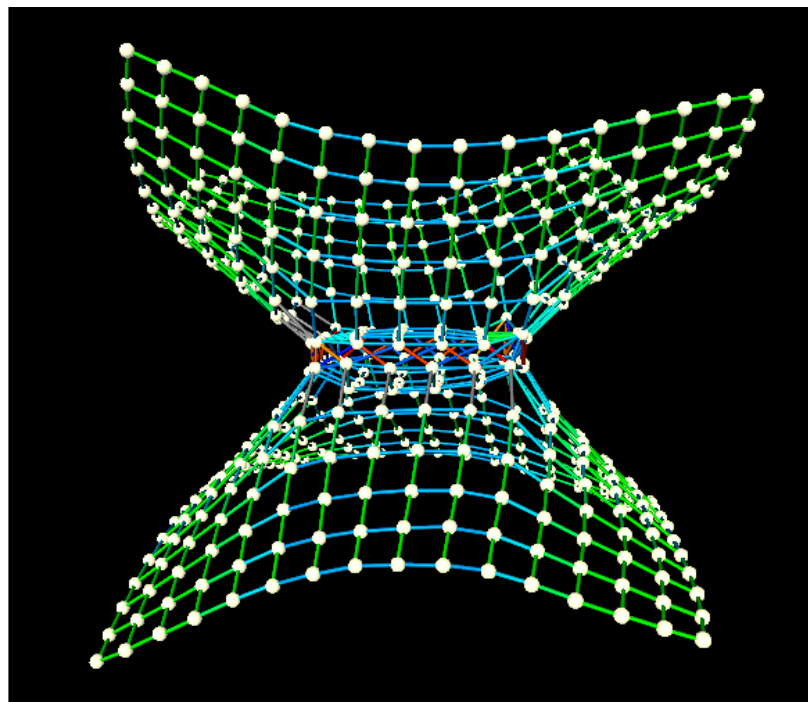
Figure 4.30: FADE3D drawing of `dwa512`
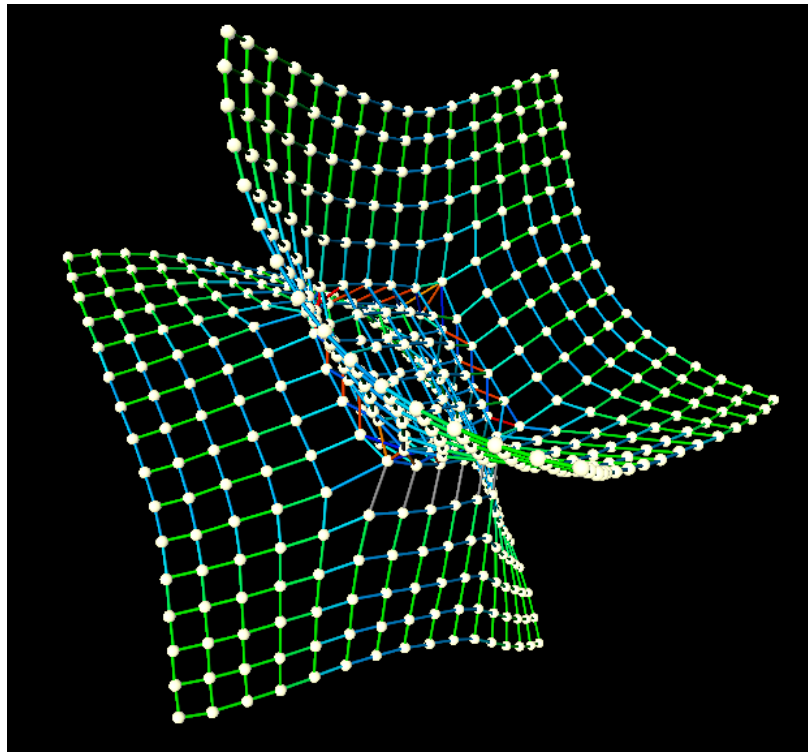
Figure 4.31: FADE3D drawing of `dwa512`
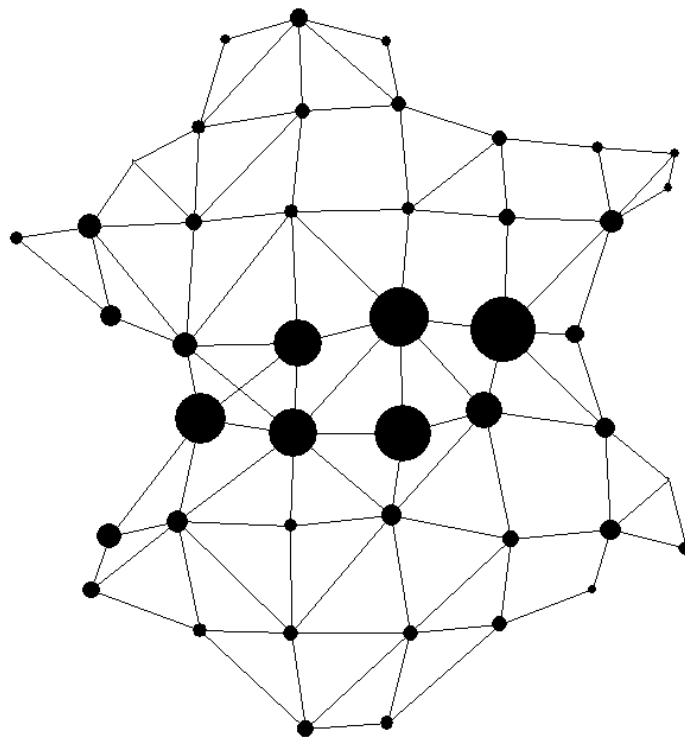
Figure 4.32: A visual précis of `dwa512a`, from the matrix market, drawn with FADE2D
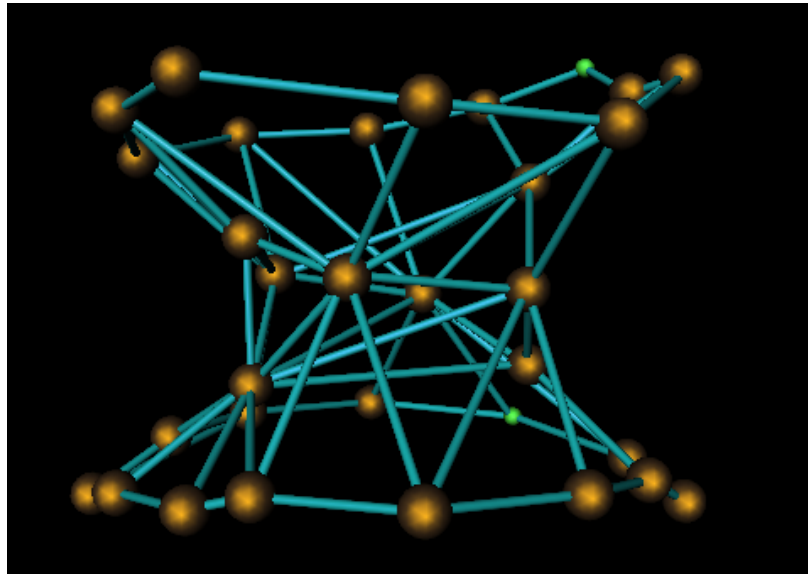
Figure 4.33: Third level horizon drawing, of `dwa512`, with a visual weight of 6% drawn with FADE3D.
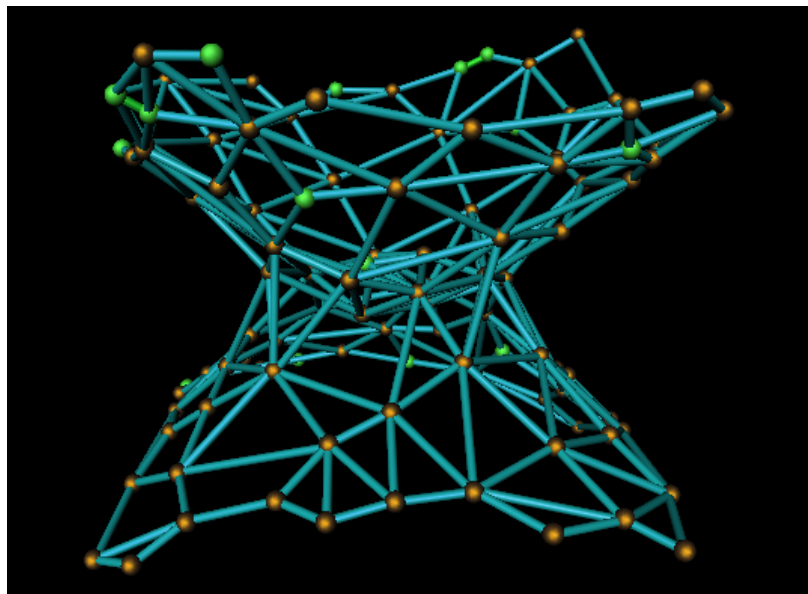
Figure 4.34: Fourth level horizon drawing, of `dwa512`, with a visual weight of 22% drawn with FADE3D.
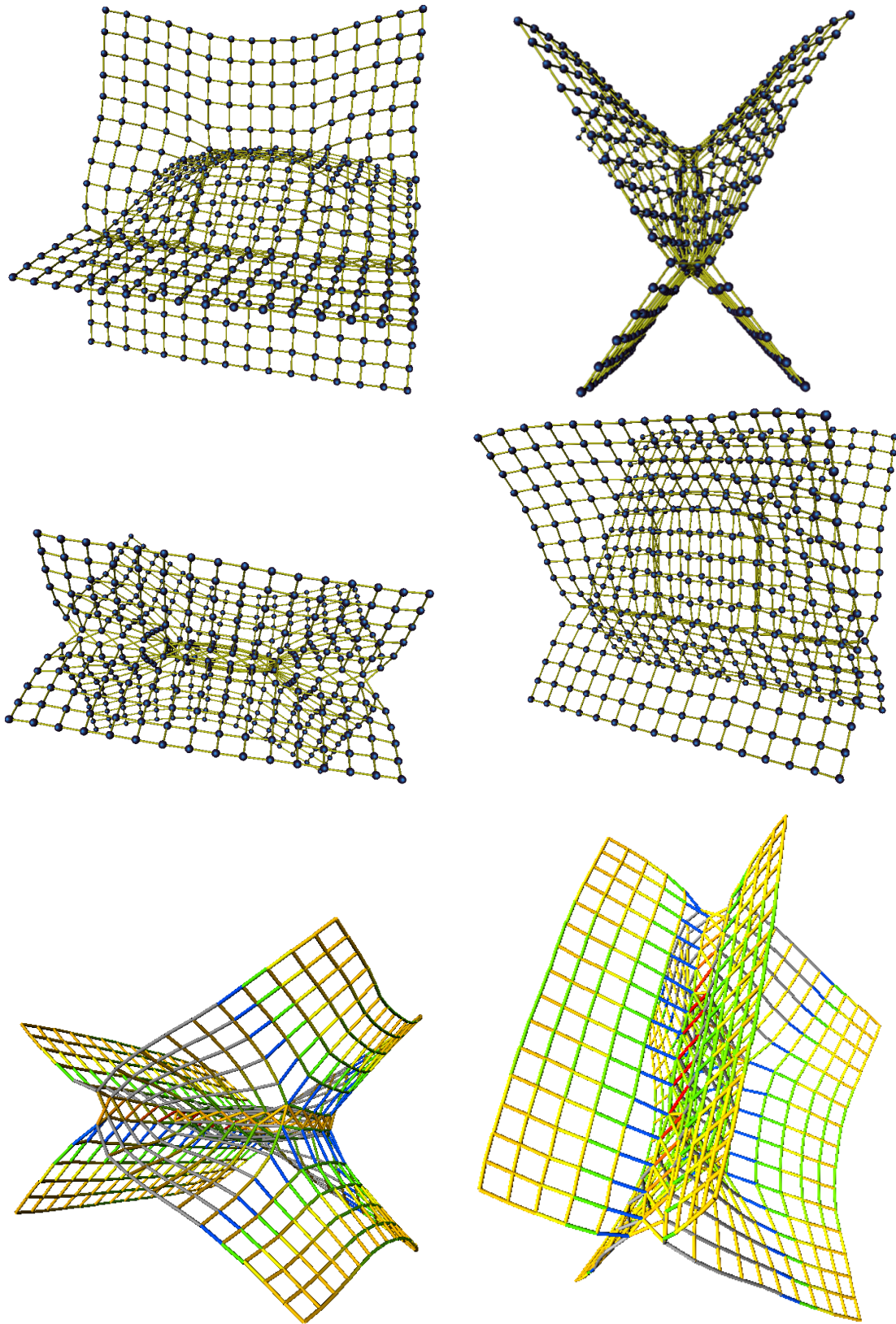
Figure 4.35: *dwb512* a graph of the Square Dielectric Waveguide problem in Electrical Engineering, drawn with FADE3D and viewed from six different 3D viewpoints. The bottom two drawings are without nodes but include a colour coding of the strengths of the edges.
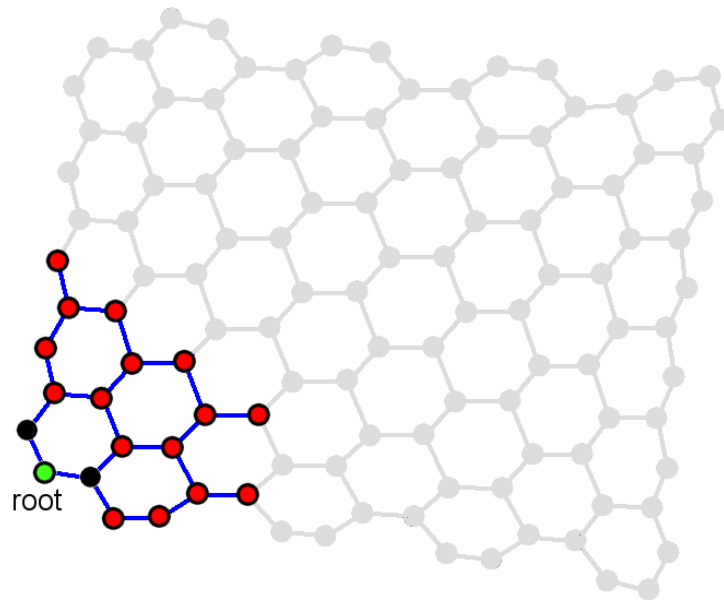
Figure 4.36: Starting to pass a wave of thickness 3 through a graph. Black - *fixed*, Red - *in the wave*, Grey - *as yet unassigned*

4.6   Wave Front FADE

A *wave-front* layout is a breadth first traversal of the graph structure coupled with the standard FADE algorithm, either in two or three dimensions. As noted earlier, although our layout method is computationally inexpensive per iteration, it typically requires a large number of iterations to move the energy state from a random configuration to a low energy state which exhibits an aesthetically pleasant drawing. For example, take a large graph which has been assigned a random layout. The first few iterations of FADE typically improve the micro relational structure rapidly but ignore the macro structure. This results in adjacent nodes being drawn together but having little regard for non-adjacent relationships (nonedge forces). Large graphs can often remain in this tangled state or require a prohibitive number of iterations of FADE to improve the aesthetics of the layout. See for example the three dimensional drawing of the triangular mesh graph of $33,975$ nodes, shown in Appendix A.

Observation of this behaviour resulted in the development of the "wave-front" force directed layout method. This is a computationally inexpensive approach to the generation
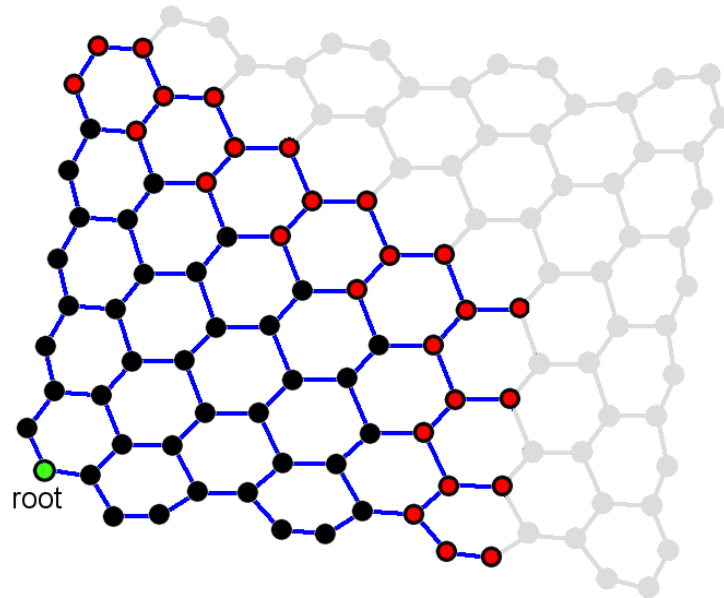
Figure 4.37: Midway through passing a wave of thickness 3 through a graph. Black - *fixed*, Red - *in the wave*, Grey - *as yet unassigned*

of the initial layout. The algorithm uses a "wave", which is a subgraph that moves through the graph from a root, in a breadth first fashion, as shown in Figures 4.36 and 4.37. At each step in the traversal, forces are applied m times, only to the nodes in the wave, denoted by the red nodes in Figures 4.36 and 4.37. After the wave passes through a node, it is fixed, denoted by the black nodes. The wave has a given thickness w, which results in each node being considered in a force calculation $w * m$ times. A thicker wave-front typically improves the layout, but increases the runtime. Nodes that have yet to enter the wave do not have an assigned geometry. When a new node is added to the wave, it is assigned a location within a bounding circle or sphere of an adjacent node in the wave, that has a geometry. The bounding circle or sphere is defined by the radius of the edge connecting this node to the node from which it takes its geometry.

**Algorithm 6** describes the basic wave-front method, with a single root. Alternate formulations of the wave-front include the selection of a random set of roots, rather than a single root. The function **breathFirstWave** expands the current waveGraph to include new nodes, connected to the current waveGraph, while removing nodes that have been in the waveGraph for a number i of iterations of **breathFirstWave** where $i = thickness$. This

ensures nodes only remain in the wave front for a fixed number (*thickness*) of iterations before they are considered fixed and not re-considered in the force directed placement.

Algorithm 6 Compute 2D Wave Front Drawing

Input: graph, thickness, Force Intensity $\mathcal{K}$, force application count m
Input: Criterion Type $\mathcal{CT}$, Cell Opening Criterion θ
Input: Clustering Arity Width $\mathcal{A}$
begin
  $maxX \Leftarrow 0$
  $maxY \Leftarrow 0$
  for each node v **do** {Reach Each Disconnected Component}
    if not $v.drawn$ **then** {Node Not Yet Positioned}
      $v.drawn \Leftarrow true$
      $waveGraph \Leftarrow v$
      $v.x \Leftarrow maxX$
      $v.y \Leftarrow maxY$
      $depth \Leftarrow 1$
      $waveGraph.waving \Leftarrow true$
      while $waveGraph.waving$ **do** {Passing Wavefront}
        $waveGraph \Leftarrow$ breathFirstWave($waveGraph$,$depth$,$thickness$)
        2DEdgeForces($waveGraph.edges$)
        for i in 1 to m **do** {Apply forces m number of times to this sub-graph}
          for each node u in $waveGraph.nodes$ **do** {Nonedge Forces on Each Node in WaveFront}
            $\mathcal{BB} \Leftarrow$ boundBoxofNodes($waveGraph.node$)
            $root \Leftarrow$ BuildGeometricClustering($waveGraph.nodes$,$\mathcal{A}$,$\mathcal{BB}$)
            2DAproxForce(u,$root$,$\mathcal{K}$,$\mathcal{CT}$,θ)
          end for
        end for
      end while
    end if
  end for
end

Figure 4.38 shows a two dimensional drawing of a 1600 node graph computed by a wave-front method. This graph is a randomly generated graph seeded with k clusters. Here $k = 28$ and the inter-cluster connectivity is weak. Figure 4.39 shows a different drawing of the same graph. Here a high level three dimensional visual précis from the hierarchical compound graph is overlaid with a cubical hierarchical space decomposition.

Table 4.4 provides a comparative analysis of a two and a three dimensional wave-front algorithm. These times are averaged over 50 runs of the wave-front algorithm. The
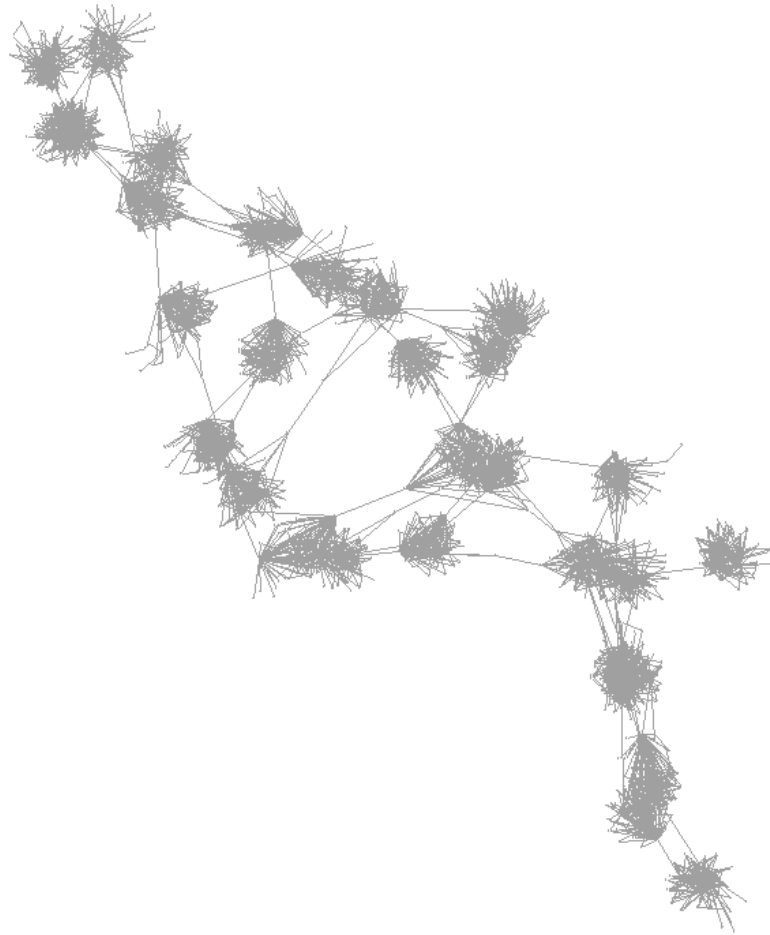
Figure 4.38: A FADE drawing, by the wavefront method, of a randomly generated clustered graph, with $k = 28$ centres and $n = 1600$ nodes.
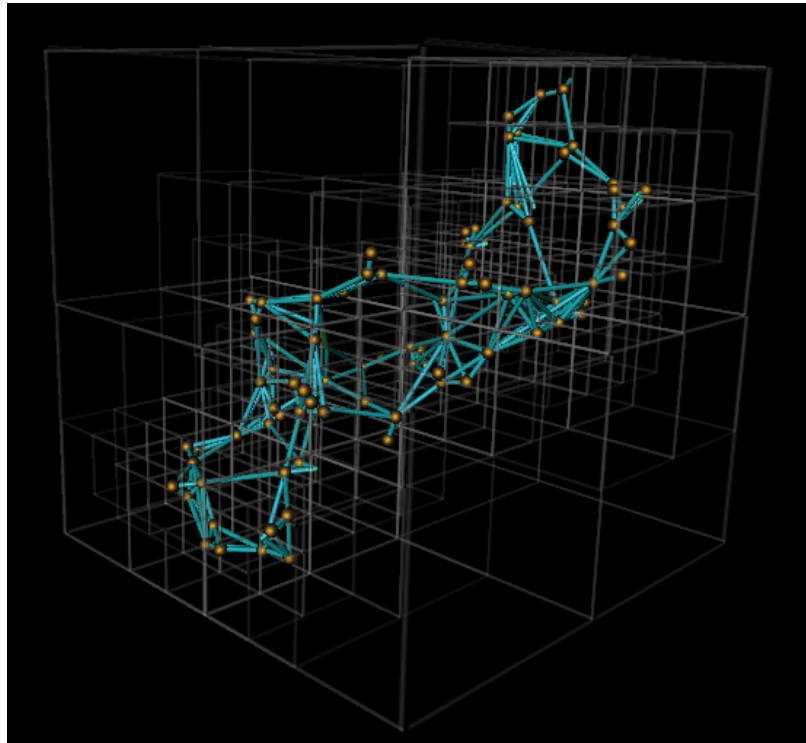
Figure 4.39: 3D Visual précis of a three dimensional layout of the graph shown in Figure 4.38
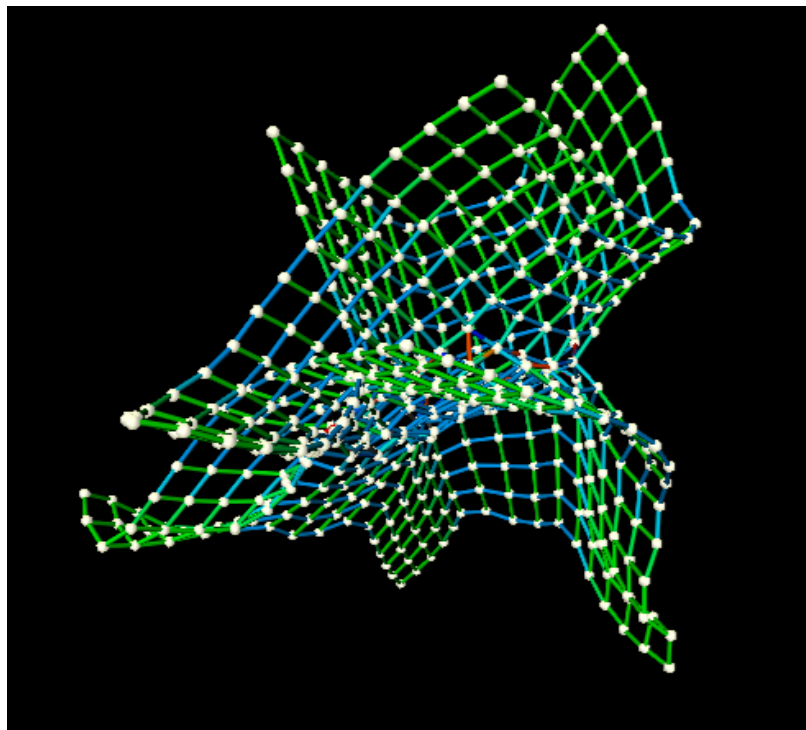

Figure 4.40: A FADE3D drawing, by the wavefront method, of `dwa512`, with $w = 6, \theta = 1.6$ and $m = 2$

Graph	$|\mathcal{N}|$	$|\mathcal{E}|$	2D $\theta = 0.8$ $m = 2$	2D $\theta = 0.8$ $m = 6$	2D $\theta = 1.6$ $m = 6$	3D $\theta = 0.8$ $m = 2$	3D $\theta = 0.8$ $m = 6$	3D $\theta = 1.6$ $m = 6$
dwa512	512	1004	0.66824	1.8654	1.1719	0.8446	2.2465	1.2501
bfw782a	782	3394	1.3338	3.8798	2.3465	1.6045	4.592	2.55
qh1484	1484	2492	2.4773	7.2217	4.472	3.2927	8.963	4.845
qh768	768	1322	1.1233	3.379	1.809	1.4976	4.021	2.078
plsk1919	1919	4831	3.2275	9.5322	6.633	3.8342	10.819	7.4941
cry10000	3699	7164	7.1737	20.7732	15.308	8.6492	23.511	17.464
bcspwr10	5300	8271	12.32	32.8294	16.29	18.126	47.822	18.512
dw8192	8192	17404	19.3218	55.644	37.011	23.973	68.989	44.063
rdb3200l	3200	7840	6.1424	18.715	11.967	7.671	22.29	13.687
add20	2395	7462	4.313	12.4418	6.1903	3.345	20.63	7.5567
add32	4960	9462	11.9706	35.7006	16.749	17.6928	52.301	18.616
data	2851	15093	6.6929	19.2406	11.4234	7.8314	22.448	12.0814
crack	10240	30380	24.919	72.5043	32.710	40.177	115.513	52.4268
whitaker3	9800	28989	28.3418	89.0405	42.6301	47.203	151.049	56.52
fe4elt2	11143	32818	33.092	107.015	61.45	45.662	139.72	75.25
cti	16840	48232	51.86	188.531	101.3961	83.2675	266.413	118.534

Table 4.4: Time in seconds, to compute two dimensional (2D) and three dimensional (3D) wave-front drawings. θ is the value for the fixed accuracy parameter of each FADE algorithm. m is the number of iterations of FADE per step in the breadth first traversal. $w = 6$ is the thickness of the wave.

thickness of the wave is held constant, while the fixed accuracy parameter and the number of iterations of the FADE algorithm per step in the traversal is modified. As expected, tripling the number of iterations of FADE per step, approximately triples the runtime for the wave-front method. Further, the three dimensional wave-front is typically 25% slower than its two dimensional equivalent. However, increasing the inaccuracies permitted (by increasing the value of θ) quickly reduces the run-time for both two and three dimensional algorithms.

The data in Table 4.4 comes from the case study in Chapter 6 and a graph partitioning archive of large graphs used in a multi-scale force directed research in [283, 284].

4.7 Example Drawings

Additional two and three dimensional examples are included in Appendix A as pictures, three dimensional models, visual précis and video simulations.

4.8   Remarks

This Chapter has presented the FADE paradigm for large graph drawing. By using the hierarchical compound graph, this suite of force directed algorithms provide considerably faster layouts. Although these estimation methods introduce an error into the force calculation we can measure and compare these errors, which allows us to study the effects of using different values, and how these affect the overall performance of the algorithm and the drawings produced. We use the FADE2D algorithm for detailed study in the following case studies.

The three dimensional FADE3D algorithm requires roughly 25% more computation for the same fixed accuracy parameter, as its two dimensional counterpart. The wave front algorithm, couples a graph theoretic traversal with either a two or three dimensional FADE force directed layout. The computational efficiency of this method stems from the inclusion of a small section of the graph, at any one time, while computing a layout. The aim of this method is not to produce a final drawing but rather to produce a better initial drawing that FADE2D or FADE3D can improve upon.

Overall, the paradigm described here uses the hierarchical compound graph model for force directed placement. This, coupled with the measures and visual précis described in Chapter 3 allows us to marry the layout, abstract representation, and clustering measurement, using a single graph model.

Case Study I: Software Visualization

"Data is not information, Information is not knowledge, Knowledge is not understanding, Understanding is not wisdom." - Cliff Stoll and Gary Schubert

Software visualization is the use of computer graphics and animation to help illustrate and present various aspects of a software system. Examples of such aspects include the algorithms and data structures used in a piece of software, or details of which functional components access data from each other. As such, software visualization is a very broad field ranging from aids for teaching algorithms to techniques for displaying large software systems. It is research in the latter form that we address in this case study.

Visualization of source code falls into two broad categories, namely *static* and *dynamic* visualization. Static aspects of the software include structures, such as call relations, which can be gleaned from the raw source code. Identifying dynamic relations or dynamic structures such as memory usage, requires the software to be executed and such dynamic data to be collected. The large data sources used in this case study come about from a static analysis of four software systems, and as such, we do not further address the issue of dynamic software visualization in this thesis.

These data sets, called *resource flow graphs*, consist of *architectural quarks* and their interrelationships, see Figure 5.1. Given that each resource flow graph contains several types of nodes and many different types of relationships, these graphs have been further refined into a set of *views*. Often these views consist of a large number of disconnected components. For the purposes of this case study and to contrast it fairly with the next, the graphs are considered as a single graph. Not as a series of smaller sub-graphs.

The work of Koschke [163] demonstrates how these views can be used in the detection of *components* by automatic and semi-automatic means. This work shows that automatic component detection is at best only moderately successful. Based on this, a semi automatic method is proposed by Koschke to integrate the user into the component discovery loop. This integration includes techniques such as allowing the user to modify weights, control search parameters, and apply combinations of discovery techniques. As noted, other more visual interaction techniques might allow better user feedback. It is this intuition that is realized and evaluated in this case study for the visualization and abstract representation of these views from the underlying resource flow graphs. All the performance and time measures in this chapter were computing using an implementation in Parlanse, running on a 500Mhz Pentium III processor with 512Mb of memory.

5.1 Context

For the purposes of this case study, we restrict our interest to program comprehension and reverse engineering. Specifically, our goal is to evaluate the FADE paradigm in producing visualizations that can be used by software engineers in the task of comprehending and classifying sections of an existing software system. In this section we present the context of our evaluation.

5.1.1 The Bauhaus Project

In this Chapter we investigate the drawing, abstract representation and measurement of a range of medium to large graphs from the "Bauhaus Project". The *Bauhaus project* is a research collaboration between the Institute for Computer Science of the University of Stuttgart (ICS) and the Fraunhofer Institute for Experimental Software Engineering in Kaiserslautern (IESE). These graphs have been used in the detection of components from "legacy software systems" by automatic and semi-automatic means. This work on architecture recovery showed that automatic component discovery is at best only moderately successful. One proposed refinement is to integrate the user, using a visual tool, into the

component discovery loop.

For the purposes of this thesis, we focus on five issues in this case study:

- The visualization and abstract representation of the overall structures and edge sets in these graphs.

- Discussion of the drawings and their abstract representations, in terms of their potential in component discovery.

- The measurement of various aesthetics of the final drawing and some *horizon drawings* (abstract representations), introduced in Chapter 2.

- The performance measurement of the primary layout algorithm (FADE2D) from our FADE paradigm, introduced in Chapter 4.

- The use of our hierarchical compound graph quality measures from Chapter 3, to determine if there is progressive cycle of layout and clustering improvement with this class of data.

5.1.2 Reverse Engineering

A *legacy software system* is an older application program which continues to be used because the cost of replacing it or redesigning it is prohibitive [247]. Such systems are maintained and updated, to repair or add new features as required, although they are often large, monolithic, and difficult to modify. Lehman hypothesized that "a program used in a real world environment must change (adapt) or become progressively less useful in that environment". His hypothesis is born out in practice where an organization *evolves* its software systems to meet the changing business environment. *Software evolution*, rather than total re-development, is now standard for most of the business software systems in use. Businesses have to come to realize that their code represents a valuable commercial asset rather than being a disposal commodity.

A large percentage of software engineering activity deals with such legacy systems, as opposed to *from scratch* software development (forward engineering). Further, most

programmers work on software that other people have designed and developed. As such, existing research has shown that up to 50% of a software evolvers time can be spent determining the intent of source code [202]. Clearly, supporting this task to enhance the ability of a software engineer to effectively analyze such legacy software systems is an important task.

Globally, there are many billions of lines of legacy code undergoing constant evolution. It is this code that current software engineers must deal with, rather than any idealized software system of the future. Software evolution continues to be an ad-hoc, haphazard, and often undocumented process of continual change. Corporate culture seems to accept this chaotic approach with studies showing that less than one in five companies maintain a complete application inventory, less than one in 30 know the rate at which their code bases are changing, and less than one in 100 companies collect quality information about their software projects [245].

There are several unfortunate side effects to this pervasive attitude in software evolution:

- Often the code is the only reliable description of the software system.

- Existing documentation does not typically reflect the software system currently in use, that is, it is out of date.

- Whatever original design took place, if any, tends to become more complex and less structured over time.

- It is often only the software itself that reliably encodes the business rules of an organization.

These effects are often seen in practice and have given rise to the need for *program understanding* or *program comprehension* techniques. These techniques typically require manual effort before any higher level abstract concepts can be formed by "Reverse Engineering". *Reverse Engineering* is the process of analyzing a subject system to identify the system's components and their interrelationships, and create representations of the system

in another form or at a higher level of abstraction [49]. "Component discovery" is one type of reverse engineering analysis.

5.1.3   Component Discovery

In OO terms an *object* is an instance of a class with methods and attributes. A *component* is an instance of a class description with generic methods through which it can advertise the functionality it supports to the system into which it is loaded. Instead of relying on these definitions, we use Koschke's terminology for components, which we now review.

A *component* is a computational element of a larger system. A *connector* links or holds together components. According to Koschke [163], an *atomic component* is a low-level software component (non-hierarchical) built from types, variables, and functions. A further distinction, between two kinds of atomic components, namely "Abstract Data Types" (ADT) and "Abstract Data Objects" (ADO) can be made as follow. An *abstract data type* is in an encapsulated abstraction from which instances of that type can be created. An *abstract data object* consists of a set of primitive variables and constants along with routines. There is only one instance of an ADO since the developer has not placed this into a grouping such as an ADT. Along with atomic components, connectors describe the interactions or communication between these components. Depending on the language, connectors include: pipes, method calls, shared variables, sockets, and files. At higher levels of abstraction, atomic components can be connectors between more abstract architectural elements. Finally, a *hybrid component* is a state-based ADT or an ADO with a subordinated type.

In an attempt to identify and classify the knowledge embodied in legacy systems, Koschke presents a framework for the comparison of automatic and semi-automatic atomic component identification techniques [163]. These techniques aid in the identification of the components of a software architecture.

The purpose of this case study is to investigate whether the FADE paradigm is suitable for visualizing the software views described in [163] to aid in low level component identification.

Note that we do not directly address the related problem of "design recovery" [111]. *Design recovery* is the coupling of domain knowledge, external information, and deduction to the observations of the subject system, to identify meaningful higher level abstractions beyond those obtained directly by examining the system itself [49].

5.2 Description of Graphs in this case study

The data sets used in this case study are based on extracted elements and relations from the static code analysis of 4 software systems. These systems are *Bash*, which contains approximately 38,000 lines of code (38 Kloc), *CVS* (41 Kloc), *Mosaic* (37 Kloc) and *Xaero* (31 Kloc) [163].

5.2.1 The Software

The **B**ourne-**A**gain **SH**ell (Bash) is a freeware Unix command interpreter. It provides features such as: interactive command line editing, shell programming, prompt expansion, job control, aliasing, and command history. The current version of Bash is 2.05, whereas this analysis data comes from version 1.14.4. Given its open-source nature, Bash has a heavily ported, modified, and maintained structure.

The open source **C**oncurrent **V**ersions **S**ystem (CVS), is a tool to provide *version control* in individual or collaborative software development. Version controls allow a software developer to retrieve an older version of the software. This is crucial for tracking and managing the ongoing feature development of software. Instead of storing every past version of every file, CVS only records the differences between versions (the *deltas*). Further, CVS insulates different developers from each other. Each developer works on a local copy and it is only later, on *check in*, that local copies are merged by CVS. This analysis data comes from CVS Version 1.8.

XAERO is an **X**-windows **A**nimation **E**ditor for **R**ealistic **O**bject movements. It is a physics based kinematic simulation and animation system based on rigid body models. Xaero includes an editor for defining simple virtual scenes consisting of basic three di-
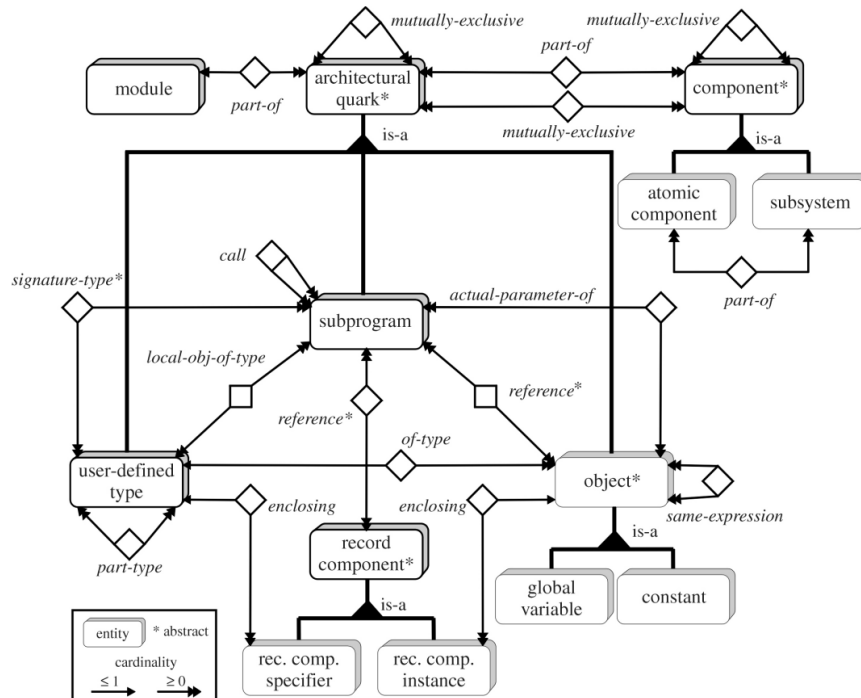
Figure 5.1: "Entities" and their interrelationships. Reproduced by courtesy of Rainer Koschke from [163]

mensional objects. These objects can be connected with links such as: springs, dampers, rods, and joints. Along with the objects and connections, a user can define forces such as: gravitation, air resistance, and friction along with other unnatural forces. Once the scene is built and the forces defined, the scene can be animated through a series of timesteps. This analysis data comes from Xaero Version 1.7.

Mosaic, from the National Center for Supercomputing Applications at the University of Illinois, is an Internet information browser and WWW client. This software, which is freely available for internal use, is no longer under development. The final version of Mosaic supports HTML 3.2. This analysis data comes from Mosaic Version 3.0 on Unix.

5.2.2   Elements and Interrelationships

As noted in Section 5.1.1, our aim is to visualize, abstractly represent and measure our FADE paradigm against the raw reference data produced by the Bauhaus project. These resource flow graphs are output from a non-commercial off the shelf (NCOTS) reverse

engineering tool called RIGI [265] from the University of Victoria in Canada. The source code from each reference system is loaded into RIGI and then various static analyzes are performed. This tool allows elements of the software to be identified such as procedures, variables, constants, and subprograms. Further, static relationships such as calls and data accesses, between these elements are identified and provided to the reverse engineer.

Koschke describes in greater detail the resource flow graphs in terms of the elements (*entities*) and their interrelationships (see [163]). The suitability of an automatic clustering technique is often dependent on the class and quality of the entity and relationship descriptions available. For the purposes of this thesis, we describe, with reference to Figure 5.1, only the entities and relationships of the views that we visualize.

- a `record` consists of a group of logically related elements that form some collective abstraction.

- a `record component` is an element that is `enclosed` in a type declaration or `enclosed` in an instance of a record.

- `is-part-of-type(`u`,`v`)`: the type element u is part of the declaration for type element v (this is a transitive relation).

- `parameter-of-type(`u`,`v`)`: the sub-program u has a formal parameter of type v.

- `return-type(`u`,`v`)`: the sub-program u returns a value of type v.

- `variable-of-type(`u`,`v`)`: the variable u is of type v.

- `member-of-type(`u`,`v`)`: the element u is of type v.

- `parameter-of-type(`u`,`v`)`: the sub-program u has a formal parameter of type v.

- `local-var-of-type(`u`,`v`)`: the sub-program u has a local variable of type v.

- `reference`

Subgraph Type (view)	Edge types used
Type composite view	`is-part-of-type enclosing`

Type usage view	`parameter-of-type return-type`
	`member-of-type variable-of-type`
	`local-var-of-type`

Object reference view	`set use address`
Signature view	`parameter-of-type return-type`
Same expression view	`same-expression`
Actual parameter view	`actual-parameter`
Dominate view	`dominate`

Table 5.1: Edge types from the resource flow graph used to form each *view*

(a) `set(`u`,`v`)`: the sub-program u sets the value of v.

(b) `use(`u`,`v`)`: the sub-program u uses the value of v.

(c) `address(`u`,`v`)`: the sub-program u takes the address of v.

- `actual-parameter(`u`,`v`)`: the element u is an actual parameter in a call to v.

- `same-expression(`u`,`v`)`: the element u appears in the same expression as v.

- `dominate(`u`,`v`)`: the subprogram u dominates the subprogram v, as every path to v passes through u.

Views

A resource flow graph or resource usage graph is a graph model that initially describes a largely syntactic view of code [263, 265]. A *view* is a subgraph of the resource flow graph representing some special aspect. In general, views can be used either to answer specific questions or to perform high level analyses, such as those described in [42, 142, 163, 85, 174, 176, 188, 263, 265]. For example, a simple *call view* can aid in the automatic or user guided detection of dead-code.

In this case study we present the visualization, abstract representation and measurement of seven views extracted from resource flow graphs of Bash, CVS, Mosaic, and Xaero. The seven elementary views are directly derived from the source code and have been used in component discovery research. The edge relations for each view that must be extracted from the resource flow graph are described in Table 5.1. These views include:

- A *type composite view* consists of *type* and *record component* nodes; this shows how types are built. The Bash type composite view is shown in Figure 5.6, for CVS in Figure 5.7, for Mosaic in Figure 5.8, and Xaero in Figure 5.9.

- A *type usage view* consists of *subprogram*, *type*, *variable* and *constant* nodes; this gives a cross-referenced view of the usage of types. The CVS type usage view is shown in Figure 5.10.

- An *object reference view* consists of *subprogram*, *variable* (object) and *constant* (object) nodes; this gives an indication as to which variable or constants are accessed by a particular subprogram. The data used in this case study only contains an *approximation* of this view. The underlying graph comes from a static data flow analysis of the code, so all relationships explicitly declared in the code are included. The Bash object reference view is shown in Figure 5.11, for Mosaic in Figure 5.12, and Xaero in Figure 5.13.

- A *signature view* consists of *subprogram* and *type* nodes; this specifies the parameter interface of the subprograms. The Bash signature view is shown in Figure 5.2, for CVS in Figure 5.5, for Mosaic in Figure 5.4, and Xaero in Figure 5.3.

- A *same expression view* consists of *variable* and *constant* nodes; this shows which entities occur in the same legal combination of symbols that represent a value, that is, in the same *expression*. The CVS same expression view is shown in Figure 5.14.

- An *actual parameter view* consists of *subprogram*, *variable* and *constant* nodes; this shows exactly which variables or constants form the actual parameter values to a given subprogram. The Bash actual parameter view is shown in Figure 5.15.

- A *dominance view* consists of *subprogram* nodes; this gives an indication of which subprograms strictly dominate each other, as described in Section 5.2.2. The CVS dominance view is shown in Figure 5.16, for Mosaic in Figure 5.17, and Xaero in Figure 5.18.

Graph Type	Bash		CVS		Mosaic		Xaero	
	$|\mathcal{N}|$	$|\mathcal{E}|$	$|\mathcal{N}|$	$|\mathcal{E}|$	$|\mathcal{N}|$	$|\mathcal{E}|$	$|\mathcal{N}|$	$|\mathcal{E}|$
Type composite view	749	767	981	924	4642	5197	1342	1411
Type usage view	835	880	1253	1459	3170	3293	1994	2465
Object reference view	749	1275	964	1871	369	487	894	2596
Signature view	277	284	484	516	1182	1322	818	1134
Same expression view	171	224	237	414	24	13	234	430
Actual parameter view	295	347	405	634	59	68	257	299
Dominate view	NA	NA	512	511	1799	1798	1137	1136

Table 5.2: Sizes of subgraph views, extracted in the Bauhaus project from four software systems.

Once each view has been formed, it can be described in simple combinatorial terms. The number of nodes and edges from each view of each system in this case study are described in Table 5.2.

5.3 The Experiments

To investigate the FADE progressive cycle in producing drawings and visual précis we tested the FADE2D, WAVE-FRONT and FADE3D algorithms on the views described in Section 5.2.

The results of this case study are presented in following five sections. First, we present a picture gallery comparison of different final layouts produced by FADE2D and FADE3D in Section 5.4. Second in Section 5.5 we present the hard graph drawing aesthetic measures of the layouts discussed in the picture gallery. Third, we present a variety of visual précis drawings along with performance, clustering, and aesthetic measurements in Section 5.6. Fourth, we present the errors versus time taken measurements, for a range of approximations used with FADE2D in Section 5.7. And finally, in Section 5.8 we evaluate the hierarchical graph clustering measures introduced in Chapter 3.

5.4 Results: Picture Gallery

The graphs in this case study typically consist of numerous disconnected subgraphs. Although the layout of each subgraph can be computed separately, with a space decomposi-

tion per subgraph, we choose to consider the entire graph in a single space decomposition. This approach allows us to compare the performance and errors of our FADE paradigm without consideration for the graph theoretic structures in each graph. Unfortunately, this approach has serious consequences for the creation of high level visual précis, since disconnected subgraphs can be drawn with implied edges between them because of the nature of the geometric clustering method used.

This conservative approach is taken to allow us to compare the results of our clustering, aesthetic, and horizon measures between this case study and the matrix market case study in Chapter 6. We further do not show any direction in the edges, to allow a fair comparison with the drawings of undirected matrix data shown in Chapter refcase2. In practice, the utility of such drawing would be much increased by the use of edge direction information either in edge colour coding or by the use of arrows.

The drawings shown in Figures 5.2 to 5.18 are the result of applying either a two or three dimensional wavefront layout to the graph, followed by a number of iterations of FADE2D or FADE3D to further improve the drawing. Further drawings of these graphs are included in Appendix A.

Distinct edge sets, within a particular graph, are colour coded. For example, Figure 5.2 is the drawing of a *signature* view extracted from the resource flow graph of Bash. This view consists of `return-type` and `parameter-of-type` edges; each of which is drawn with a distinct colour. For the purposes of consistency the edges in these graphs are assigned uniform ideal edge lengths.
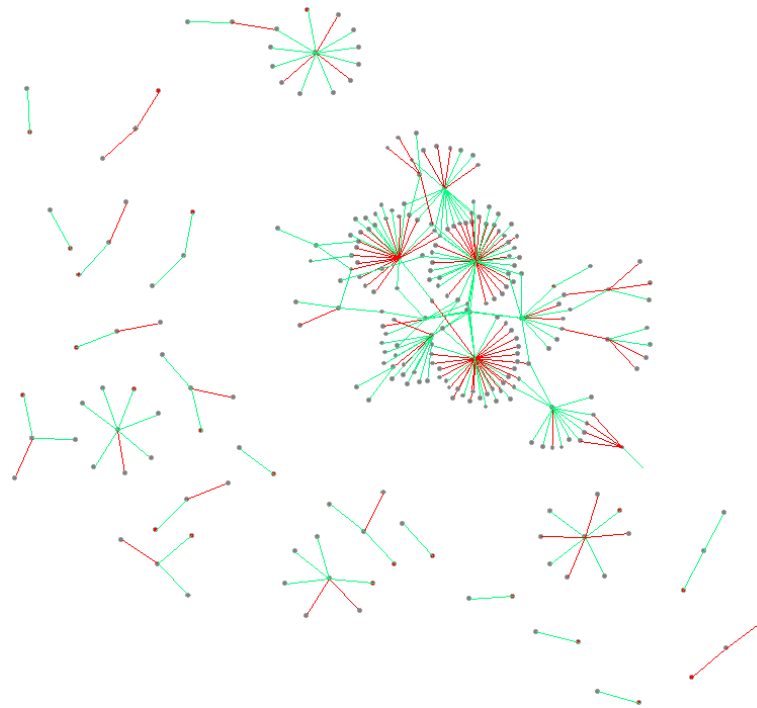
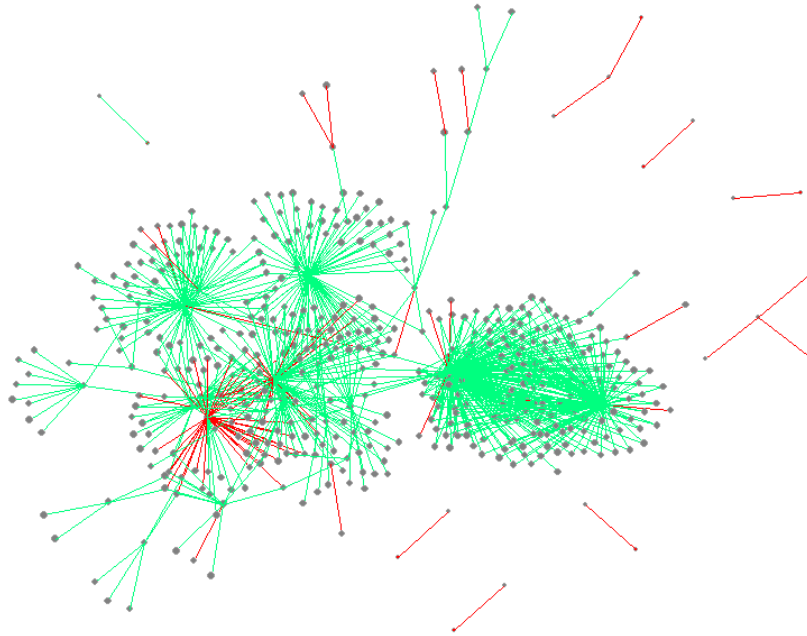**Figure 5.2:** Bash Signature View, with red `return-type` edges and green `parameter-of-type` edges

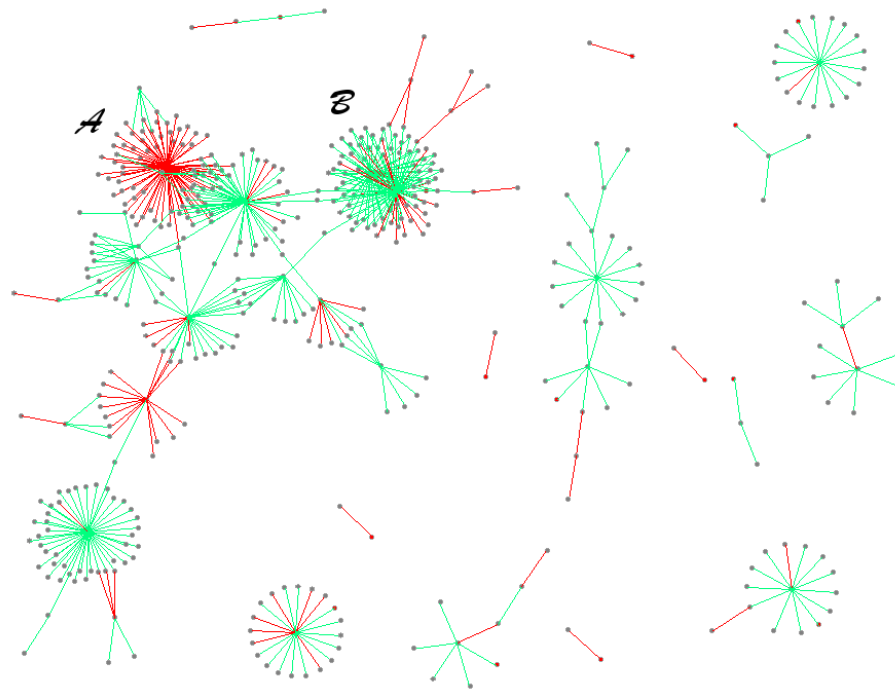**Figure 5.3:** Xaero Signature View, with red `return-type` edges and green `parameter-of-type` edges

Figure 5.4: Mosaic Signature View, with red `return-type` edges and green `parameter-of-type` edges [1]

[a]The labels $\mathcal{A}$ and $\mathcal{B}$ in this and other figures are annotation used in the discussing the figure in Section 5.4.1.
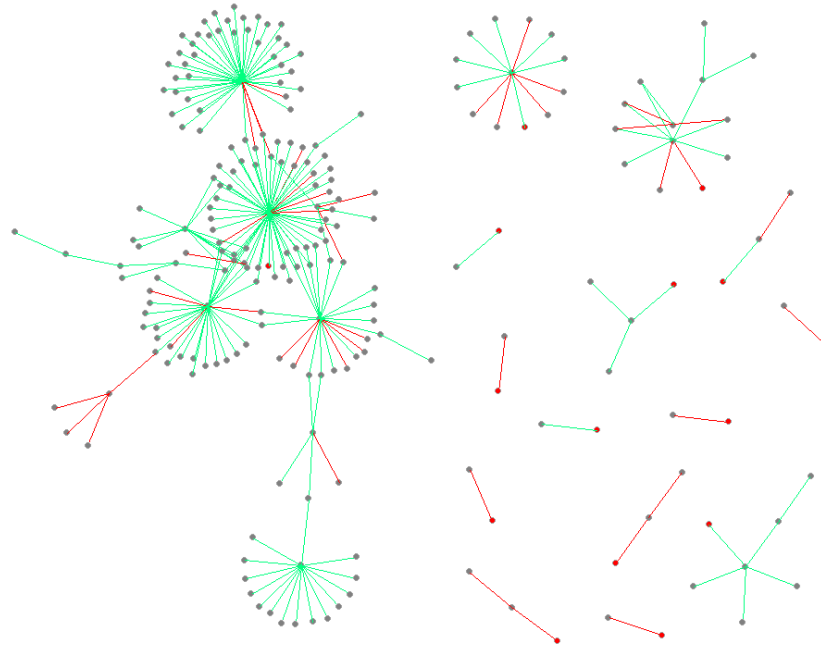
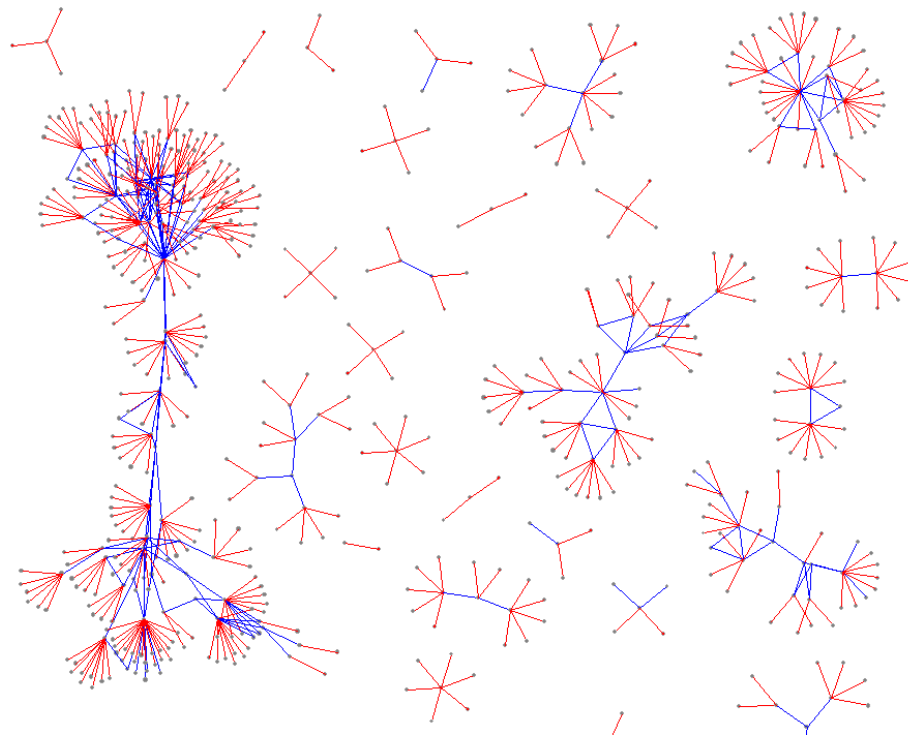**Figure 5.5:** CVS Signature View, with red `return-type` edges and green `parameter-of-type` edges

Figure 5.6: Bash Type Composite View, with red `enclosing` edges and blue `is-part-of-type` edges
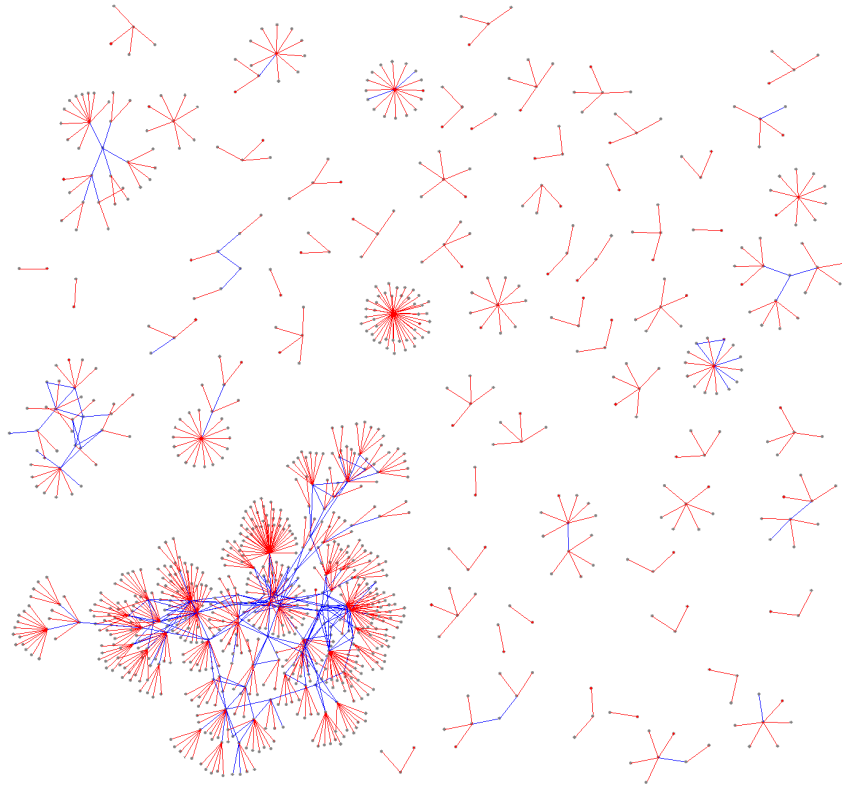
Figure 5.7: CVS Type Composite View, with red `enclosing` edges and blue `is-part-of-type` edges
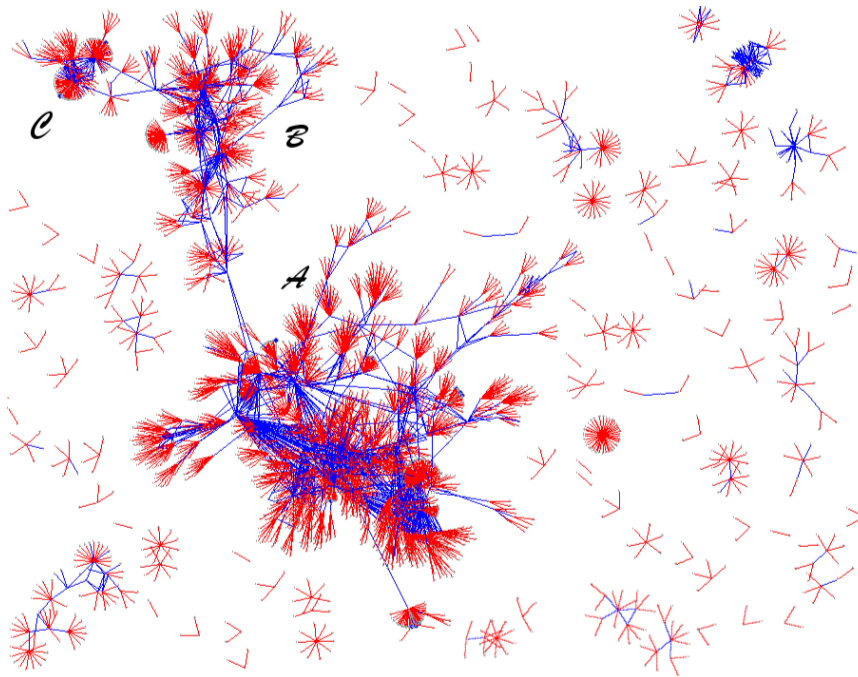

Figure 5.8: Mosaic Type Composite View, with red `enclosing` edges and blue `is-part-of-type` edges
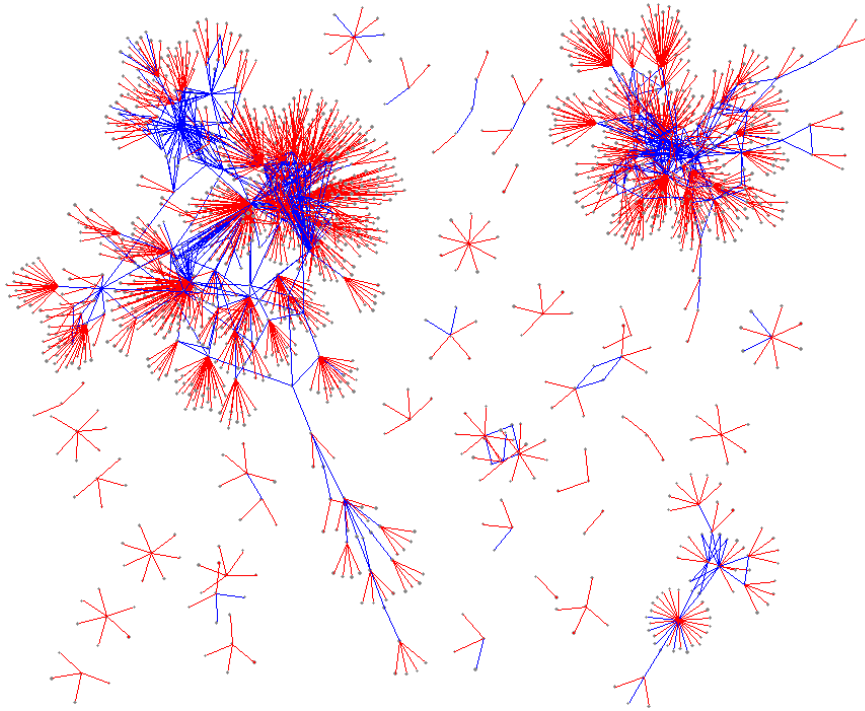
Figure 5.9: Xaero Type Composite View, with red `enclosing` edges and blue `is-part-of-type` edges
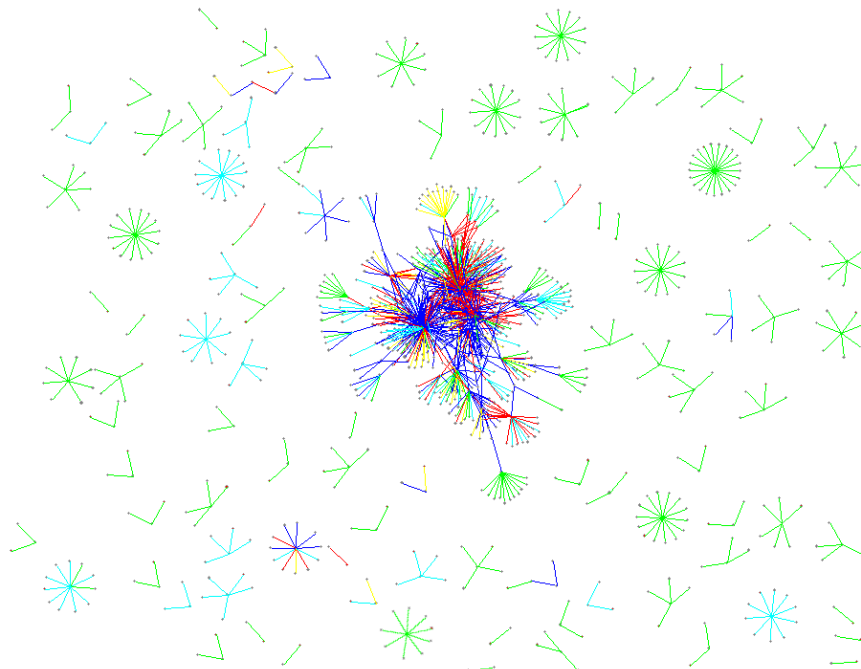
Figure 5.10: CVS Type Usage View, with red `parameter-of-type`, yellow `return-type`, green `member-of-type`, cyan `variable-of-type` and blue `local-variable-of-type`
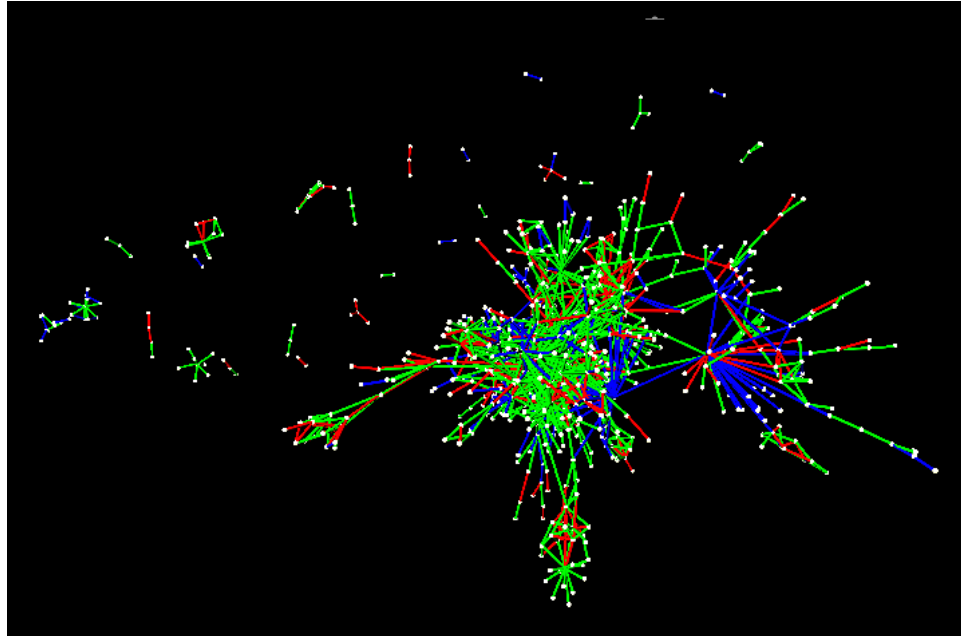
Figure 5.11: Bash Object Reference View (drawn with FADE3D), with red `set`, green `use`, and blue `address`
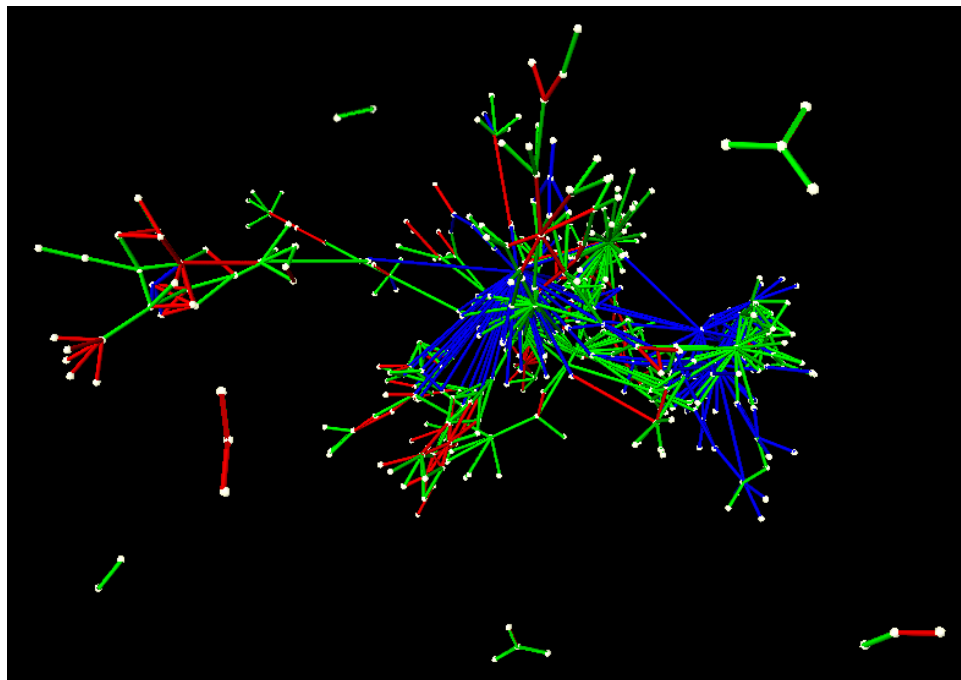
Figure 5.12: Mosaic Object Reference View (drawn with FADE3D), with red `set`, green `use`, and blue `address`

Figure 5.13: Xaero Object Reference View (drawn with FADE3D), with red `set`, green `use`, and blue `address`

Figure 5.14: CVS Same Expression View (drawn with FADE3D)

Figure 5.15: Bash Actual Parameter View (drawn with FADE3D)

Figure 5.16: CVS Dominance View
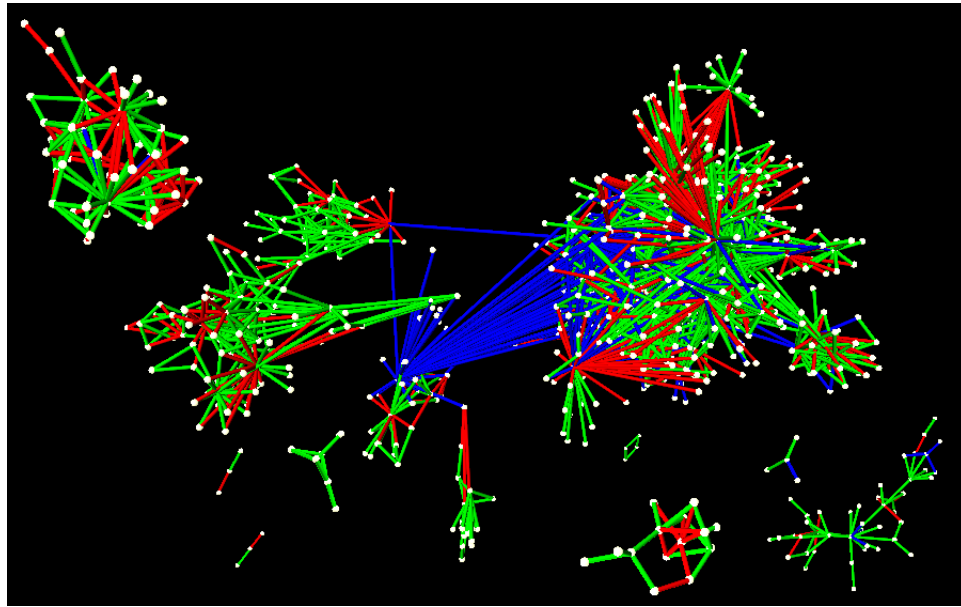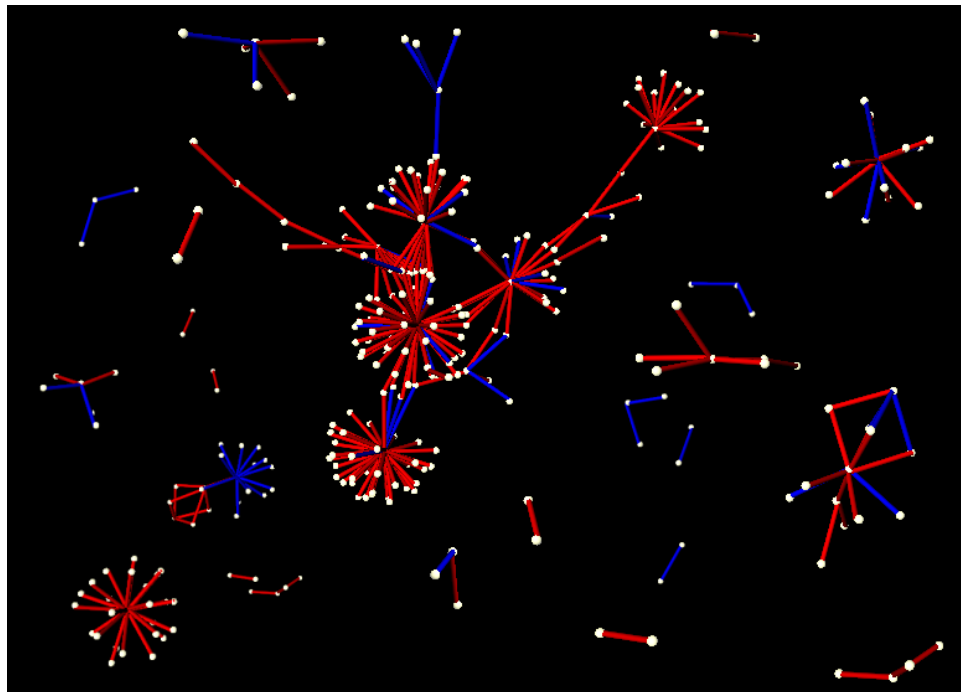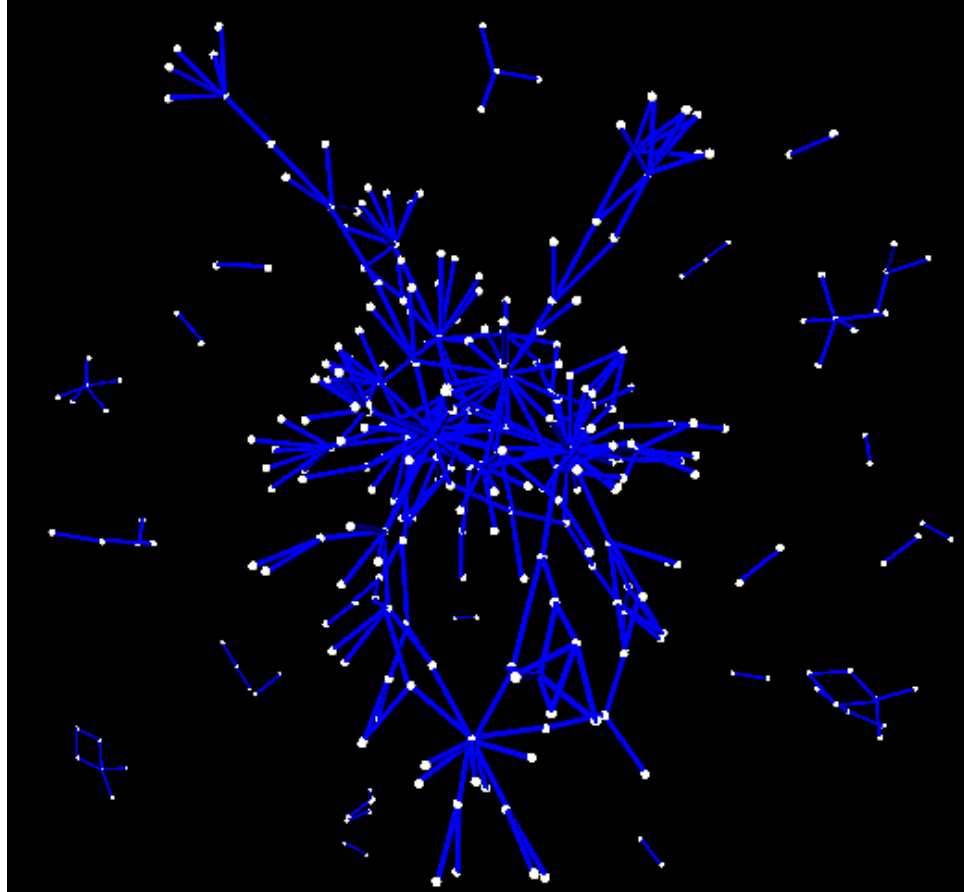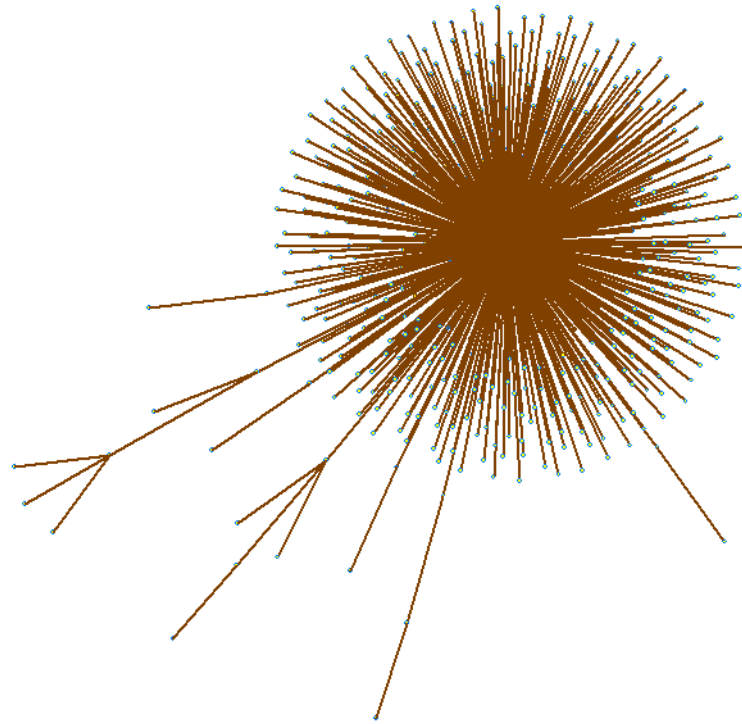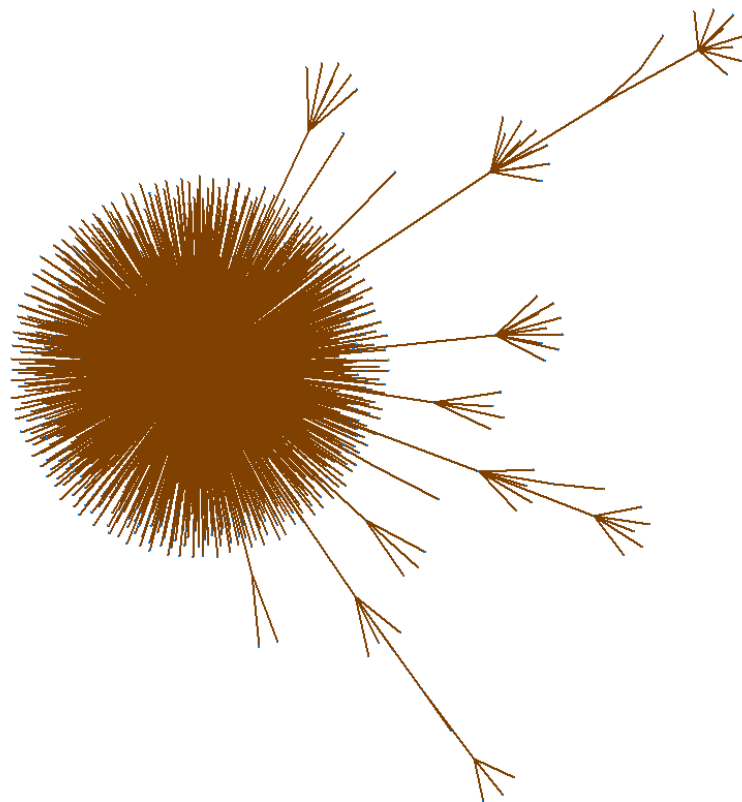

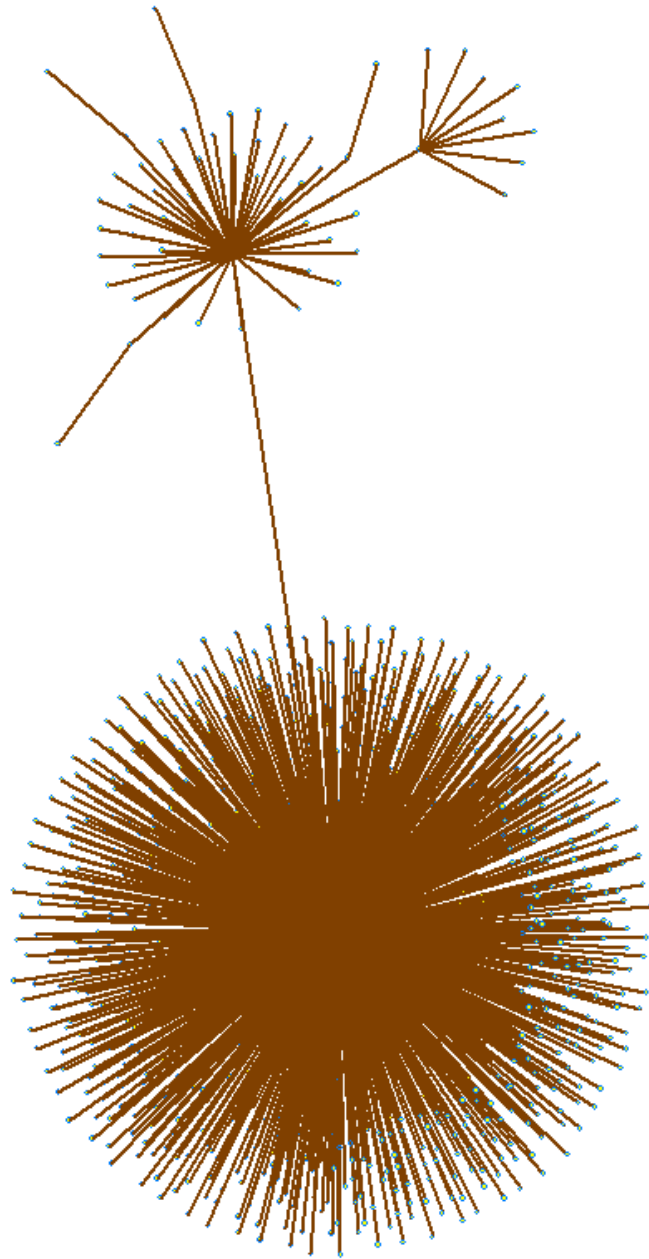**Figure 5.17:** Mosaic Dominance View

Figure 5.18: Xaero Dominance View

5.4.1 Discussion

The subgraphs extracted from the resource flow graph of each software system represent a *view* of the software. These views are used in a range of graph theoretic analyses of the software systems. We do not provide a full analysis of the knowledge gained from such visualizations, as exisiting work has shown the utility of such drawings, instead in this discussion, our aim is to evaluate, in general terms, the use of the FADE paradigm for the visualization of these extracted views.

The discussion of the results focuses on two areas:

- The appropriateness of the FADE paradigm for the visualization of each graph type.

- Discussion of individual graphs, in terms of structure, with reference to little domain knowledge.

Type Composite Views

Providing a visualization of a type composite view to a reverse engineer or program evolver should allow them to visually identify aspects of the structure of the composition of types within a particular system. The evaluation of any understanding gained is outside the scope of this thesis, instead we restrict our interest to general discussion of the potential of such visualizations.

Using the FADE visualizations we first note that the `is-part-of-type` edges form a spanning set of each sub-graph. This is an immediate and obvious observation, from each of the visualizations in Figures 5.6, 5.7, 5.8, and 5.9 of the type composite view. In each drawing the `is-part-of-type` edges form a visual backbone for the drawing, whereas the `enclosing` edges form the *visual hair* to end-nodes which are `record-components`. Nodes that are connected to such end-nodes are `type nodes`. Type nodes have an `is-part-of-type` relationship to other type nodes.

In the FADE visualization of the Type Composite View of Bash shown in Figure 5.6, a reverse engineer can easily identify a single large group of type nodes, along with several smaller groups. The large group consists of two weakly connected regions of nodes, drawn on the left. In an interactive visualization, a reverse engineer can query the names of

the nodes. An obvious starting point for such queries can be the nodes linking the two regions in the large subgraph. In this case, the type nodes connecting these regions relate to `process` and `job control`.

In the FADE visualization of the Type Composite View of CVS shown in Figure 5.7, there is one main subgraph with many `is-part-of-type` relationships. This visualization shows an example of why a reverse engineer may first filter the view. By filtering, you can effectively ignore the many small subgraphs so as to focus on the larger groups. In this case, the largest subgraph includes type nodes relating to `client-state`, `versioning` and `streaming`.

In the FADE visualization of the Type Composite View of Mosaic shown in Figure 5.8, there are two major connected groups labeled $\mathcal{A}$ and $\mathcal{B}$ connected through one node. Other groups of nodes have been labeled to facilitate discussion. Likewise, $\mathcal{B}$ and $\mathcal{C}$ are only weakly connected. As with the Bash type composite view, reverse engineers are typically interested in locating such connection points, when starting to understand the software. Such points of interest do not necessarily represent "software interfaces" but rather, are signposts a reverse engineer can follow [188].

In this case, the single node connecting $\mathcal{A}$ to $\mathcal{B}$ is a `process-id` type. The large group at $\mathcal{A}$ pertains to **XM** and **X11** types with a strong coupling to user defined types in Mosaic. **XM** is a Motif widget with functions capable of displaying HTML 3.2 and image formats such as *GIF* and *PNG*. The group denoted by $\mathcal{B}$ relates to `process control`, `file control`, `streams` and `queues`. The sub-grouping denoted by $\mathcal{C}$ contains types related to *JPEG* image handling.

The Xaero Type Composite View shown in Figure 5.9 includes large subgraphs. Such large disconnected groups are of particular interest to a reverse engineer, as they represent a very clear decoupling between the types in use. Categorization of each grouping can aid in the overall process of understanding. In this case, the large grouping on the left represents **X11** types and the large group on the right includes modeling types, such as: `vectors`, `spheres`, `cylinders`, `forces`, `material`, and `colour`.

A visualization such as that in FADE, where nodes that are adjacent are drawn close together, allows the reverse engineer to quickly identify high level groupings in the type

composition within a particular system. However, the granularity of the understanding gained from such a visualization is difficult to measure.

Type Usage Views

The appropriateness of our FADE drawing of this class of view, which contains more edges than any of the other classes, is not immediately apparent. A reverse engineer can however use such a visualization to query nodes not connected to a major group. For example, in Figure 5.10 if a variable or constant is not associated with the major component, then it typically has a `members-of-type` relationship (drawn in green) with its associated type.

Object Reference Views

Providing a visualization of an object reference view to a reverse engineer or program evolver should allow them to visually identify aspects of the structure of interactions between groups of elements within the system, at a very approximate level. Such visualizations typically exhibit large amounts of local interactions along with a few interactions to other collections of nodes. As with the type composite view, identification of the connecting nodes and categorization of the groups can form an essential part in the understanding of the system.

An example of this is shown in Figure 5.13. Appendix A includes a video of this three dimensional graph drawing, along with the model itself. Here, there is one large group, connected to two other smaller groups. Both of those groups are further connected to a third group.

The largest group, is a good example of where the FADE paradigm fails. Here there are many tightly connected nodes which results in a combinatorial subgraph structure with low diameter, which is difficult to draw using force directed methods. Although this is a poor drawing of this subgraph, overall the other subgraph structures and their interrelationships are evident in this drawing. The largest group is connected to two of the smaller groups, only by `address` edges. The smaller subgraphs consist primarily of `set` and `use` edges.

Examination of the nodes, in this example, reveals a rich array of domain specific aspects about which nodes are interacting and what the names of those nodes are. For example, the small group on the top left of Figure 5.13 contains collision detection and scene configuration subprograms.

Signature Views

Signature views provide an overview of how sophisticated or complex the type usage is within a particular program. Such visualizations aid in the identification of the structure of subprogram type usage.

Such visualizations makes apparent several aspects of the underlying code:

- The most heavily used types are easily identified.

- Sub-programs using the same types, although not named the same or even co-located in code, are drawn close together.

- Distinct sub-groupings, which use only one or two types, are drawn as disconnected graphical components.

- Symmetry or asymmetry in the edge colouring, indicates the same usage pattern of types by different sub-programs.

- Certain types are used primarily to pass information to subprograms (`paramter-of-type`), whereas others are primarily used to collect information from subprograms (`return-type`).

The *signature* views of Bash, CVS, Mosaic, and Xaero show that most subprograms use only one user defined *type* at a time, where the nodes of high degree are types. However, there are a few sub-programs that use multiple types. The visual effect called *barreling* can be seen where multiple subprograms use the same pair of types and the visual effect we call *cross baring* occurs where subprograms use three or four user defined types.

Whereas Bash uses many user defined types both to pass and collect information from sub-programs, CVS predominately uses user defined type to pass information to sub-programs. This is evidenced by the fact that over 95% of subprograms that share types,

do so with `parameter-of-type` edges only. Mosaic has 11 major types with some clearly used for passing and others for collecting information from subprograms.

Artifacts of our drawing paradigm (denoted by $\mathcal{A}$ and $\mathcal{B}$ in Figure 5.4) are aptly demonstrated in the signature view of Mosaic. The drawing of the shared types in section $\mathcal{A}$ is typical when the number of sub-programs that share two or more types represent less than half of all the sub-programs using those types. Whereas section $\mathcal{B}$ shows the effect when the majority of nodes share two or more types. Due to the forces exerted in the FADE2D algorithm, the type nodes are drawn close together in section $\mathcal{B}$ which results in a large number of edge crossings. Although aesthetically unappealing, it is visually apparent that these two types are highly related.

In this case, both types *XtPointer* and *Widget* come from the **X11** X-Windows GUI. This should not be surprising, since Mosaic is a graphical WWW client.

Xaero has a few types which are predominately used for passing information to subprograms. Here a reverse engineer can focus on the nodes connecting the sections of the graph to build an understanding of what components are in this system.

The type usage in Mosaic and Xaero indicates that vast majority of subprograms that interact with the user interface types do not use other types, this indicates low coupling. Those subprograms that do interact with other types and the user interface types may be good candidates for interface inspection.

Other Views

The other three views in this case study, namely the same expression view, the actual parameter view and the dominate view are based on either small or very simple graphs. These views are suitable for interactive querying or filtering to locate a node or set of nodes. In terms of aiding in identifying structures in the code it is unclear if these views help.

Conclusions

As discussed in Section 2.2.6, the FADE paradigm tends to produce drawings where the natural clusters and structures are visually apparent. For a reverse engineer, each view

Graph Name	Nodes	Edges	Crossings	Aspect Ratio	Min—d / Max-dim	Min—d / AvgEdge	MinEdge / MaxEdge	MinEdge/ AvgEdge
bash Signature view	279	286	39	1.037	.002	.075	.431	.706
cvs Signature view	299	307	32	1.041	.001	.14	.453	.673
mosaic Signature view	491	568	1045	1.308	.001	.094	.154	.271
xaero Signature view	481	803	7632	1.238	0.	.028	.045	.12
bash Type Composite View	749	767	727	1.334	0.	.006	.11	.3
cvs Type Composite View	1139	1137	1195	1.307	.001	.147	.134	.307
mosaic Type Composite View	4642	5197	36859	1.2	.001	.048	.013	.096
xaero Type Composite View	1804	2069	11518	1.099	0.	.005	.032	.133
bash Type Usage View	837	882	4304	1.035	.001	.111	.054	.157
cvs Type Usage View	1087	1232	7634	1.181	.001	.088	.032	.119
mosaic Type Usage View	3087	3120	18076	1.051	0.	.012	.019	.066
xaero Type Usage View	1960	2367	30318	1.113	0.	.012	.02	.083
bash Object Reference View	749	1275	10922	1.087	0.	.02	.039	.146
cvs Object Reference View	820	1386	17323	1.532	.001	.038	.02	.062
mosaic Object Reference View	446	725	3069	1.065	.001	.02	.015	.075
xaero Object Reference View	863	2690	121488	1.16	0.	.006	.004	.032
bash Same Expression View	171	224	204	1.011	.012	.288	.132	.34
cvs Same Expression View	237	414	625	1.078	.005	.116	.056	.116
mosaic Same Expression View	24	12	0	1.373	.058	.988	.983	.988
xaero Same Expression View	244	430	679	1.161	.004	.088	.033	.098
bash Actual Parameter View	297	349	424	1.021	.003	.071	.124	.313
cvs Actual Parameter View	401	630	2563	1.	.001	.011	.026	.073
mosaic Actual Parameter View	59	68	61	1.039	.019	.307	.177	.307
xaero Actual Parameter View	257	299	173	1.036	.01	.192	.078	.256

Table 5.3: Graph Drawing Aesthetic Measures of the final drawings

provides a unique insight into specific aspects of the structure of the system. Of particular note are the natural clusterings and colour symmetries exhibited in the signature view and the high level interaction groups in the object reference views.

5.5   Results: Graph Drawing Aesthetics

Table 5.3 gives the graph drawing aesthetic measures for the underlying graph drawings shown in the picture gallery in Section 5.4.

5.5.1   Discussion

Table 5.3 indicates that object reference views consist of many strongly connected sets of nodes, which result in strongly clustered groups with many edge crossings in the two

dimensional drawings. Further, the drawings are typically displayed with a uniform aspect ratio.

Due to the nature of the force model used in the FADE paradigm, disconnected subgraphs force each other apart slowly over many iterations. As such, iterating FADE a large number of times quickly reduces the ratio of the minimum inter node distance to the maximum extent of the drawing. Although each edge has the same ideal length, the examples in this case study show that the effect of the interacting forces can result in some edges being elongated, see for example Figure 5.13. Table 5.4 shows that, depending on the structure of the graph, some edges can be much longer than either the average or minimum edge length. This negatively impacts on the goal that nodes with an edge between them should be drawn close together. However, as Table 5.4 shows, the minimum edge length is typically close to the average edge length in the drawing.

5.6 Results: Horizon Drawings (*Visual Précis*)

Figures 5.19 to 5.21 show example visual précis extracted from a hierarchical compound graph of different *views*. The aim here is to assess whether such abstract representations can be used by reverse engineers in studying the structures exhibited in the different visualizations of the views. These graphs have highly connected subgraphs which are difficult to draw using a force directed approach. Drawings of such subgraphs typically consist of many nodes and edges packed into a small area or volume. The use of a pseudonode to approximate a highly connected subgraph often removes unnecessary detail while retaining the overall visual shape of the structure that the reverse engineer wants to investigate.

As described in Section 3.11, horizon views provide a systematic means to investigate different levels of abstraction of the hierarchical compound graph. Other views, such as surface views or bell curve views, are suitable for interactive graph exploration, where a small set of nodes must be drawn in detail along with their overall shared global neighbourhood.

The discussion of the underlying drawings, in the previous section, emphasized the need for an interactive visualization system. This system should support filtering, node
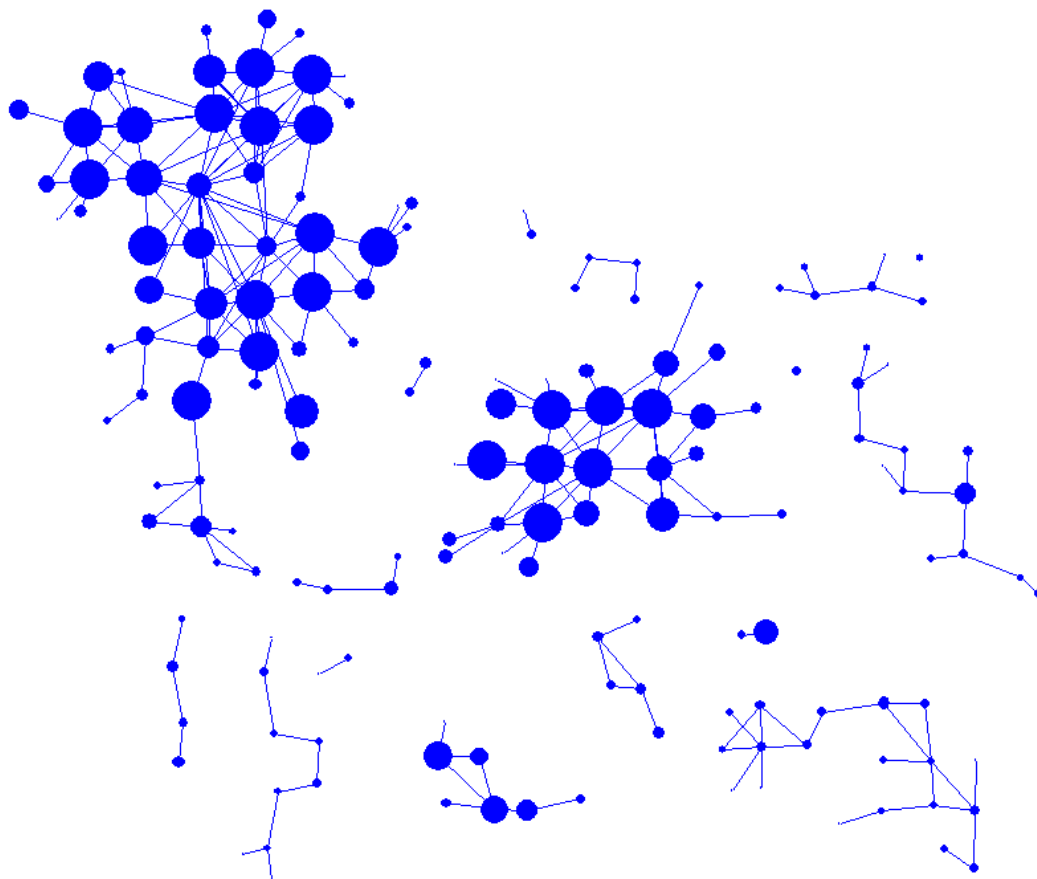
Figure 5.19: Xaero *Visual Précis* of a Type Composite View

querying, and the ability to focus in on a small set of nodes; all these features can be supported with different types of précis, as described in Section 3.11.

The number of possible précis that can be extracted from a hierarchical compound graph is very large. Here we restrict our interest to the horizon level précis and the drawings of the underlying graphs.

Table 5.4 shows the results of applying a range of graph drawing aesthetic measures to each horizon level of the hierarchical compound graph. These measures are averaged over 100 FADE drawings of each graph. The number of clusters $\mathcal{C}$, implied edges $\mathcal{I}$, nodes $\mathcal{N}$ and edges $\mathcal{E}$ is shown for each level, including the lowest level (that is, the entire graph). Recall that the visual weight is the ratio of the sum of these four values to the number of nodes and edges in the underlying graph. The $Min\text{-}d$ is the minimum distance between any two nodes in the drawing of the graph.
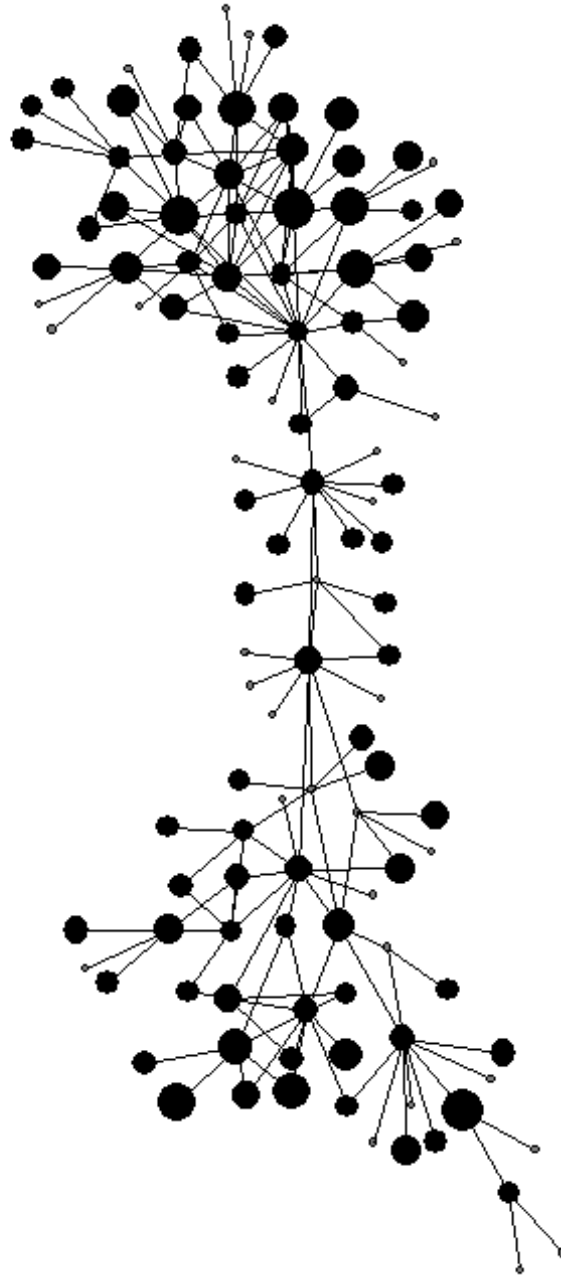
Figure 5.20: Section of a Bash *Visual Précis* of a Type Composite View
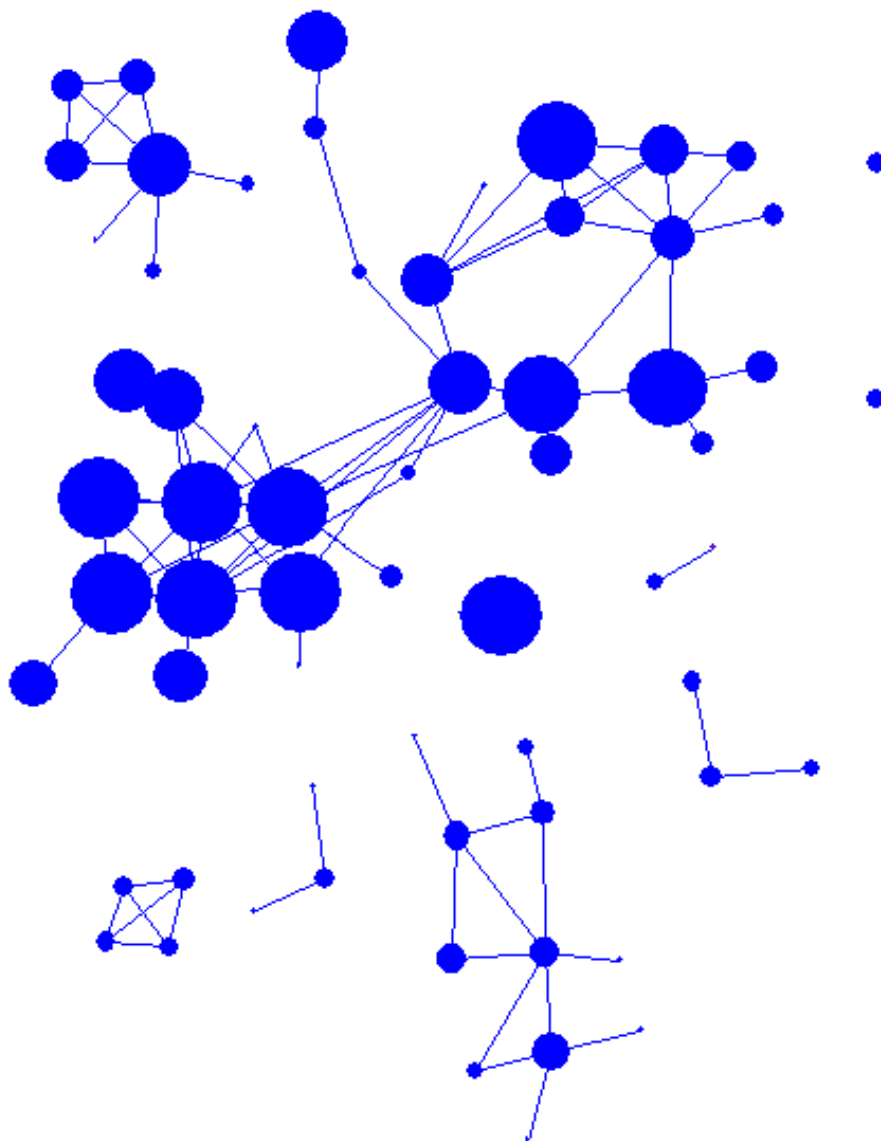
Figure 5.21: Xaero *Visual Précis* of an Object Reference View

Table 5.4: Graph Drawing Aesthetic Measures of Visual Horizons of Various Views from Bash, CVS, Mosaic and Xaero

Graph Name	Horizon Level ($\mathcal{HL}$)	Clusters ($\mathcal{C}$)	Implied Edges ($\mathcal{I}$)	Nodes ($\mathcal{N}$)	Edges ($\mathcal{E}$)	% Visual Weight ($\mathcal{VW}$)	Crossings ($\mathcal{X}$)	Aspect Ratio ($\mathcal{AR}$)	Min–d / Max-dim ($\mathcal{MD}$)	Min–d / AvgEdge ($\mathcal{ME}$)	MinEdge / MaxEdge ($\mathcal{MM}$)	MinEdge/ AvgEdge ($\mathcal{MA}$)
Mosaic	2	4	2	0	0	.6	0	1.088	.485	.631	.461	.631
signature	3	15	7	0	0	2.1	0	1.449	.03	.174	.072	.174
view	4	31	13	3	0	4.4	0	1.251	.016	.225	.115	.225
	5	47	30	16	1	8.9	0	1.304	.012	.384	.162	.384
	6	57	59	36	7	15.	2	1.306	.005	.226	.256	.538
	7	81	128	84	25	30.	13	1.309	.004	.25	.185	.437
	8	90	79	194	138	47.3	12	1.308	.001	.095	.19	.396
	9	76	71	330	323	75.5	232	1.308	.002	.139	.242	.424
	10	2	4	487	560	99.4	1037	1.308	.002	.14	.154	.271
	11	0	0	491	568	100.	1045	1.308	.001	.094	.154	.271
Xaero	2	4	2	0	0	.2	0	1.588	.241	.554	.383	.554
Object	3	9	4	0	0	.4	0	1.278	.097	.601	.416	.601
Reference	4	16	8	1	0	.7	0	1.167	.043	.501	.291	.501
View	5	27	32	6	2	1.9	10	1.181	.009	.142	.05	.142
	6	56	133	18	2	5.9	195	1.166	.007	.172	.046	.204
	7	140	462	41	3	18.2	2598	1.152	.007	.227	.045	.256
	8	218	1083	252	167	48.4	17268	1.153	.002	.089	.024	.171
	9	82	768	670	1046	72.2	32388	1.16	.001	.042	.008	.068
	10	26	496	807	1872	90.1	75400	1.16	.001	.027	.006	.045
	11	8	105	847	2489	97.1	104822	1.16	0.	.006	.004	.032
	12	0	0	863	2690	100.	121488	1.16	0.	.006	.004	.032
Mosaic	2	3	2	0	0	.1	0	1.364	.514	.813	.685	.813
Type	3	8	7	0	0	.2	0	1.206	.11	.521	.424	.521
Composite	4	19	18	0	0	.4	1	1.15	.018	.197	.096	.197
View	5	43	46	0	0	.9	1	1.186	.011	.21	.176	.323
	6	97	113	4	0	2.2	17	1.185	.009	.315	.125	.422
	7	240	375	34	0	6.6	176	1.191	.003	.141	.063	.307
	8	566	1088	201	47	19.3	1796	1.194	.002	.152	.031	.207
	9	1042	1998	798	476	43.8	10062	1.201	.001	.116	.031	.242
	10	773	1736	3049	2285	79.7	18054	1.2	.001	.061	.016	.125
	11	0	0	4642	5197	100	36859	1.2	.001	.048	.013	.096
Xaero	2	4	3	0	0	.2	0	1.007	.383	.536	.772	.883
Type	3	9	7	2	0	.5	0	1.074	.095	.438	.296	.438

Table 5.4: *continued*

Graph	$\mathcal{L}$	$\mathcal{C}$	$\mathcal{I}$	$\mathcal{N}$	$\mathcal{E}$	$\mathcal{VW}$	$\mathcal{X}$	$\mathcal{AR}$	$\mathcal{MD}$	$\mathcal{ME}$	$\mathcal{MM}$	$\mathcal{MA}$
Composite	4	23	17	2	0	1.1	1	1.16	.024	.325	.186	.325
View	5	55	46	3	0	2.7	2	1.094	.012	.245	.277	.463
	6	110	122	25	0	6.6	11	1.101	.01	.388	.142	.389
	7	185	320	112	22	16.5	92	1.087	.005	.284	.117	.417
	8	313	612	314	230	37.9	1148	1.099	.002	.132	.083	.336
	9	437	714	805	566	65.1	3790	1.099	.001	.11	.048	.19
	10	11	27	1782	2017	99.1	10837	1.099	.001	.079	.032	.133
	11	0	0	1804	2069	100.	11518	1.099	0.	.005	.032	.133
Xaero	2	3	1	0	0	.1	0	1.223	.587	.827	1.	1.
Type	3	7	3	0	0	.2	0	1.18	.194	.897	.846	.897
Usage	4	15	8	0	0	.5	0	1.125	.039	.433	.268	.433
View	5	31	22	3	0	1.3	1	1.108	.01	.242	.244	.496
	6	65	56	7	0	3.	3	1.108	.009	.336	.212	.377
	7	139	134	24	0	6.9	20	1.114	.003	.187	.124	.37
	8	239	308	144	21	16.4	347	1.109	.002	.192	.076	.286
	9	249	487	511	324	36.2	2512	1.114	.001	.175	.071	.315
	10	437	759	829	533	59.	8763	1.113	.001	.09	.031	.141
	11	66	531	1824	1603	92.8	29803	1.113	0.	.034	.023	.091
	12	0	0	1960	2367	100.	30318	1.113	0.	.012	.02	.083
CVS	2	4	6	0	0	1.	1	1.046	.794	.789	.612	.789
Actual	3	16	17	0	0	3.2	1	1.112	.118	.448	.271	.448
Parameter	4	43	47	6	2	9.4	3	1.003	.034	.331	.19	.331
View	5	76	134	36	7	24.4	81	1.01	.026	.378	.183	.379
	6	94	305	158	87	62.	718	1.	.009	.178	.069	.178
	7	23	144	351	393	87.8	1481	1.	.004	.08	.068	.188
	8	4	16	391	602	97.6	2376	1.	.001	.011	.026	.073
	9	2	5	398	624	99.1	2547	1.	.001	.011	.026	.073
	10	0	0	403	632	100.	2563	1.	.001	.011	.026	.073
Bash	2	4	1	0	0	.2	0	1.659	.194	1.	1.	1.
Object	3	12	5	0	0	.8	0	1.044	.144	.696	.473	.696
Reference	4	26	20	4	0	2.5	1	1.014	.04	.352	.195	.352
View	5	55	66	15	1	6.8	20	1.061	.011	.173	.105	.248
	6	104	227	35	4	18.3	300	1.091	.012	.288	.115	.288
	7	174	595	171	93	51.	3214	1.087	.007	.245	.102	.339
	8	92	475	552	536	81.8	5031	1.087	.002	.1	.068	.242
	9	7	56	735	1197	98.6	9990	1.087	.001	.059	.039	.147
	10	1	32	747	1242	99.9	10869	1.087	.001	.037	.039	.146
	11	0	0	749	1275	100.	10922	1.087	0.	.02	.039	.146

5.6.1 Discussion

The quality of these horizon drawings are measured using a variety of graph drawing aesthetic measures. These abstract representations should also adhere to the multilevel aesthetic measures described in Section 2.2.2, namely:

- Minimize the introduction of edge crossings by abstraction.

- Glyphs representing groups of nodes should not overlap.

- Minimize the variance in abstraction aspect ratios.

From these results, it is clear that abstraction typically greatly reduces the number of edge crossings as many edges which cause crossings are elided in such drawings. The glyphs used to represent each pseudo-node are constrained to have a radius of at most half the width of the cell. This constraint ensures that no two glyphs, in two or three dimensions overlap in a visual précis. From Table 5.4, we see that the aspect ratio of each horizon level remains approximately uniform for each graph.

The results support the notion that higher level visual précis of horizons are better graph drawings than the drawings of the underlying graphs. Typically, for each graph the number of crossings in the higher level visual précis is much smaller than the number in the drawing of the graph.

Some remarks are in order.

- As with any abstraction, the higher level précis do contain higher level structural information of the software, and this is well displayed by the visualization.

- In an interactive system a user may define the maximum number of graphical elements or visual weight they require the initial visual précis to contain. The visual weight measure can therefore can be used to determine which horizon level is initially drawn.

- Table 5.4 also indicates that the bottom levels of the tree show the greatest amount of node clustering, which the is the intuition behind our $\mathcal{LCA}$ measure, the results of which are presented in Section 5.8 below.
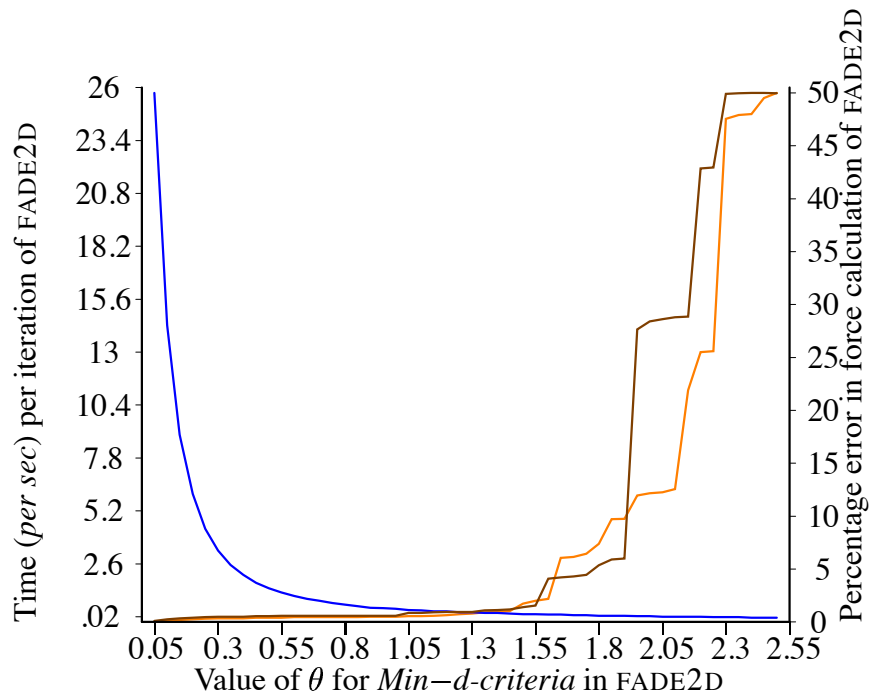
Figure 5.22: Performance versus Error of FADE2D for *TUV* of Mosaic

5.7   Results: Time and Error Performance

Here we present the results of a performance and error analysis of the FADE2D algorithm with a Min−distance cell opening criterion and varying values of θ on the range of views extracted from the resource flow graphs in this case study.

Figures 5.23 to 5.40 show charts of the performance versus error tables, for some of the graphs in this case study, the remainder are included in Appendix B. Figure 5.22 shows a representative chart. The following three measures appear in each chart:

- The time per iteration of FADE2D is shown as the blue line .

- The nonedge error as compared with the direct *node-to-node* nonedge force calculation is shown as the orange line . These results are averaged in two ways. First the error, for a given θ, is averaged over a sampling of the application of FADE2D from the first 400 runs. Second, the initial layout is randomized 100 times and the above average is computed again for each initial layout.

- The brown line is the averaged error for the first 400 iterations of FADE2D.
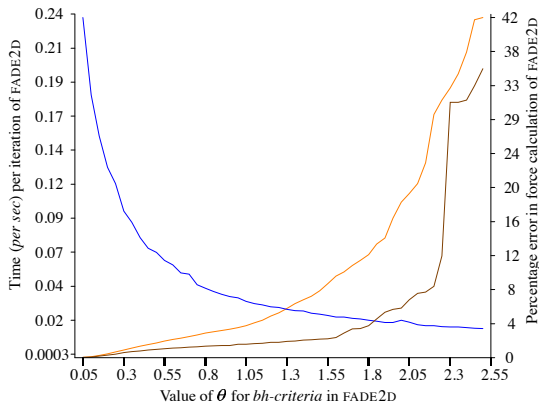
Figure 5.23: Performance versus Error of FADE2D for *APV* of Bash
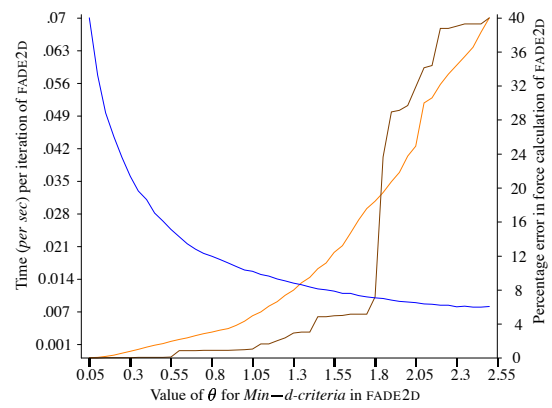
Figure 5.24: Performance versus Error of FADE2D for *SEV* of Bash
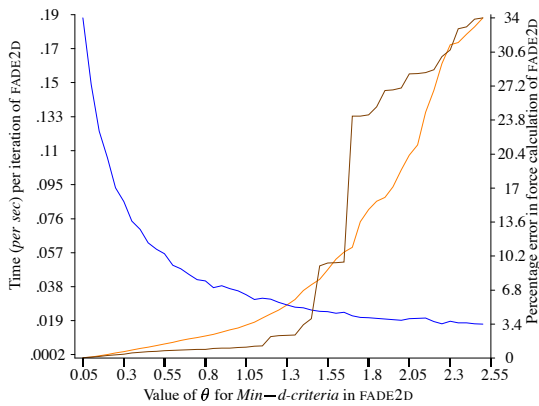
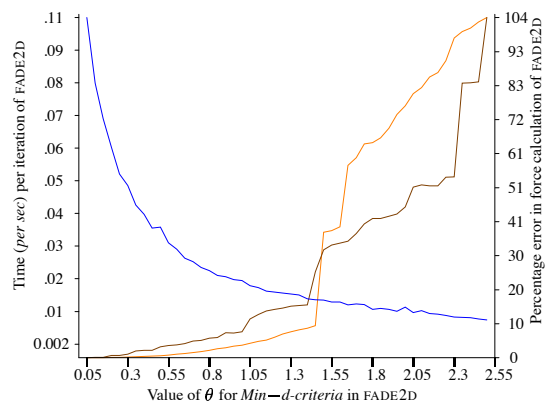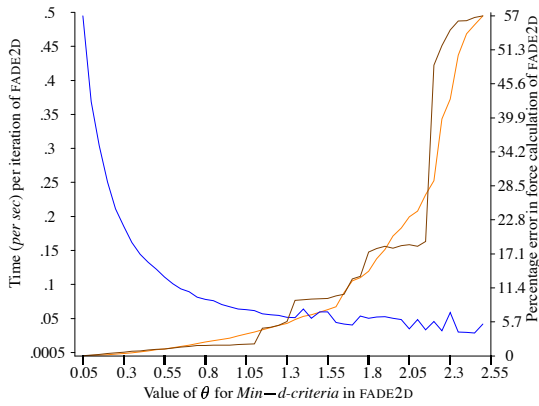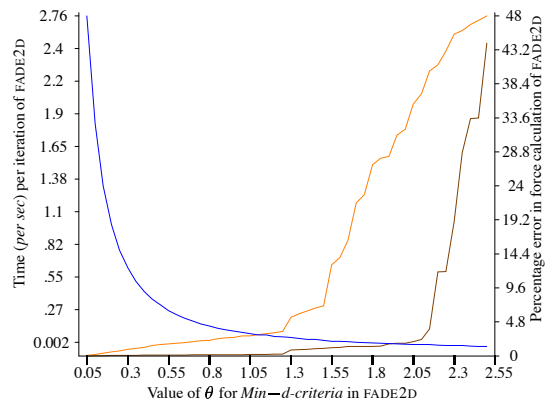**Figure 5.25:** Performance versus Error of FADE2D for *SIG* of Bash
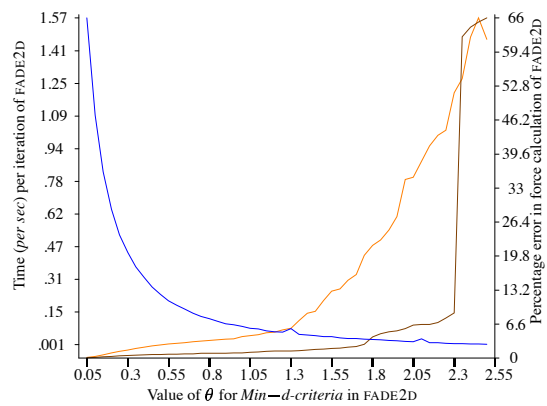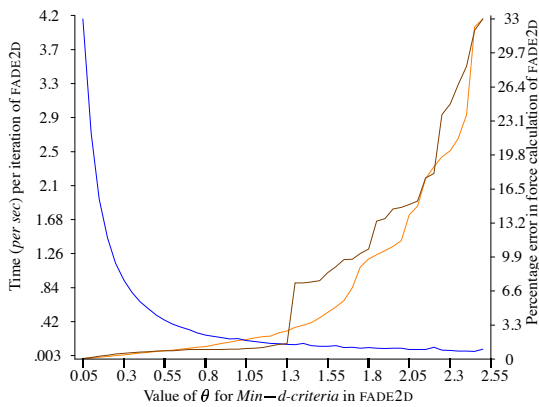

Figure 5.26: Performance versus Error of FADE2D for *ORV* of Bash
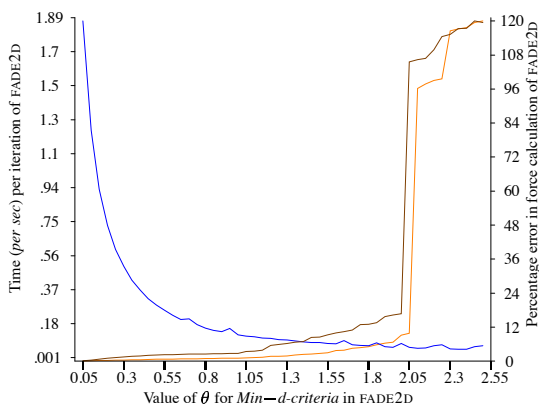
Figure 5.27: Performance versus Error of FADE2D for *TUV* of Bash
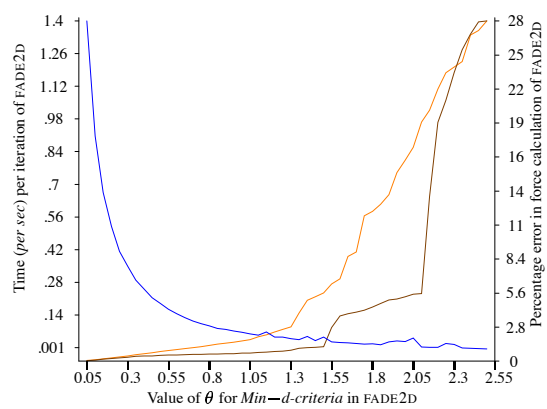
Figure 5.28: Performance versus Error of FADE2D for *TCV* of Bash
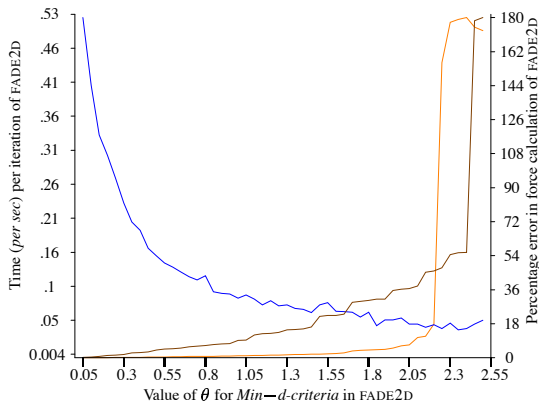
Figure 5.29: Performance versus Error of FADE2D for *APV* of CVS

Figure 5.30: Performance versus Error of FADE2D for *SEV* of CVS

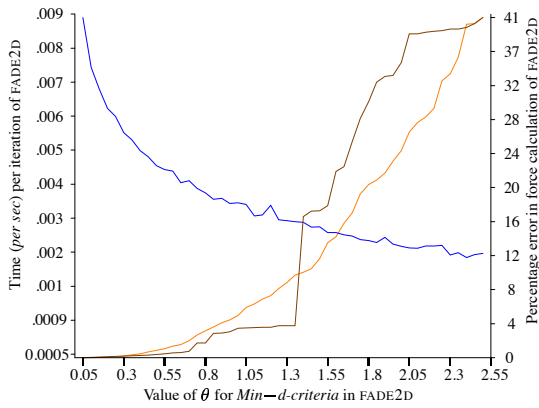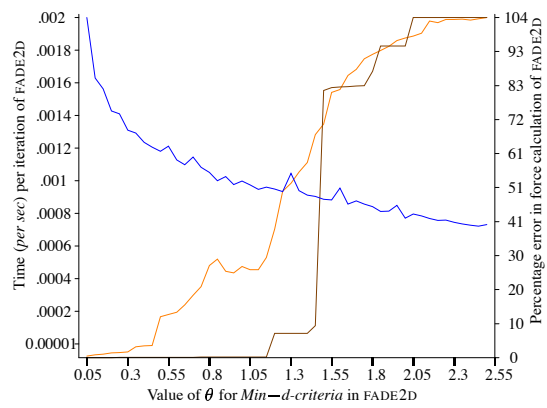**Figure 5.31:** Performance versus Error of FADE2D for *SIG* of CVS

Figure 5.32: Performance versus Error of FADE2D for *ORV* of CVS

Figure 5.33: Performance versus Error of FADE2D for *TUV* of CVS
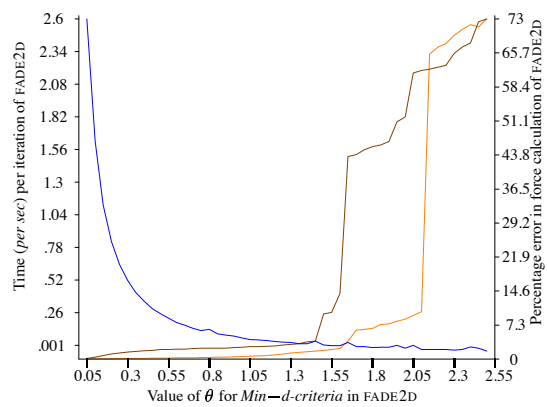
Figure 5.34: Performance versus Error of FADE2D for *TCV* of CVS

Figure 5.35: Performance versus Error of FADE2D for *APV* of Mosaic

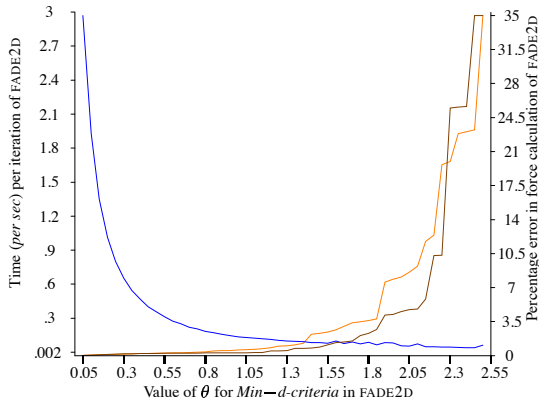**Figure 5.36:** Performance versus Error of FADE2D for *SEV* of Mosaic

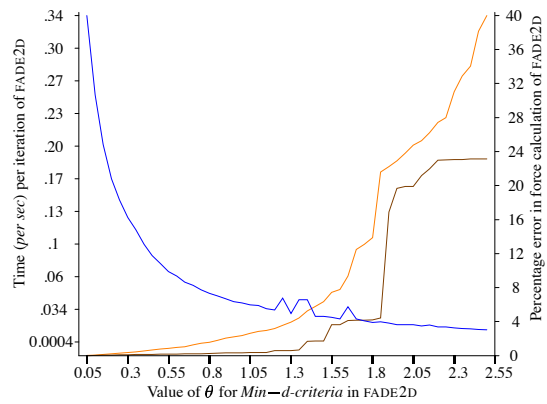**Figure 5.37:** Performance versus Error of FADE2D for *SIG* of Mosaic

Figure 5.38: Performance versus Error of FADE2D for *ORV* of Mosaic
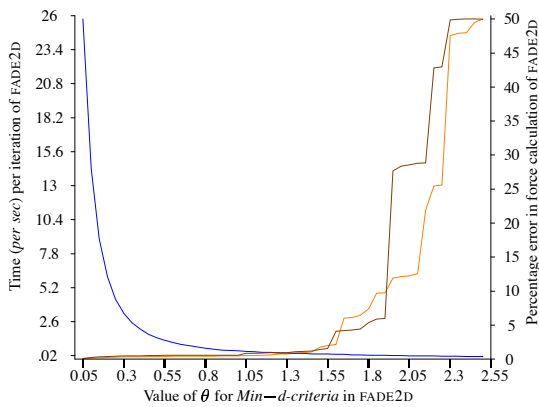

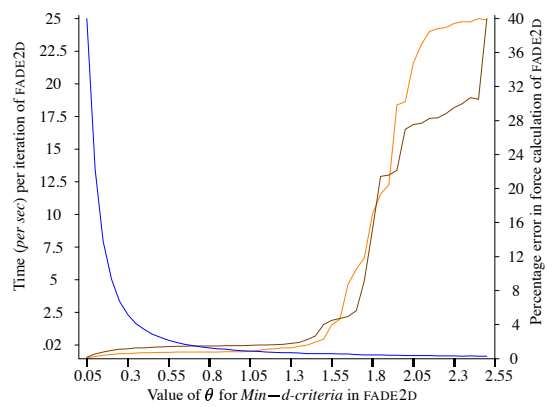**Figure 5.39:** Performance versus Error of FADE2D for *TUV* of Mosaic

Figure 5.40: Performance versus Error of FADE2D for *TCV* of Mosaic

5.7.1 Discussion

The performance of FADE2D improves as the value for θ decreases, due the more approximate nonedge force calculations and less direct *node-to-node* nonedge force calculations. However, the error in the overall force calculation also increases as θ increases. These values are in line with the predicted values from particle simulation work. FADE is not suitable for small graphs, such as Figures 5.35 and 5.36 where the error rises dramatically at a value of θ which is proportional to the maximum dimension and the length of the ideal edge. This sudden jump is less pronounced in the slightly larger graphs. It should be noted that for larger values of θ the error increases but this is not strictly correlated with any lack of improvement of the layout.

Given that the edge forces dampen these nonedge force errors, it is often permissible to have a value of θ in the range 1.3-1.8. Such error rates are catastrophic in particle simulation and lead to highly inaccurate final positions for the particles in the simulation.

We can deduce a clear guideline for graph drawing systems using the FADE paradigm: they should allow the user to alter various parameters, such as the value of θ, the cell opening criterion used, the dimension, and the arity of the space decomposition.

5.8 Results: Clustering Measures

Here we present the results of applying the hierarchical compound graph quality measures, introduced in Section 3.5, to layouts produced across the first 300 iterations of the FADE2D algorithm. Here we use the $Min\text{-}d$ cell opening criterion and a value of $\theta = 0.8$ on the range of views extracted from the resource flow graphs in this case study.

Figure 5.46 to Figure 5.53 show charts of the normalized clustering measures versus crossings, for some of the graphs in this case study. Further charts are shown in Appendix B.

Figure 5.41 shows an example of one of the charts from the results of applying these measures to a graph in this case study. These results are averaged over 40 sets of runs of the FADE2D algorithm. In each case, the graph is given an initial random layout to ensure
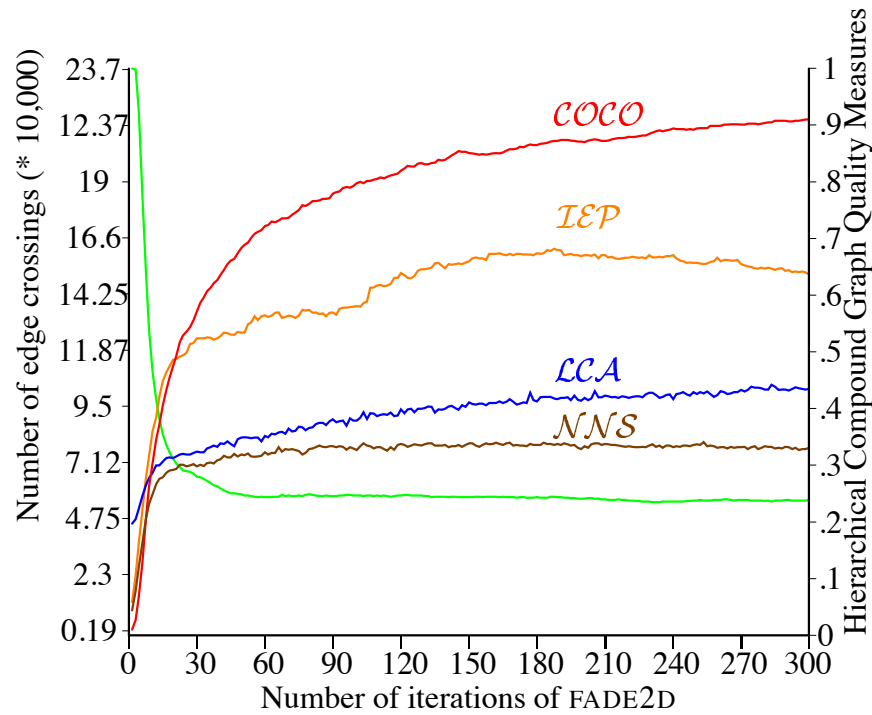
Figure 5.41: *HCGQM* versus Crossingsfor *ORV* (Mosaic)

a uniform starting point for each layout process.

The values of hierarchical clustering quality measures are indicated as normalised values according to the right hand axis. Each measure is colour coded as follows:

- The Implied Edge Precision ($\mathcal{IEP}$) measure is drawn with an orange line.

- The Lowest Common Ancestor ($\mathcal{LCA}$) measure is drawn with an blue line.

- The Coupling and Cohesion ($\mathcal{COCO}$) measure is drawn with an red line.

- The Node Neighbourhood Similarity ($\mathcal{NNS}$) measure is drawn with a brown line.

The number of edge crossings in the layout is indicated on the left hand axis and is drawn as a green line in each chart.
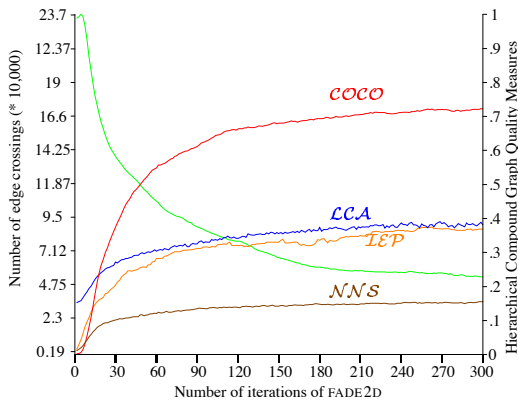
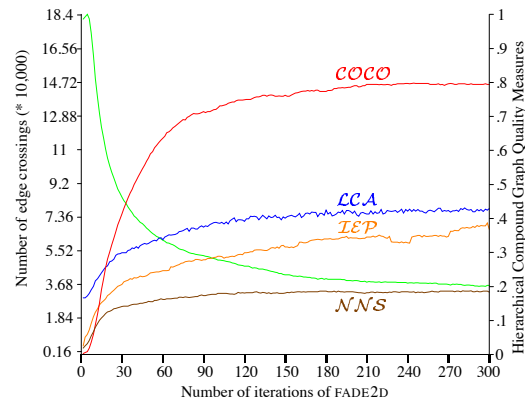**Figure 5.42:** *HCGQM* versus Crossings for *TCV* (Mosaic)
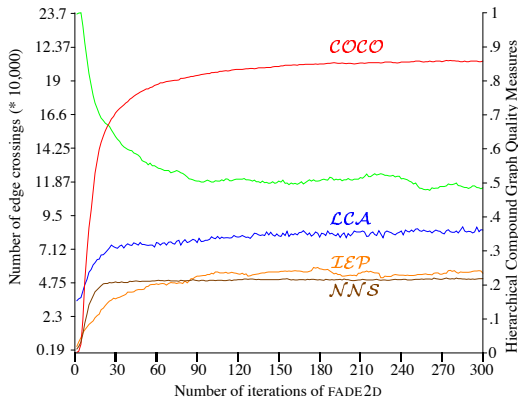

Figure 5.43: *HCGQM* versus Crossings for *TCV* (Xaero)

Figure 5.44: *HCGQM* versus Crossings for *TUV* (Mosaic)
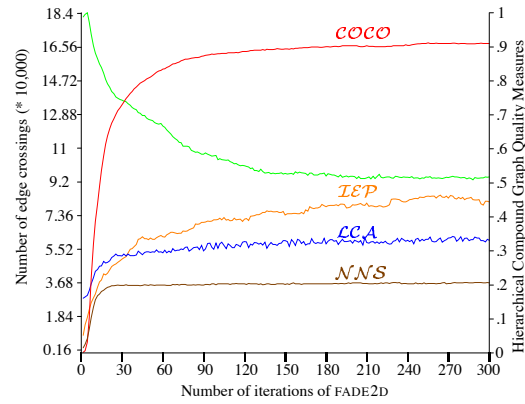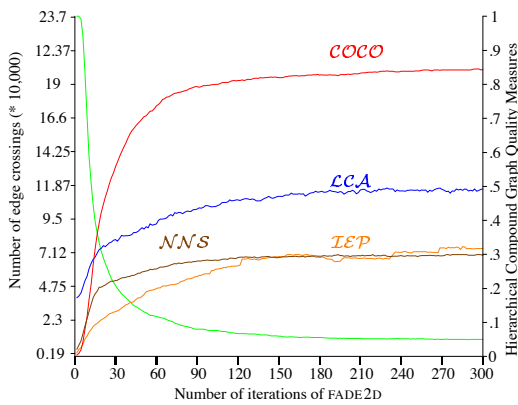

Figure 5.45: *HCGQM* versus Crossings for *TUV* (Xaero)

Figure 5.46: *HCGQM* versus Crossings for *TCV*(CVS)
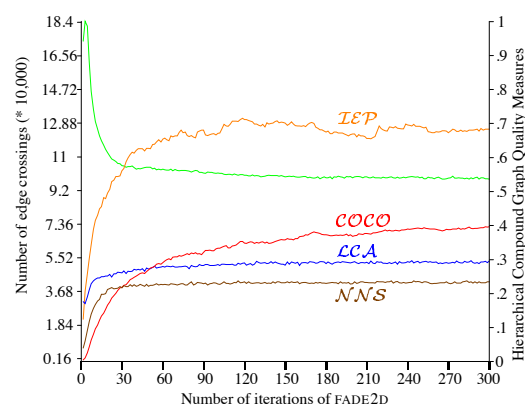
Figure 5.47: *HCGQM* versus Crossings for *ORV* (Xaero)

Figure 5.48: *HCGQM* versus Crossings for *SIG* (Mosaic)

Figure 5.49: *HCGQM* versus Crossings for *SIG* (Xaero)

Figure 5.50: *HCGQM* versus Crossings for *SEV* (Mosaic)

Figure 5.51: *HCGQM* versus Crossings for *SEV* (Xaero)
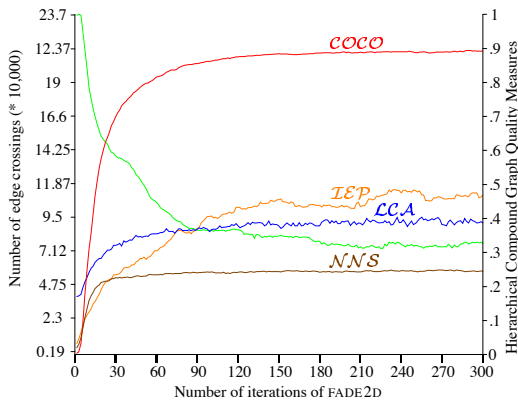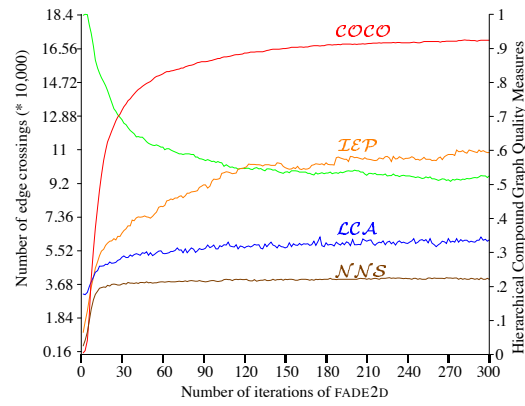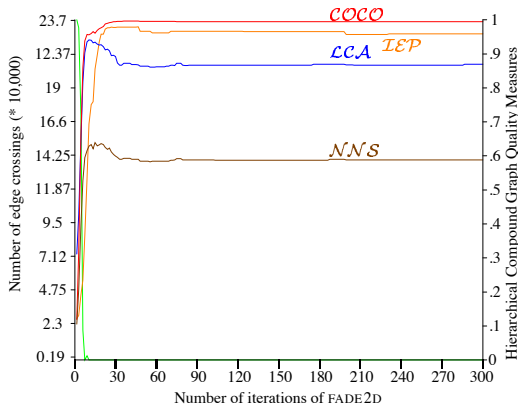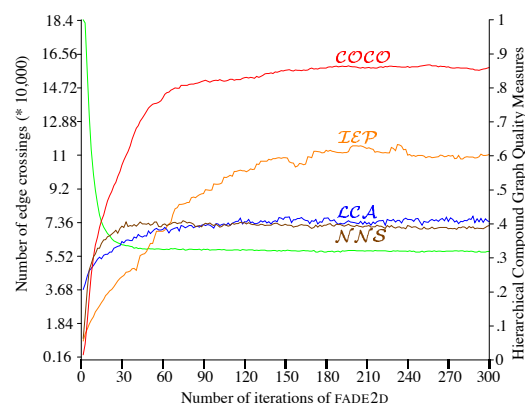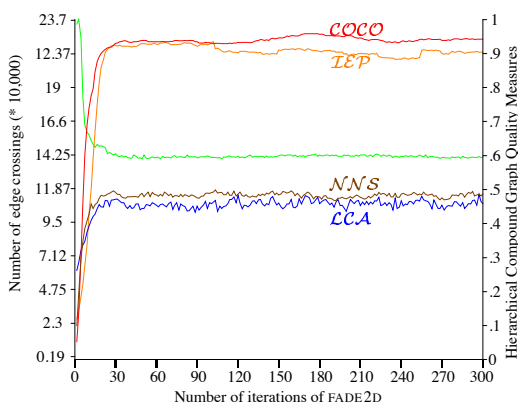

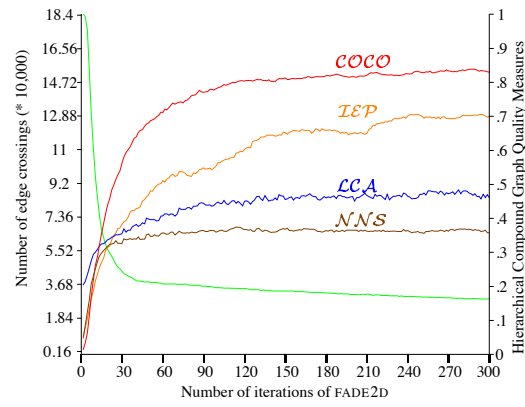**Figure 5.52:** *HCGQM* versus Crossings for *APV* (Mosaic)

Figure 5.53: *HCGQM* versus Crossings for *APV* (Xaero)

5.8.1 Discussion

The hierarchical clustering quality measures have been applied during the first 300 iterations of FADE2D in the progressive cycle. In this discussion our aim is to evaluate, in general terms, the usefulness of our hierarchical compound graph quality measures for these graphs for reverse engineering. Each chart also shows the number of edge crossings in the layout, which steadily decreases as the progressive cycle iterates. As predicted, the micro structure of the layout rapidly improves as indicated by the increase in each measure over the first 60 iterations.

Given this data comes from a software system, where the aim of a good design is to produce components with strong cohesion and low coupling, our classical geometric Coupling and Cohesion measure ($\mathcal{COCO}$) provides a good indication as to quality of a particular layout. Here the $\mathcal{COCO}$ measure indicates the ratio of the number of edges inside each cluster to those crossing a cluster boundary, on every level of the inclusion tree. This measure steadily increases as the FADE2D algorithm is applied. This indicates that the forces, when applied, cause the layout to form into the underlying clusters in the data. This process exhibits the natural clusters as shown in the picture gallery. The results of the $\mathcal{COCO}$ measure in this case study, are in stark contrast to the results in the matrix market case study. Here the underlying data contains clusters one hopes to visualize where as the matrix market data is often highly uniform, without clusters and as such exhibits low values for the $\mathcal{COCO}$ measure.

The $\mathcal{IEP}$ measure gives a general indication as to the overall quality of the horizon level précis in the hierarchical compound graph. A value of 1, indicates that every implied edge represents a true path between every pair of nodes in each cluster. However, due to regular nature of the space decomposition used, some clusters contain unconnected nodes. Here this measure is computed on a single space decomposition, ignoring the fact that the graphs contain many disconnected subgraphs. As a result, for graphs with a large number of small disconnected subgraphs the value for the $\mathcal{IEP}$ is low, whereas for connected graphs the measure is often high. The higher the value for $\mathcal{IEP}$, the greater confidence in a visual précis, since it is a more accurate representation.

The $\mathcal{LCA}$ measure is difficult to interpret. It should provide an indication as to the clustering depth of each edge in the hierarchical compound graph, that is, on average how far up the tree do edges cause implied edges. These results indicate that the $\mathcal{LCA}$ does improve as the layout improves but also that the measure itself may be badly normalised.

The $\mathcal{NNS}$ provides a measure for how strongly interconnected nodes are within a cluster. Given this is real world data of various structural views of a software system one does not expect every entity in every cluster to access one another. Here the value of $\mathcal{NNS}$ increases quickly to a small value which indicates that on average nodes in a layout of each cluster are related to a few other nodes but not all the nodes in the cluster.

Overall, the measures steadily increase to a plateau which supports the hypothesis that as the quality of the drawing improves (measured here by edge crossings), the quality of the clustering exhibited by the layout also improves.

5.9   Remarks

This Chapter has presented a software visualization case study using the FADE paradigm for large graph drawing. These results shown that the FADE paradigm proceeds by making small iterative changes to the layout of the graph which improves the values of the hierarchical compound graph quality measures. As the layout improves groups of related nodes are drawn together, whereas unrelated nodes are typically drawn far apart. Drawings of such layouts tend to exhibit the natural clusterings of the graph.

Overall, we have demonstrated how the FADE paradigm provides a fast layout method coupled with abstract representation and measurement for a reverse engineer analyzing a software system.

Case Study II: Matrix Market Visualizations

"Intuition becomes increasingly valuable in the new information society precisely because there is so much data." - John Naisbitt

Numerical matrix data is used in comparative studies of algorithms for numerical linear algebra applications in a range of domains. Examples of such domains include, Chemical Engineering, Computer System Simulation, Finite Element Analysis, and Power System Networks. The Matrix Market is a repository of such test matrix data [29]. This repository consists of 482 sparse matrices, from different application domains.

This Chapter is arranged as follows. Section 6.1 provides the context for this case study and outlines issues addressed using this matrix data. A brief synopsis of the background details of each matrix class is given in Section 6.2. Section 6.3 contrasts the drawings produced with the FADE paradigm and existing structure and cityplot visualizations along with a comparative review of alternate layout methods in Section 6.3.1. The results of this case study are presented in Sections 6.5 to 6.9. A picture gallery, comparing and contrasting existing structure and cityplot visualizations, with drawings produced by FADE2D is given in Section 6.5. Section 6.5.1 presents a discussion of the picture gallery results in structural terms rather than according to any application domain specific analyses. Section 6.6 provides the graph drawing aesthetic measures and a discussion of these results.

In Section 6.7 the results of applying the graph drawing aesthetic measures to the horizon level précis of these graphs are presented along with a discussion of these results in Section 6.7.1. Two class of performance versus error charts are given in Section 6.8 along with a discussion of them.

The results of the hierarchical compound graph quality measures, applied to these graphs, are given in Section 6.9 along with a discussion of these results in Section 6.9.1.

6.1 Context

For the purposes of this case study, we restrict our interest to forming visual representations of this data rather than application domain specific analysis of the results produced. Specifically, our goal is to evaluate the FADE paradigm in rapidly producing graph drawings and visual précis of the structured, semi-structured and high clustered data contained in this repository. The graph drawing and visual précis can be used by data analysts in the task of studying the data to discover patterns or inconsistencies. In this section we present the context of our evaluation.

6.1.1 The Matrix Market

In this Chapter we investigate the drawing, abstract representation and measurement of a range of medium to large graphs from the Matrix Market. The Matrix Market is a component of the NIST (National Institute of Standards and Technology, USA) project on Tools for Evaluation of Mathematical and Statistical Software which has focus areas in linear algebra, special functions and statistics. Within the Matrix Market these matrices are visualized as both two dimensional "structure plots" and three dimensional "cityplots". Here we aim to use our FADE drawing and visual abstraction paradigm to represent these data sets as large graph drawing coupled with an analysis according to our hierarchical compound graph quality measures.

For the purposes of this thesis, we focus on five issues in this case study:

- The visualization and abstract representation of these matrix data sets as graphs with nodes and edge strengths.

- Discussion of the drawings and their abstract representations, in terms of their ability to visually show the natural clusters or patterns in the data.
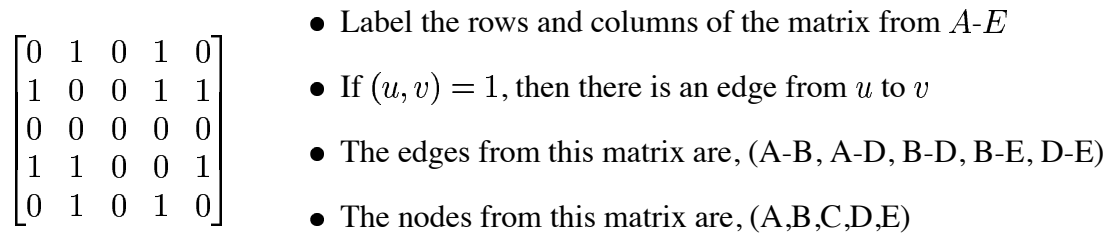
$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

- Label the rows and columns of the matrix from A-E

- If $(u, v) = 1$, then there is an edge from u to v

- The edges from this matrix are, (A-B, A-D, B-D, B-E, D-E)

- The nodes from this matrix are, (A,B,C,D,E)

Figure 6.1: A matrix represented as an undirected graph

- The measurement of various aesthetics of the final drawing and horizon drawings, introduced in Chapter 2.

- The performance measurement of the primary layout algorithm (FADE2D) from our FADE paradigm, introduced in Chapter 4.

- The use of our hierarchical compound graph quality measures introduced in Chapter 3, to determine if there is progressive cycle of layout and clustering improvement in this class of structured, semi-structured and clustered data.

6.1.2 Matrix Data

The data in this case study comes in the form of a matrix description. A matrix is a square or rectangular arrangement of symbols or numbers, in rows or columns, used to summarize the relationships between different entities. Matrices are used for various mathematical operations, such as representing the coefficients of simultaneous linear equations. A square matrix $(n * n)$ represents an adjacency matrix of a graph with n nodes. The (u, v) entry of the matrix is the strength or number of edges from node u to node v. A symmetric adjacency matrix represents an undirected graph, whereas an non-symmetric adjacency matrix represents a directed graph. An example of representing a symmetric matrix as a graph of nodes and edges is shown in Figure 6.1.

For the purposes of this thesis we are interested in drawing, abstractly representing and measuring large simple undirected graphs. The matrix market data sets do not contain multiple edges but rather edge strengths. The matrices do contain self-loops and in a few cases the graphs are directed. In this case study we treat all matrices as simple undirected graphs with edge strengths, that is, as attributed graphs.

Many of these data sets have already been visualized, by other researchers, with simple three dimensional "cityscape plots" or two dimensional "surface plots". The FADE paradigm introduced in Chapter 4 introduces several new ways to visualize and abstractly represent this data, which we will compare and contrast with the original visualizations. Most of the graphs in this collection are non-planar and as such, might be amenable to three dimensional layout and visualization.

6.2 Description of Graphs (matrix data) in this case study

We now present an overview of background details, and an explanation of each matrix class. These matrices come from a wide variety of areas, from Chemical Engineering to Computer Systems models. It is important to note that some of the graphs represent geometric elements and their interrelationships; however the data available contains none of this geometry, only the elements and the strength of the relations (if present). Other graphs in this case study are of purely abstract elements and relations.

Power System Distribution Networks

A *power grid* or *power system distribution network* consists of power lines that distribute electricity to consumers from a few high-voltage sources, such are coal fired power plants, nuclear reactors and hydro-electric stations. These networks typically have a small number of high-voltage electricity lines, which are connected to the sources of power. These lines supply electricity to regional networks that eventually deliver power to customers. To ensure redundancy in supply, the regional networks draw power from several high-voltage lines, along with being connected to other regional networks. These networks represent relational information, where nodes are power stations or sub-stations and the edges are the electricity transmission lines. These graphs are very sparse but also quite irregular. While the average degree is typically small, it is common to have a few nodes of very high degree. These properties make it difficult to develop parallel algorithms for power system simulations and applications in general. Several medium to large power system distribution networks have been codified into sparse matrices in the *Boeing Harwell set*,

which is part of the larger matrix market series. These matrices are often used to benchmark parallel sparse linear algorithms or in incomplete factorization problems [19]. Examples of graphs drawn from this set include `1138bus` (Figure 6.25), `bcspwr07` (Figure 6.21) `bcspwr09` (Figure 6.25) and `bcspwr10` (Figure 6.27).

Bounded Finline Dielectric Waveguide

A *waveguide* is a device that acts as a conduit (channel) to guide the propagation of electromagnetic waves along a path defined by the physical structure of the guide. Physically, a waveguide is a rectangular or circular pipe used to guide electromagnetic waves at microfrequencies. The electromagnetic field propagates lengthwise and in theory each wave guide consists of perfect electric conductor (PEC) and dielectric (non conducting) structures. A *finline waveguide* is a bounded waveguide which operates extremely well in the millimeter wave spectrum, that is, at microfrequencies. Waveguides are most often used to connect the output of a radar amplifier to a horn or dish antenna.

The graphs in this set arise from the study of the propagating modes and magnetic field profiles of a rectangular waveguide filled with dielectric and PEC structures [109, 258]. The electric and magnetic fields of an electromagnetic wave have a number of possible arrangements when the wave is traveling through a waveguide. Each of these arrangements is known as a mode of propagation. This graph set is called BFWAVE, and comes from the Department of Electrical Engineering at the University of Kentucky [258]. The nodes represent sample points and the edges represent the change in the strength of the electromagnetic waves between those points.

Examples of graphs drawn from this set include `bfw398a` (Figure 6.29), `bfw398b` (Figure 6.31), `bfw782a` (Figure 6.33), `bfw62a` (Appendix A), `bfw782b` (Appendix A).

Crystal Growth Simulation

Crystal growth is the study of the conditions for controlling the change of state of certain chemical or physical properties, to grow crystal experimentally. These matrices are used to determine the stability of the interfacial crystallization of a piece of solid crystal solidifying
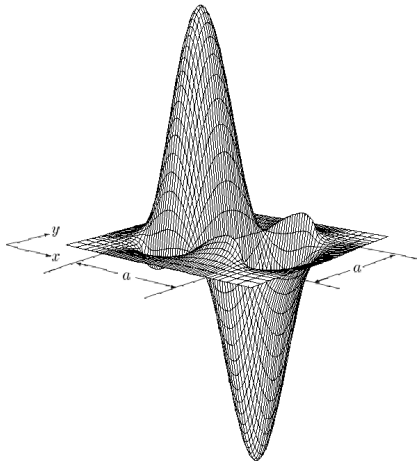
Figure 6.2: Square waveguide visualized as a normalised frequency, courtesy of Albert T. Galick [100]

from some undercooled melt [300]. The nodes represent sample points and the edges represent the change in the stability of the substance between those points.

An example graph (`cry10000`) from this set is drawn in Figure 6.63. Here the change in stability can readily be seen in the colour change across the edges in the top right part of the graph drawing.

Square Dielectric Waveguide

The matrices in this set relate to the same general class of waveguide structures as described in Section 6.2. However, unlike the finline dielectric waveguides, here the waveguide consists of a dielectric material surrounded by another dielectric material with a lower refractive index, such as an optical fiber surrounded by air. These waveguide problems arise in many integrated circuit applications and these graphs represent the finite difference discretisation of the magnetic field profiles. An alternate visualization of such a matrix is shown in Figure 6.2.

Examples of graphs drawn from this set include `dw2048` (Figure 6.35), `dw8192` (Figure 6.39), `dwa512` (Figure 6.47), and `dwb512` (Figure 6.6).

Dispersive Waveguide Structures

Unlike the other waveguide structures, these waveguides consist of conductors with finite conductivity and cross section in a lossy dielectric medium. The "strength" of an edge represents a conductivity measure. An example graph `dwg961a` from this set is drawn in Figure 6.65.

Finite Element Approximation

An *element* is a basic building block of finite element analysis. Models are put together from an assembly of elements. These matrices represent finite element approximations of both physical and theoretical structures. The underlying geometry is absent in these matrices and the edges represent varying diffusivity between elements. These matrices are primarily used in finite element analysis, which is a modeling technique to discretize a continuous geometric data set for analysis. In particular, the method is used to represent the behavior of a structure under an external loading.

Examples of graphs drawn from this set include `nos4` (Figure 6.49) and `nos7` (Figure 6.53).

The Olmstead Model

The Olmstead model represents the flow of a layer of "viscoelastic fluid" heated from below. A *viscoelastic fluid* is simply a liquid that has relatively high resistance to flow but can easily resume its original shape after being stretched or expanded.

An example graph (`olm1000`) from this set is drawn in Figure 3.21.

Oceanic Modeling

These matrices are Platzman models which describe tidal motions in bays, enclosed basins and in this case oceans. Here we have finite-difference models for the shallow wave equations of the Atlantic and Indian Oceans. The smaller matrix corresponds to the North Atlantic Ocean. Here, as in Section 6.2, the geometry underpinning this data set has been

stripped away. So the graph is simply a mathematical model describing the tidal motion of water in various "channels", between recording locations.

Examples of graphs drawn from this set include `plat362` (Figure 6.43), `plsk1919` (Figure 6.51), `plskz362` (Figure 6.55), and `plat1919` (Appendix A).

Small Signal Model

This collection comes from an application of the Hydro-Quebec power systems' small-signal model. These models are used in power system simulations and a variety of methods are used to analyze the operation of such nonlinear devices. Typically the signal represents small variations of current or voltage about motionless points. These matrices are highly unbalanced, which presents significant problems for the parallelization of solvers to numerical linear algebra problems.

Examples of graphs drawn from this set include `qh1484` (Figure 6.57), `qh768` (Figure 6.37), and `qh882` (Figure 6.45).

Reaction-diffusion Brusselator Model

The *Brusselator model*, proposed by I. Pregogine *et al.*, is often used to describe oscillating chemical reactions [291]. The models in this collection are of 2D reaction-diffusion models representing the concentrations of two reactions.

Examples of graphs drawn from this set include `rdb3200l` (Figure 6.59), `rdb200` (Figure 4.24), `rdb450` (Figure 3.31), `rdb200l`, `rdb2048`,`rdb1250l`, `rdb2048l`, `rdb800l`, `rdb968`, `rdb1250` (Appendix A).

Oil Reservoir Simulation

These matrices represent a system of linear equations extracted from oil reservoir modeling programs. The models come from finite-difference approximation of a simulation model, where the underlying geometry has been removed. These matrices were issued as a challenge to the petroleum industry and the numerical analysis community to find the fastest solution to these sets of linear equations. The sets in this collection represent, Black oil
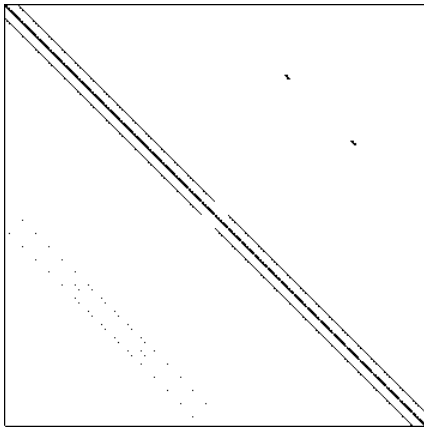
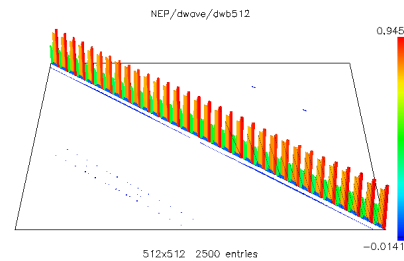**Figure 6.3:** A *structure plot* of `dwb512`

Figure 6.4: A *cityplot* of `dwb512`

simulation with shale barriers, a thermal simulation with steam injection, an IMPES simulation of a black oil model, an IMPES simulation with flow barriers, and a fully implicit black oil model.

An example graph (`sherman4`) from this set is drawn in Figure 6.61.

6.3 Matrix Visualization and Graph Drawing

Visualization is clearly a useful tool for analyzing sparse matrix structures. Given the format of the data, a natural style of visualization is to represent the matrix in two or three dimensions, as shown in Figures 6.3 and 6.4 respectively. In two dimensions one possible visualization is a *structure plot*, which consists of the nodes ordered along the x- and y-axes with a (u, v) grid point representing an edge. A structure plot is used to provide a quick visual check on the sparsity pattern of a particular matrix. In three dimensions a visual representation called a *cityplot view* can be used to visualize a matrix. Intuitively a matrix cityplot view looks similar to large buildings in a city with a grid layout. Along with the benefits of a structure plot, cityplots allow for a quick check of the relative magnitude of matrix entries. In three dimensions an edge is represented as a vertical block. The height and colour of each block represents the relative strength of the edge. The edges with the highest relative strengths are drawn as the tallest positive blocks at the top of the colour scale, whereas the edges with the lowest relative strengths are drawn as the tallest negative

Domain	Graph Name	$|\mathcal{N}|$	$|\mathcal{E}|$
Power systems network	`1138bus`	1138	1458
Western US power network	`bcspwr07`	1612	2106
	`bcspwr09`	1723	2394
	`bcspwr10`	5300	8271
Bounded Finline Dielectric Waveguide	`bfw62a`	62	200
	`bfw398a`	398	1658
	`bfw398b`	398	1256
	`bfw782a`	782	3394
	`bfw782b`	782	2600
Crystal Growth Simulation	`cry10000`	3699	7164
Square Dielectric Waveguide	`dw2048`	2048	4094
	`dw8192`	8192	17404
	`dwa512`	512	1004
	`dwb512`	512	1024
Dispersive Waveguide Structures	`dwg961a`	706	1350
FIDAP package graph	`fidap005`	27	126
	`fidapm02`	200	2805
Finite Element Approximation	`nos4`	100	247
	`nos7`	729	1944
The Olmstead Model	`olm1000`	1000	1997
Oceanic modeling	`plat1919`	1919	15240
	`plat362`	362	2712
	`plsk1919`	1919	4831
	`plskz362`	362	880
Small Signal Model	`qh1484`	1484	2492
	`qh768`	768	1322
	`qh882`	882	1533
Reaction-diffusion Brusselator Model	`rdb1250`	1250	3025
	`rdb1250l`	1250	3025
	`rdb200`	200	460
	`rdb200l`	200	460
	`rdb2048`	2048	4992
	`rdb2048l`	2048	4992
	`rdb3200l`	3200	7840
	`rdb450`	450	1065
	`rdb800l`	800	1920
	`rdb968`	968	2332
Oil reservoir simulation	`sherman4`	1104	1341

Table 6.1: Combinatorial properties of the matrices used in this case study

$$
\begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1
\end{bmatrix}
\quad
\begin{bmatrix}
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1
\end{bmatrix}
$$

Figure 6.5: A matrix which the row and column orders are changed from a,b,c,d,e,f to a,d,b,e,c,f

blocks at the bottom of the colour scale. Structure and cityplot visualizations are primarily used to access the quantity of edges in a particular matrix. This information is useful to quickly determine the sparseness of a particular matrix or to visually compare the relative sparsity of two different matrices.

Other analyses of such matrices are highly dependent on the relative ordering of the rows and columns, as shown in Figure 6.5. The two matrices shown in Figure 6.5 represent the same elements and interrelations. A structure plot or cityplot visualization of the first matrix would easily show the two disjoint clusters of elements with the strong interrelationships, whereas with a visualization of the second this fact would be different to discern visually. The matrices in this data set have been pre-ordered to best represent the structure of the set when displayed in two or three dimensions using a structure or cityplot visualization. This pre-ordering, which is not used within our drawing paradigm, is in effect an analysis step which occurs before the visual analysis commences. Once the matrices have been ordered then the use of either the structure of cityplot view is appropriate.

Clearly there are numerous ways to represent the relative strengths of the relations (edges) between two elements in the data set. In the structure plot view, the strength of a relationship is hidden and only the fact that two nodes are connected is shown, regardless of the strength or weakness of the relationship. In the cityplot view the relative strength or weakness of a relationship is indicated by the height of the block above or below the x,y plane. These blocks are also colour coded to further aid in the identification of groups, patterns or repeating patterns. This approach is clearly advantageous if the exploratory data analysis is concerned with issues such as; locating the strongest/weakest elements, determining an overall view of relative relationship strength, and determining the local or

global uniformity in relationship strengths.

Unlike these visualization approaches we take a "proximity" and "structure" view of the data set. Instead of representing an element u as row-column, we treat it as a node. The relationship this element has with other nodes, is represented as an edge. Our visualization paradigm aims to draw adjacent nodes close to each other along with non-adjacent nodes being drawn far apart. The *proximity* of nodes in the drawing allows the viewer not only to see the relationships but also how these relationships relate to other geometrically close nodes and relationships. In our two dimensional drawings, the strength of a particular relationship is coded according to a similar colour scale as that used in the cityplot visualizations. This approach makes it easier to visually compare and contrast the cityplot and graph drawing visualizations. The *structure* and *colour coding* used in the graph drawing approach allows large patterns, structures, groups, sub-graphs, and clusters to be readily identified. Overall, our approach is useful for showing, on various levels of abstraction, the major structures and patterns exhibited in these sparse matrices, rather than the classical quantitative visual analysis which structure and cityplot visualizations are useful for.

The structure and cityplot visualizations are reproduced with the kind permission of Dr. Ronald F. Boisvert of the Mathematical and Computational Sciences Division of the Information Technology Laboratory at the National Institute of Standards and Technology, who maintain the Matrix Market [29].

We now briefly introduce other layout methods that can be employed in the drawing of the graphs from this data set.

6.3.1   Alternate Layouts

Other layout algorithms, not just those from FADE, can be employed to visualize these graphs. The aim of this section is to compare and contrast the final drawings produced by the FADE2D and FADE3D algorithms with the drawings produced by other two and three dimensional layout methods. The comparisons made here are in terms of the quality of the drawing (measured in terms of graph drawing aesthetics) and performance (measured in time). The layout methods used in this comparison include a standard force directed
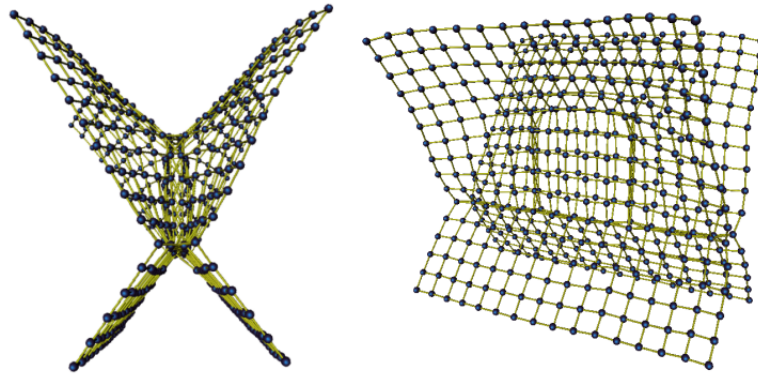
Figure 6.6: Two views of `dwb512b`, drawn in three dimensions without edge strengths.
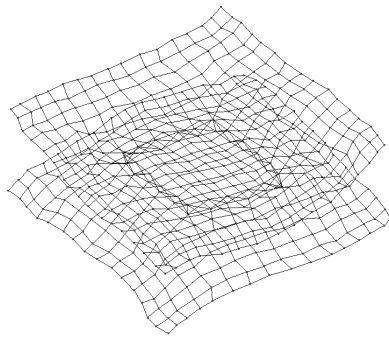

Figure 6.7: Two dimensional force directed layout of `dwa512`, from AGD
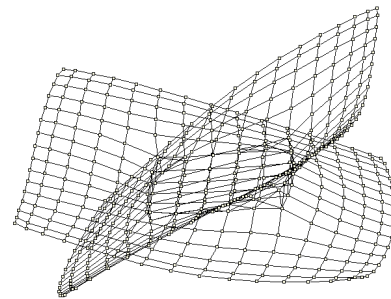
Figure 6.8: Three dimensional force directed layout (projected to 2D) of `dwa512`, from AGD

drawing method, a hierarchical drawing method, a planarized drawing method, a circular drawing method, and a Tutte style drawing method.

The drawings produced by FADE, for example Figures 6.6 and 6.18, clearly show the overall structure of the graph and the fact that it is composed of two related and connected sub-graphs. The alternate drawings come from a graph drawing tool called AGD which includes 18 possible layout algorithms that can be applied to graphs [2]. Unfortunately, due to the nature of the graphs in this case study, that is, non-planar with irregular degree, only 5 layout algorithms can actually be applied. The algorithms include; spring embedder (force-directed), planarization, circular, Sugiyama, and Tutte. We use `dwa512` as an example in this comparative review.

Graph	Nodes	Edges	FADE2D	θ	AGD
dwa512	512	1004	0.041	1.0	0.0956634
rdb1250	1250	3025	0.113	1.0	0.3507658
rdb2048	2048	4992	0.1892	1.0	0.5739804
cry10000	3699	7164	0.459	1.0	2.1683704
bcspwr10	5300	8271	0.573	1.0	2.8061264
dw8192	8192	17404	1.423	1.0	7.97195

Table 6.2: Time in seconds, to compute one iteration FADE2D with a particular theta, as compared with one iteration of the spring layout in AGD. Here the time per run is averaged across the first 400 runs of each algorithm.

Fruchterman Reingold Force Directed Layout

The spring embedder layout provided in AGD is the classical Fruchterman Reingold grid based refinement [97] to the original force directed layout algorithm of Eades [69]. As AGD is a sophisticated and highly customizable graph drawing tool it would be unfair to begin a comparison without first showing a comparable two and three dimensional force directed layout produced by this tool. The results of applying this layout, in terms of graph drawing aesthetics, are similar to the results obtained with FADE2D and 3DFADE, as one would expect. However, our drawing paradigm also directly supports the generation of multi-level views along with a greater computational efficiency. Our layout algorithm performs approximately twice as fast even on the relatively small graph shown in line 1 of table 6.2. The performance is 0.041 seconds per iteration (at a 0.6% error) compared with 0.0956 seconds per iteration for AGD. It should be noted that observational evidence suggests the spring layout method in AGD contains further ad-hoc algorithmic refinements to improve the performance of the basic Fruchterman Reingold force directed layout algorithm. If the algorithm has been further refined then any comparisons using the data shown in table 6.2 are invalid. Any comparison is attempting to compare two essentially different algorithms, rather than two force approximation methods. For example, one hypothesis is that in the early iterations in AGD it appears that only edge attractions are computed or that very large grids are used. However without access to the implementation of this system, it is impossible to verify this hypothesis.
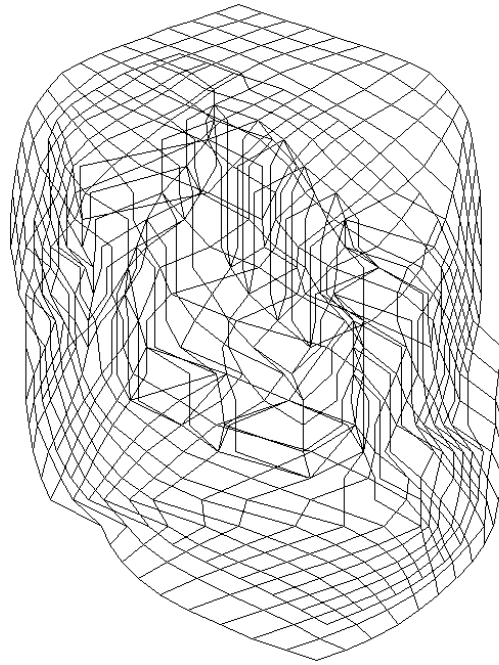
Figure 6.9: Hierarchical layout of dwa512 drawn with Sugiyama's algorithm

Hierarchical Graph Drawing

The drawing shown in Figure 6.9 is a hierarchical graph drawing of `dwa512`, based on Sugiyama's algorithm as described in Section 2.2.3. This hierarchical drawing presents a fair rendering of the graph, although it is difficult to tell that the graph consists of two "sections". Different crossing minimization steps such as median, split and sifting make the two sections of the graph more apparent but over reduce the visual balance. Regardless of ranking assignment or the crossing minimization method used, it is impossible to recognize the fact that the two sections are connected in a regular manner. In graph aesthetic terms this drawing has good aspect ratio, good angular resolution, good edge length uniformity, poor display of symmetry and has fewer edge crossings (678) than the FADE2D drawing in Figure 6.18.

Circular Layout

The drawing shown in Figure 6.10 is a circular layout graph drawing. In graph aesthetic terms this drawing has good aspect ratio, poor angular resolution, poor edge length uniformity, poor display of symmetry and has many more edge crossings (4828) than the drawing
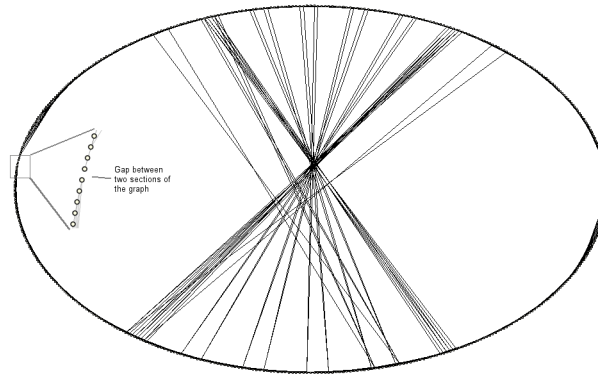
Figure 6.10: A circular layout of `dwa512`, with the gap between the two sections illustrated

in Figure 6.18. However, unlike the hierarchical layout it is possible to see that there are large sets of nodes which are only locally related to each other. And in examining the graph very closely, it is possible to see that there are two "sections" separated by a gap. Finally, as with the hierarchical layout it is difficult to tell that the two sections are connected in a regular manner.

Tutte Force Directed Layout

The drawing shown in Figure 6.11 is circular force directed layout called a Tutte graph drawing. In graph aesthetic terms this drawing has good aspect ratio, very poor angular resolution, very poor edge length uniformity, very poor display of symmetry and has many more edge crossings than the drawing in Figure 6.18. This is a very bad graph drawing which definitely qualifies for Tuftes axiom that "...if a picture isn't worth a thousand words, the hell with it".

Planarization Graph Drawing

Finally, the drawing shown in Figure 6.12 is a planarization graph drawing. In graph aesthetic terms this drawing has good aspect ratio, very good angular resolution, poor edge length uniformity, very poor display of symmetry and has few edge crossings (552). According to all the aesthetic measures, this is a "good" drawing of the graph. However, it is clear that by not displaying the symmetries and by not maintaining a uniform edge length this drawing does not show the overall shape of the graph, nor the fact that it consists of two

sections or the regular connection between the sections. In fact it would be a conceptual challenge to attempt a cognitive mapping from the drawing in Figure 6.12 to the drawing in Figure 6.6.
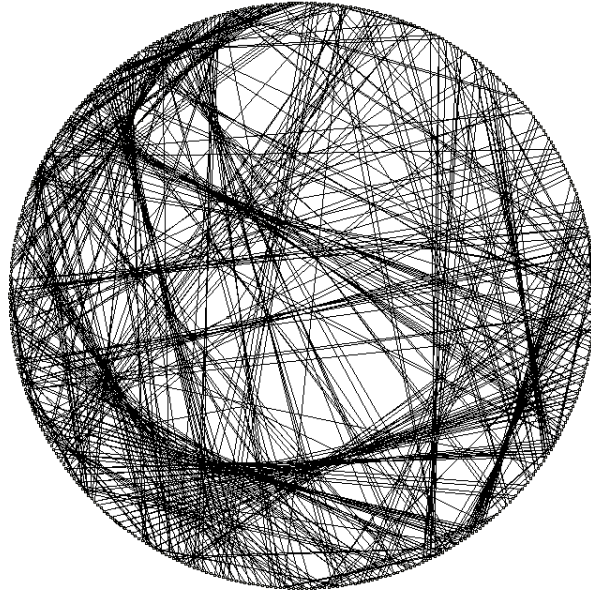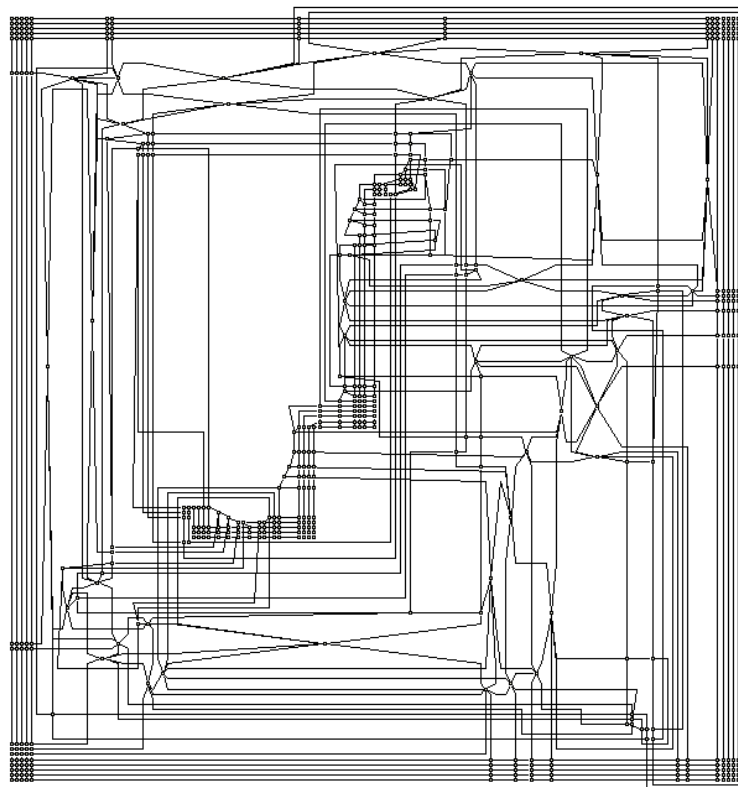
Figure 6.11: A Tutte style layout of `dwa512`

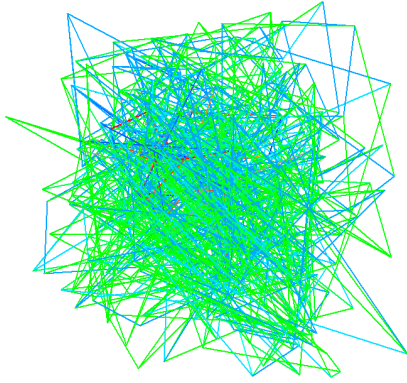**Figure 6.12:** A Planarization Style Layout of `dwa512`

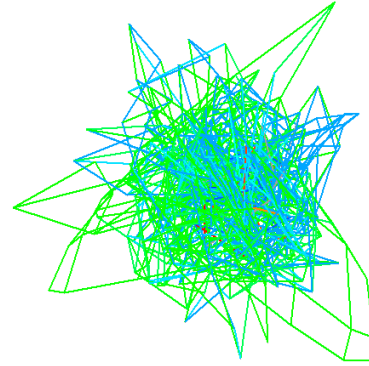**Figure 6.13:** Initial random layout of `dwa512` with 42170 edge crossings

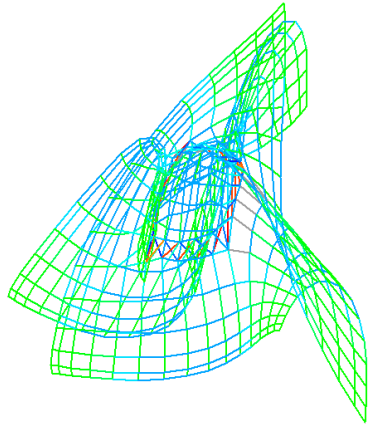**Figure 6.14:** Layout of `dwa512` after one iteration of FADE2D with 33126 edge crossings

Figure 6.15: Layout of `dwa512` after 30 iterations of FADE2D with 1178 edge crossings
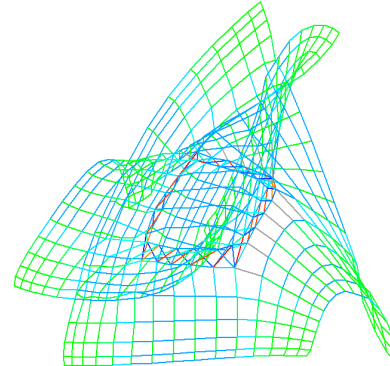
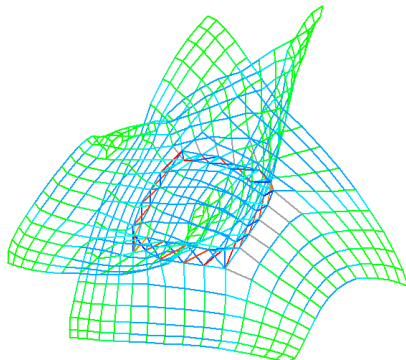**Figure 6.16:** Layout of `dwa512` after 50 iterations of FADE2D with 829 edge crossings

Figure 6.17: Layout of `dwa512` after 80 iterations of FADE2D with 697 edge crossings
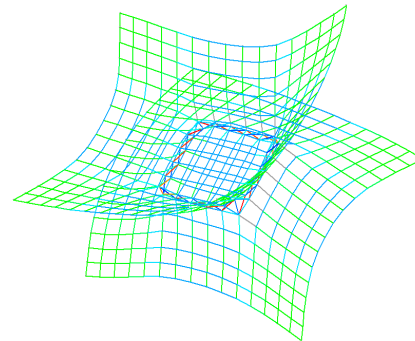

Figure 6.18: Layout of `dwa512` after 140 iterations of FADE2D with 661 edge crossings

6.3.2   Initial Layout

The progressive cycle in the FADE paradigm, described in Chapter 4, requires an initial layout of the graph. The *quality* of this initial layout may be important in determining the quality and time required for the final layout.

Classical force directed layouts typically begin by assigning a random geometry (layout) to the nodes of a graph, as shown in Figure 6.13. By all graph drawing aesthetic measures (except for aspect ratio) this is a poor initial drawing of the graph. By randomly positioning the nodes we introduce many more edge crossings than in the final drawing. With this example we have 63 times as many edge crossings in the initial drawing as the final drawing in Figure 6.18.

As we show in Section 6.9 the first few iterations of FADE2D rapidly decrease the numbers of edge crossings and quickly improve each of the hierarchical compound graph quality measures accordingly. As a starting point for comparing the change in quality, the random layout is a uniformly bad initial layout. This means that we can fairly compare the rate of improvement of quality measures if the initial layout is random.

As part of the discussion of our results, we show how other computationally inexpensive heuristic layout methods based on the FADE paradigm, such as wave-front FADE can be used to give a better initial layout. These layouts are often much closer, in terms of energy, aesthetic measurement and clustering measurements, to the final drawings.

6.4   The Experiments

To investigate the FADE progressive cycle in producing drawings and visual précis we tested the FADE2D, WAVE-FRONT and FADE3D algorithms on the matrices described in Section 6.2.

The results of this case study are presented in following five sections. First, we present a picture gallery of visual comparison between existing structure and cityplot visualizations and the layouts produced by a wavefront layout coupled with FADE2D in Section 6.5. Second in Section 6.6 we present the hard graph drawing aesthetic measures of the FADE

Figure 6.19: Colour range used to codify the weakest to strongest edges in the graph drawings shown in Figures 6.25 to 6.57

layouts shown in the picture gallery. Third, we present a variety of visual précis drawings along with performance, clustering and aesthetic measurements in Section 6.7, which are described in Chapter 4. Fourth, we present the error versus time taken measurements and error versus calculations measurement, for a range of approximations used with FADE2D in Section 6.8, as outlined in Chapter 4. And finally, in Section 6.9 we compare the hierarchical graph clustering measures introduced in Chapter 3.

6.5   Results: Picture Gallery

The graphs in this case study typically consists of a single connected graph, rather than a series of small disconnected subgraphs as in Chapter 5. Here we consider the entire graph with a single space decomposition, which allows us to compare all the measures taken in this case study with the software visualization case study results. Further, this approach produces very accurate high level visual précis.

The final drawings shown here in Figures 6.21 to 6.57 and in Appendix A, are the results of applying a wavefront layout to the graph, followed by a number of iterations of a FADE method to further improve the drawing. Many of the graphs from the Matrix Market contain strengths for each edge, where an edge strength is present this is coded as a colour in the range red to purple, as shown in the colour scale in Figure 6.19. Where no edge strengths are present, the edge is drawn as a black line. As in the previous case study, we are only concerned with simple graphs with no self-edges and no multiple edges. This compromise allows for the use of our FADE paradigm to represent the overall connectivity and edge strengths. As a result, this case study deals with a general class of graphs from a range of domains and compares them according to our aesthetic and clustering measures.

Many of these matrices in this case, when drawn as graph drawings are highly uniform and contain very regular structures. For example, the matrices from the Brusselator
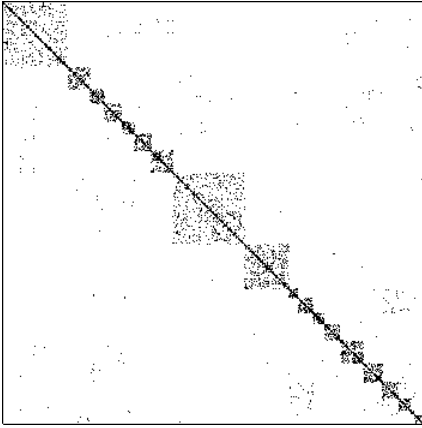
Figure 6.20: `bcspwr07` as a structure plot
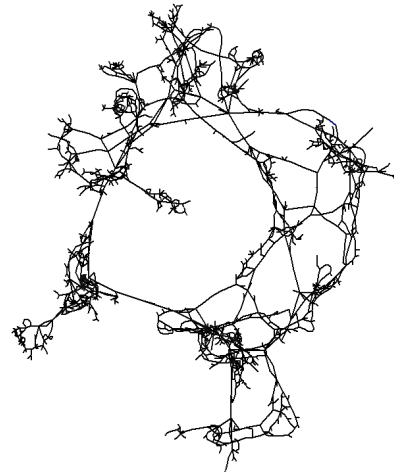
Figure 6.21: `bcspwr07` as a graph drawing
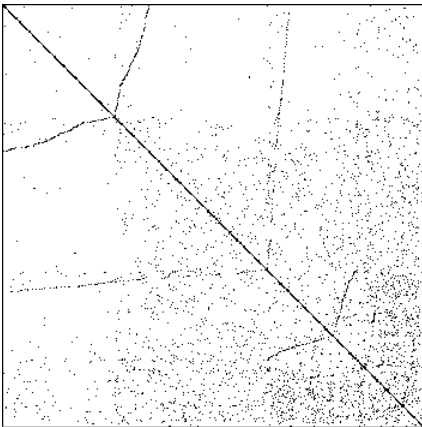
Figure 6.22: `bcspwr09` as a structure plot
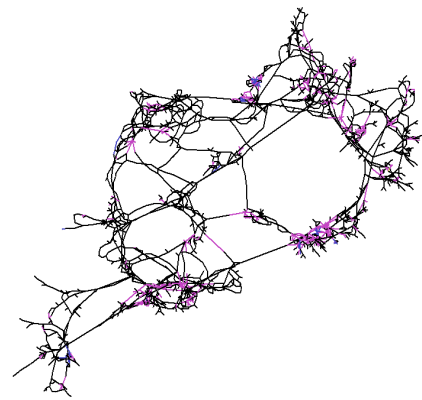
Figure 6.23: `bcspwr09` as a graph drawing

models are drawn as two overlapping grids with FADE2D and as a three dimensional mesh structure with FADE3D. Domain knowledge of such regular combinatorial graph structures should allow an analyst, working with such matrices, to choose a potentially faster layout algorithm than those in the FADE paradigm. Recall that, domain knowledge of the graph structure often suggests a particular layout method or an optimized heuristic to an existing algorithm.
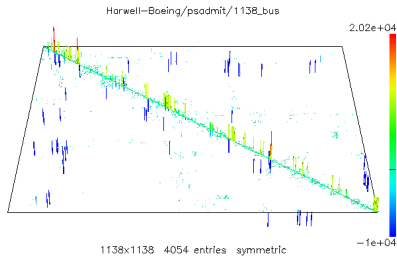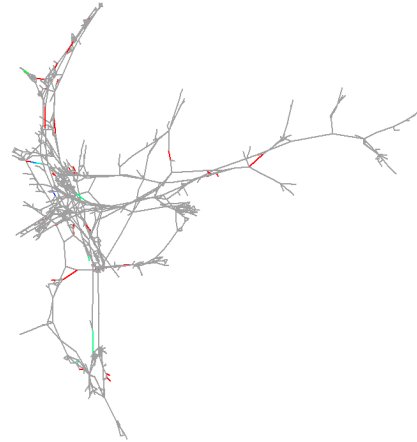
Figure 6.24: `1138bus` as a cityplot
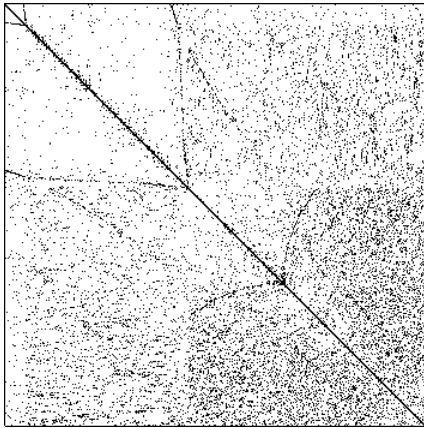

Figure 6.25: `1138bus` as a graph drawing

Figure 6.26: `bcspwr10` as a structure plot
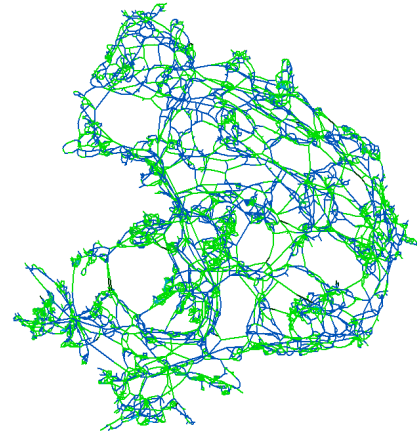

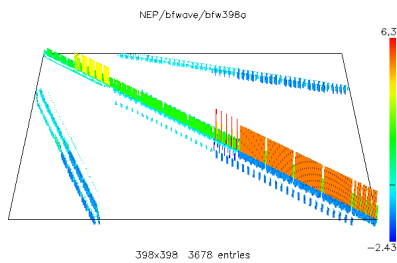**Figure 6.27:** `bcspwr10` as a graph drawing

Figure 6.28: `bfw398a` as a cityplot
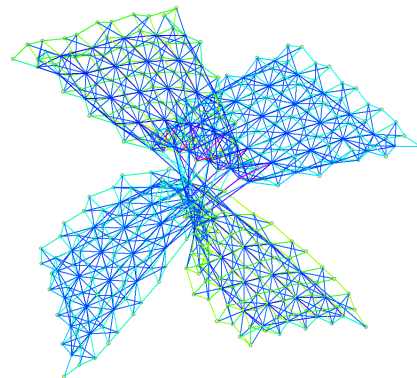

Figure 6.29: `bfw398a` as a graph drawing
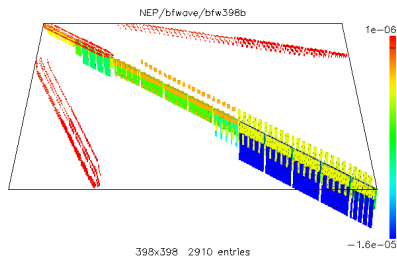
Figure 6.30: `bfw398b` as a cityplot
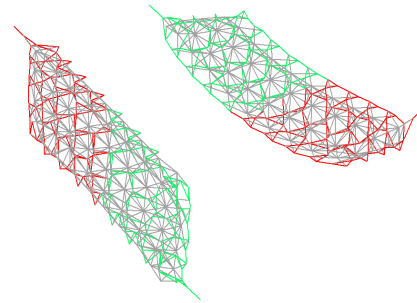
Figure 6.31: `bfw398b` as a graph drawing
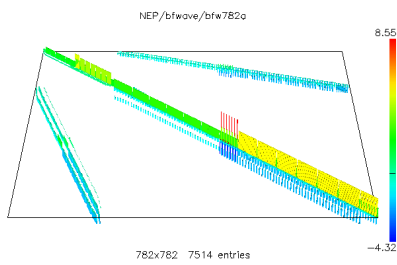
Figure 6.32: `bfw782a` as a cityplot
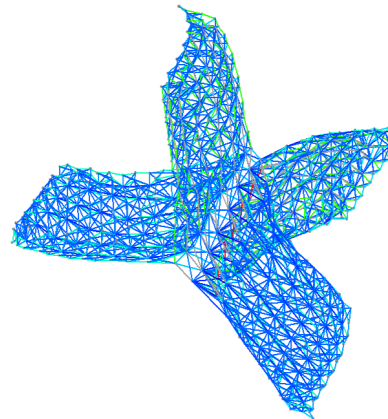
Figure 6.33: `bfw782a` as a graph drawing
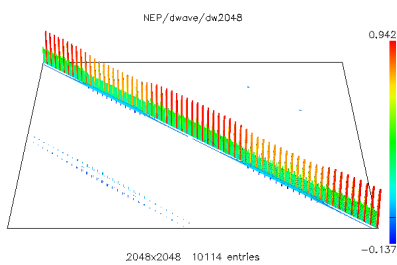
Figure 6.34: `dw2048` as a cityplot
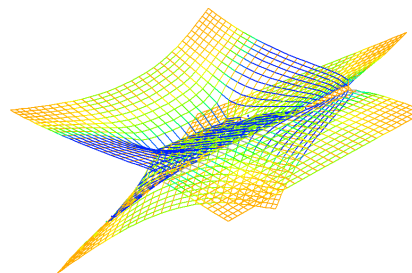
Figure 6.35: `dw2048` as a graph drawing
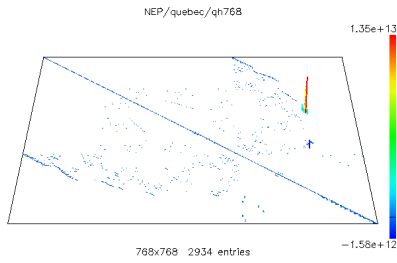
Figure 6.36: qh768 as a cityplot
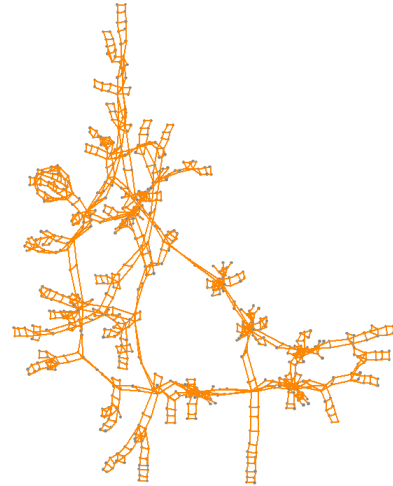
Figure 6.37: qh768 as a graph drawing
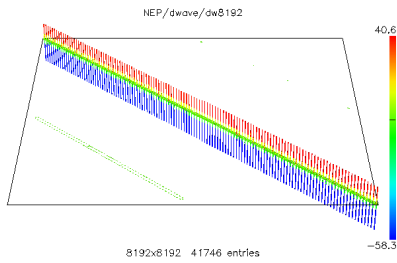
Figure 6.38: dw8192 as a cityplot view
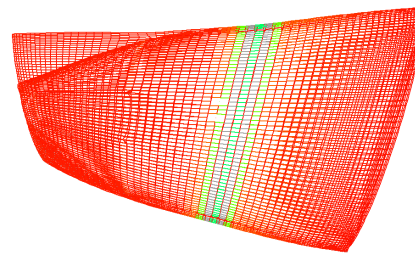
Figure 6.39: dw8192 as a graph drawing
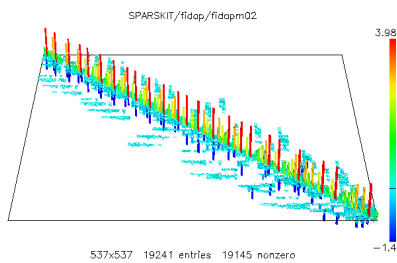
Figure 6.40: fidapm02 as a cityplot
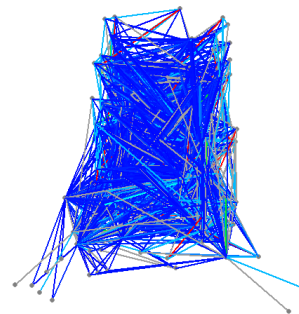
Figure 6.41: fidapm02 as a graph drawing
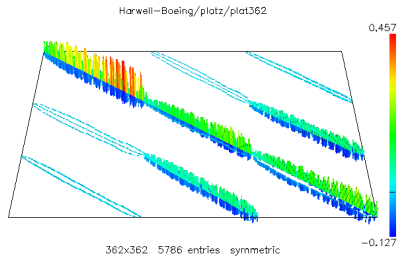
Figure 6.42: `plat362` as a cityplot view
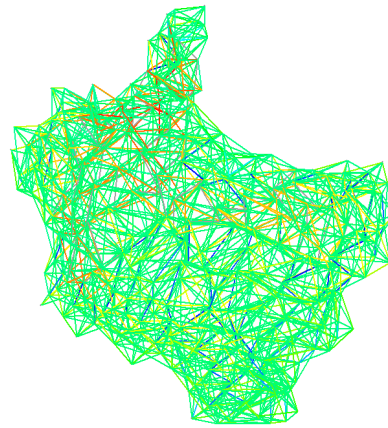
Figure 6.43: `plat362` as a graph drawing
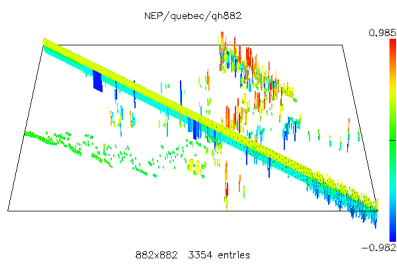
Figure 6.44: `qh882` as a cityplot
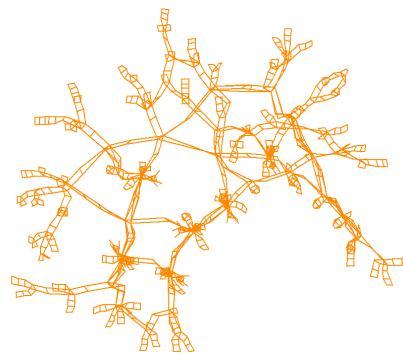
Figure 6.45: `qh882` as a graph drawing
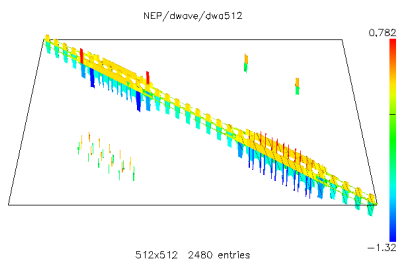
Figure 6.46: `dwa512` as a cityplot
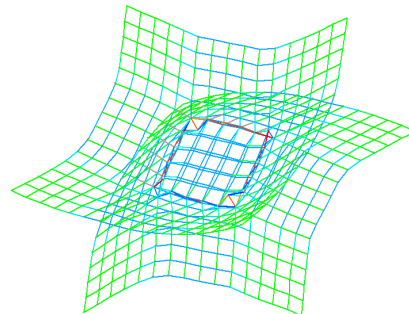
Figure 6.47: `dwa512` as a graph drawing
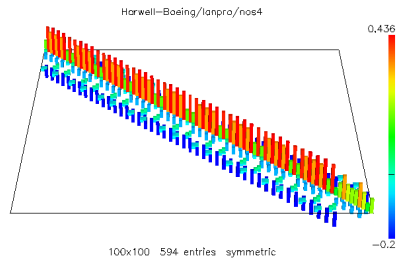
Figure 6.48: `nos4` as a cityplot
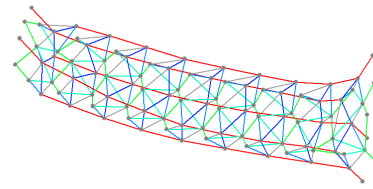
Figure 6.49: `nos4` as a graph drawing
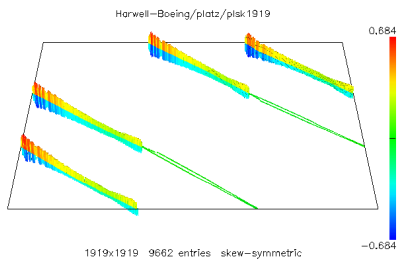
Figure 6.50: `plsk1919` as a cityplot
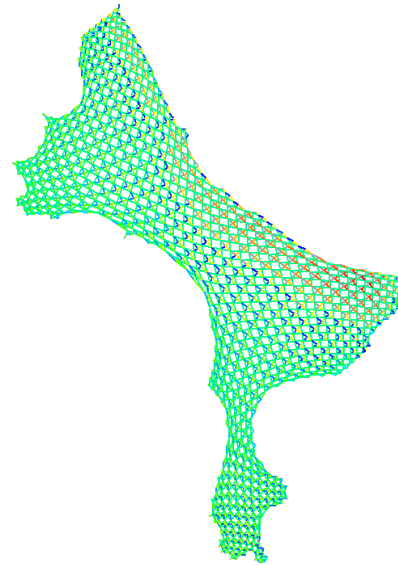
Figure 6.51: `plsk1919` as a graph drawing
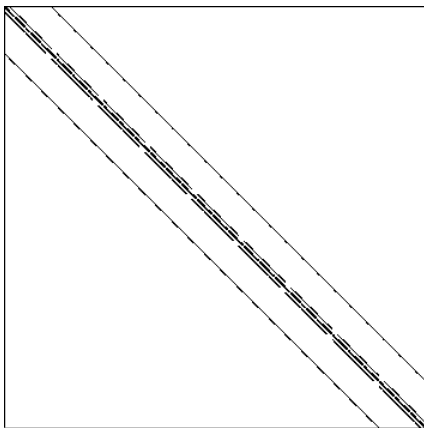
Figure 6.52: `nos7` as a **structure plot**
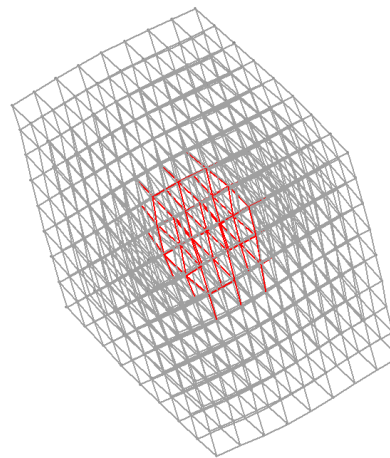
Figure 6.53: `nos7` as a graph drawing
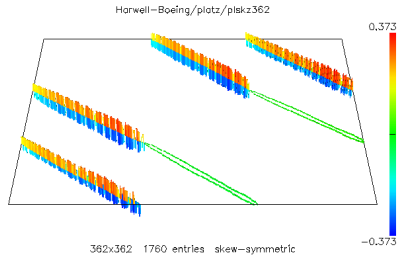
Figure 6.54: `plskz362` as a cityplot
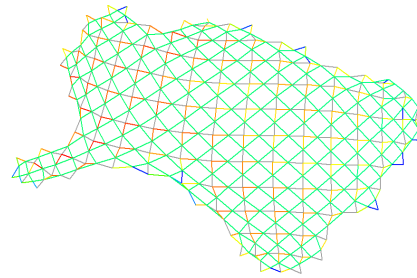
Figure 6.55: `plskz362` as a graph drawing
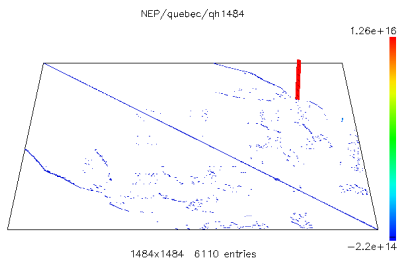
Figure 6.56: `qh1484` as a cityplot
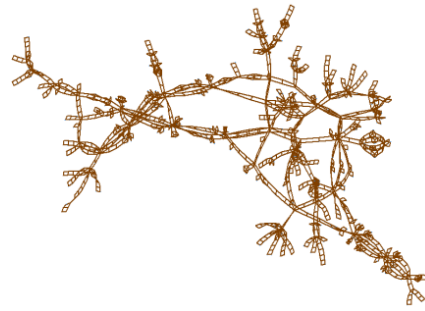
Figure 6.57: `qh1484` as a graph drawing
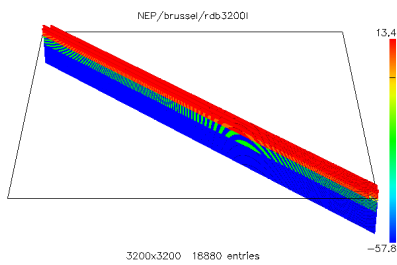
Figure 6.58: `rdb3200l` as a cityplot
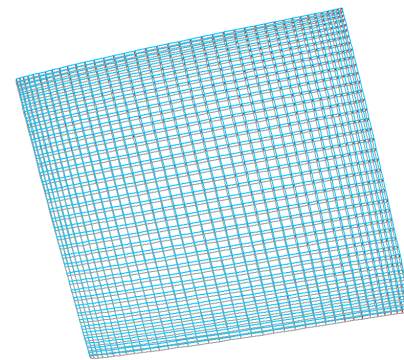
Figure 6.59: `rdb3200l` as a graph drawing
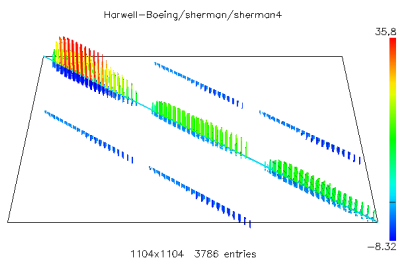
Figure 6.60: `sherman4` as a cityplot
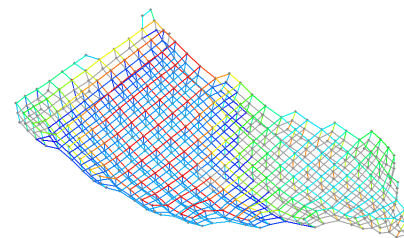
Figure 6.61: `sherman4` as a graph drawing
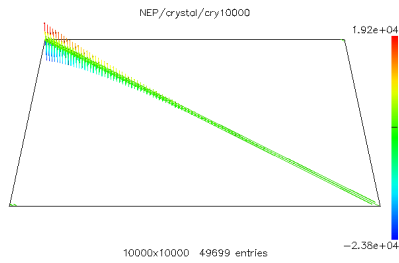
Figure 6.62: `cry10000` as a cityplot
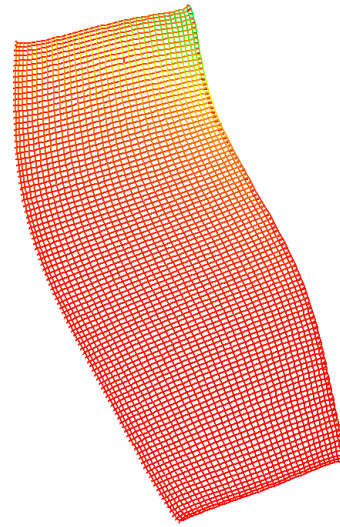
Figure 6.63: `cry10000` as a graph drawing
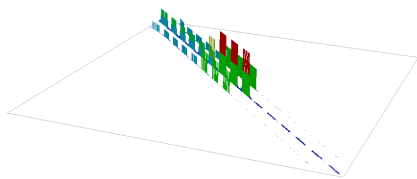
Figure 6.64: `dwg961a` as a cityplot
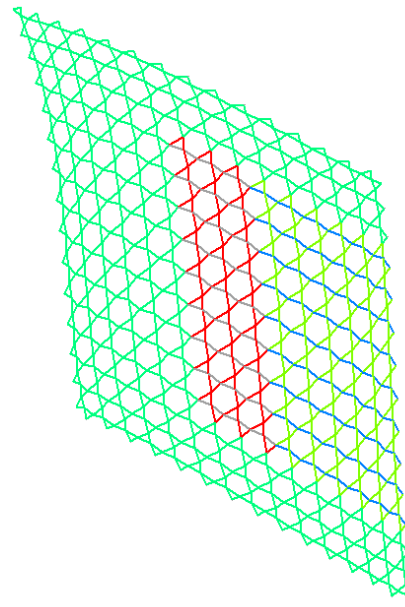
Figure 6.65: `dwg961a` as a graph drawing

6.5.1   Discussion

Given the wide range of application domains these matrices are drawn from, a cross comparison in terms of usability requires domain knowledge, domain experts, and a through interpretation of the visualization of each matrix. In this discussion, the aim is to evaluate, in general terms, the use of the FADE paradigm for the visualization of these matrices rather than any interpretation of the source of the data.

The discussion of the results focuses on five areas:

- The appropriateness of the FADE paradigm for the visualization of these matrices

- Comparison between the graph drawing approach and the approach taken with the structure and cityplot views

- Visually apparent Patterns

- Visually apparent Symmetries

- Visually apparent Clusters.

Most of the graphs, in this case study, are suitable for drawing with the FADE paradigm in two or three dimensions as appropriate. However, graphs with low diameter, such as that shown in Figure 6.41 are not suitable for drawing with the FADE paradigm. The force model underpinning the FADE paradigm is not suitable for *small world* graphs, that is, graphs with a large number of nodes and a small diameter (see [1]) . The layouts produced for such graphs are typically rendered in a small volume or area to satisfy the force model, which often results in many edge crossing and an aesthetically unappealing visualization.

Structure & Cityplot Views

Recall that the structure and cityplot views are often used to provide a quick visual check on the sparsity pattern of a particular matrix and in three dimensions the relative magnitude of matrix entries. The FADE approach also shows such information along with patterns, symmetries and clusters which we collectively refer to as structures in the drawing. However,

since the structure and cityplot views are based on pre-ordered data paths, asymmetries and directedness can often be seen.

The cityplot shown in Figure 6.24 displays little structure but it clearly shows the outlying edges and the edges with the greatest or weakest relative strengths. The graph drawing in Figure 6.25 does show a structure but identifying the strongest or weakest edges and the edges not about the main diagonal is almost impossible.

Although clusters can be visually apparent in the structure plots, as in Figure 6.20 this is dependent on the pre-ordering that has taken place in matrix. Figures 6.22, 6.26 show structure plots with little indication of any clusters, although in the graph drawing several natural clusters appear. This might be due to the ordering in the structure plot to emphasize paths through the data which are shown as lines emanating from the main diagonal.

The cityplot drawing in Figure 6.30 gives little indication that the data consists of two disconnected subgraphs, as shown in Figure 6.31. The cityplot views of the related matrices in Figures 6.28, 6.30 and 6.28 appear similar but the graph drawings reveal aspects of the true structures and how they differ. The connection between the two areas in the graphs, drawn as the central edges in Figures 6.31 and 6.33, are not apparent in either of the cityplot views although the strengths of these edges are.

Although the graph drawings shown in Figures 6.43, 6.51 and 6.61 have many edge crossings, they do clearly show the overall structure or shape of the underlying graph data. The cityplot views of these matrices show little of this structure. Any direct comparison between the graph drawing and the cityplot views is difficult since the cityplot views are devoid of any structures, patterns or symmetries which the graph drawings exhibit.

Clusters

Much of the data in this case study is uniform rather than consisting of clustered nodes. However, although clustering is typically concerned with the grouping of similar nodes and visualization of data with natural clusters aims to show them, here the edges are also attributed and can be considered in the clustering of the data. In this case study often the clustering appears where edges with the same strength are drawn close together without reference to the nodes. We address such visual edge clustering in the next section.

The structure plot, of the ordered data, shown in Figure 6.20 clearly shows clusters of related nodes about the main diagonal of the matrix visualization. These clusters are also visually apparent in the graph drawings. The graph drawings of the power system networks, shown in Figures 6.21, 6.23 and 6.27 consist of edges with categories. Here the natural node clusters, which are sometimes apparent in the structure plots, are quite evident in the graph drawings. Clearly knowledge, such as the location of the nodes would greatly assist a data analyst in using such a visualization for domain specific analyses.

The graph drawings of the power system simulation matrices shown in Figures 6.37, 6.45 and 6.57, are in stark contrast to the cityplot drawings. The graph drawings clearly show the natural clusters and hence the overall structure and shape of this data. The cityplots in Figures 6.36, 6.44 and 6.56 show only the distribution of relative edge strengths within the matrix. This class of data provides a clear example of why domain knowledge about these graphs is essential in any interpretation of the clusters displayed in the FADE drawings.

Although many of the visualizations display two sets of related data, connected in a regular or semi regular manner, we leave the discussion of these to the patterns, and symmetries that the drawings display rather than focusssing on the two sets of data in each drawing.

Patterns

FADE2D treats all of the graphs in this case study as undirected graphs without regard for the combinatorial properties of the graphs itself. The *patterns* discussed in this section pertain to both structural and colour, and often both.

The colour patterns in the Brusselator Model graph drawing, such as that in Figure 6.59, simply reinforce the notion that this data consist of two subgraphs which are related in a regular manner.

Figures 6.29, 6.31 and 6.33 from the bounded finline dielectric waveguide set display several patterns each in terms of their structure and the colours of the edges. Each graph consists of a similar micro structure, where nodes are drawn in a similar pattern in related sets. Figure 6.31 first shows two subgraphs and within each subgraph there are two clear colour patterns evident along with a regular repeating relationship pattern.

Figures 6.35, 6.47, 6.39 and 6.6 from the square dielectric waveguide set each display many patterns. Of note are the colour variation, the cross over pattern and the non-adjacent nodes pattern. In each visualization the colour pattern provides a very regular transition through areas of the graph drawing. In each case the absolute strength of the edges increases closer to the area of cross over. However the increase is not uniform and the pattern of increase itself a pattern. Some of the edges included in the cross over section are the strongest in the entire graph.

For each graph drawing, there are clearly two related areas in the graph which are connected by a relatively few number of edges in a cross over. In Figures 6.35, 6.47 and 6.6 this cross over is draw as a square pattern of nodes and edges forming a three dimensional "pillow" when draw with FADE3D. The sets of nodes not close to the cross over nodes typically form a symmetric layout, which is discussed in the next section.

Figures 6.37, 6.45 and 6.57 display a ladder like pattern within areas of the graph drawing. Figures 6.51 and 6.61 display regular repeating patterns of coloured edges along with a pattern of change in the colour values through the visualization.

Figure 6.49 contains a few categories of edges which form very clear patterns through the graph. Likewise 6.65 has a few categories which form four visual patterns within the graph drawing and the pattern from by the change in colour tone through the edges in Figure 6.63 is quite apparent.

Symmetries

Many of these graphs exhibit highly symmetric layouts when drawn with the FADE paradigm. This is similar to other force direct layout algorithms, which employ a force model. Here the drawing display symmetry not only in the locations of the nodes but also in the strengths of the edges. Difference or asymmetries, where they occur, are noticeable for the analysts using the drawing produced by the FADE paradigm. Clearly, some of these graphs when drawn with FADE3D have many more symmetries than when drawn with FADE2D.

Conclusion

Here we have shown how the FADE2D algorithm produces drawings where the structures such as, natural clusters, patterns, and symmetries in the graphs are apparent. For an analyst, in a particular domain, these visualization provide a unique view of the matrix data in terms of the structure, relationships, and clusters in the data. Of particular note is the ability of the FADE paradigm to display the regular colour patterns and symmetries in the underlying matrices.

6.6 Results: Graph Drawing Aesthetics

Table 6.3 gives the graph drawing aesthetic measures for the underlying graph drawings shown in the picture gallery in Section 6.5.

6.6.1 Discussion

Table 6.3 indicates that many of these final layouts are aesthetically appealing given they typically are drawn with very few edges crossings. As with the graphs in Chapter 5 here the aspect ratio of these drawings is good. Again due to the force model used in the FADE paradigm many of the edges can be elongated which results in the adjacent and non-adjacent aesthetic measures being quite poor. In many of the drawings, there is a macro relational structure (such as a grid) which constrains many of the nodes at a micro level. This results in non-adjacent nodes being drawn close together, in two dimensions, even though they are graph theoretically far apart. The results of the $Min\text{-}d/Max\text{-}dim$ aesthetic measure support the use of high level visual précis given the overall resolution of the underlying drawings is low. Here the graph drawings show the overall structure but this ignores the time it takes to render such large drawings. Given the average inter-node distance is quite small and visual clumping takes place, the visual précis simply exploits this fact when forming the visualization.

Graph Name	Nodes	Edges	Crossings	Aspect Ratio	Min−d / Max-dim	Min−d / AvgEdge	MinEdge / MaxEdge	MinEdge/ AvgEdge
dwa512	512	1004	621	1.16	0.	.012	.4	.553
1138bus	1138	1458	756	1.077	.002	.063	.01	.063
rdb1250	1250	3025	1152	1.34	.006	.239	.494	.664
dw2048	2044	4079	4207	1.719	0.	.013	.105	.34
qh1484	1482	2488	1186	1.029	.002	.068	.028	.221
dwg961a	706	1351	0	1.427	.01	.401	.27	.401
nos4	100	247	117	1.47	.009	.113	.121	.232
nos7	729	1944	6610	1.281	.01	.164	.988	.991
bfw782a	782	3394	7442	1.059	.001	.014	.004	.014
fidap005	27	126	398	1.947	.021	.092	.049	.092
plsk362	362	2712	14576	1.203	.002	.022	.009	.022
sherman4	1104	1468	1040	1.004	0.	.127	.008	.127
plat1919	1919	15240	98153	1.466	0.	.019	.006	.019
rdb968	968	2332	882	1.047	.013	.4	.926	.96
olm1000	1000	1997	55	1.21	.001	.141	.075	.141
olm1000	1919	4831	0	1.031	.004	.283	.129	.283
bfw62a	62	200	356	1.133	.019	.134	.052	.134
bfw398a	398	1256	2968	1.675	.004	.115	.294	.454
bfw782b	782	2600	3586	1.836	.001	.061	.03	.063
dwb512	512	1024	853	1.018	.014	.303	.073	.303
fidapm02	200	2805	388466	1.031	.003	.014	.005	.014
plat362	362	2712	14502	1.298	.004	.06	.025	.06
qh768	768	1322	721	1.528	.002	.089	.036	.185
qh882	882	1533	1175	1.045	.002	.084	.036	.164
rdb1250l	1250	3025	1152	1.102	.006	.239	.331	.507
rdb200	200	460	162	1.038	.03	.399	.988	.991
rdb450	450	1065	392	1.006	.019	.391	.977	.986
rdb200l	200	460	162	1.044	.03	.407	.988	.992
rdb800l	800	1920	722	1.012	.005	.135	.527	.698
dw8192	8184	17371	12149	1.849	0.	.008	.02	.041
rdb2048	2048	4992	1922	1.122	.004	.173	.232	.379
rdb2048l	2048	4992	1922	1.082	.003	.168	.232	.378
cry10000	3699	7164	0	2.084	.002	.192	.178	.256
rdb3200l	3200	7840	3067	1.117	.003	.164	.173	.295
bcspwr07	1612	2106	699	1.135	.002	.174	.024	.205
bcspwr09	1723	2394	2604	1.164	.004	.175	.07	.28
bcspwr10	5296	8260	25770	1.364	0.	.027	.004	.071

Table 6.3: Graph Drawing Aesthetic Measures of the final drawings

6.7 Results: Horizon Drawings (*Visual Précis*)

Figures 6.70 and 6.89 show two detailed examples of multiple horizons from a hierarchical compound graph of `qh1484` and `bcspwr09` drawn as visual précis. For each horizon the associated space decomposition, to the depth of that horizon, is also shown. These examples clearly shows how groups of nodes are approximated and hence the implied edges between these groups are created. The overall structure of the graph drawing is retained while progressively more abstract views of the data are formed. The visual weight of the highest level horizons is a small faction of the total number of nodes and edges that are drawn in the Figures 6.66 and 6.78.

Example three dimensional visual précis extracted from a hierarchical compound graph of each graph after a wavefront coupled with a number of iterations of FADE3D have been applied are shown in Figures 4.27, 4.28, 4.29, 4.33, 4.34 and 4.39. The aim here is to assess whether such abstract representations of structured, semi-structured and clustered graphs are of sufficient quality to represent the underlying graph drawings.

The number of possible précis that can be extracted from a hierarchical compound graph is very large. Here we restrict our interest to the horizon level précis and the drawings of the underlying graphs.

Table 6.4 shows the results of applying a range of graph drawing aesthetic measures to each horizon level of the hierarchical compound graph. These measures are averaged over 100 FADE2D drawings of each graph. The number of clusters $\mathcal{C}$, implied edges $\mathcal{I}$, nodes $\mathcal{N}$ and edges $\mathcal{E}$ is shown for each level, including the lowest level (that is, the entire graph).

Table 6.4: Graph Drawing Aesthetic Measures of Visual Horizons of Matrix Market Graphs

Graph Name	Horizon Level ($\mathcal{HL}$)	Clusters ($\mathcal{C}$)	Implied Edges ($\mathcal{I}$)	Nodes ($\mathcal{N}$)	Edges ($\mathcal{E}$)	% Visual Weight ($\mathcal{VW}$)	Crossings ($\mathcal{X}$)	Aspect Ratio ($\mathcal{AR}$)	Min–d / Max-dim ($\mathcal{MD}$)	Min–d / AvgEdge ($\mathcal{ME}$)	MinEdge / MaxEdge ($\mathcal{MM}$)	MinEdge/ AvgEdge ($\mathcal{MA}$)
1138bus	2	4	5	0	0	.3	0	1.182	.776	.802	.598	.802
	3	16	28	0	0	1.7	1	1.094	.155	.488	.345	.488
	4	55	105	0	0	6.2	8	1.052	.047	.331	.17	.331
	5	170	346	7	0	20.1	64	1.08	.023	.32	.108	.321
	6	359	738	186	52	51.4	223	1.08	.01	.23	.066	.303
	7	87	169	961	1088	88.8	550	1.077	.005	.168	.029	.168
	8	4	11	1130	1441	99.6	754	1.077	.004	.118	.029	.171
	9	0	0	1138	1458	100.	756	1.077	.002	.063	.01	.063
dw2048	2	4	5	0	0	.1	0	1.992	.476	.637	.476	.637
	3	8	9	0	0	.3	0	2.256	.238	.734	.56	.734
	4	23	45	0	0	1.1	1	1.93	.073	.417	.269	.417
	5	68	161	1	0	3.7	8	1.735	.033	.362	.235	.362
	6	214	587	9	0	13.2	37	1.662	.014	.294	.19	.396
	7	557	1559	229	158	40.8	177	1.696	.007	.304	.097	.305
	8	265	929	1304	2022	73.6	983	1.719	.001	.066	.077	.263
	9	146	603	1686	3030	89.	2195	1.719	.001	.051	.103	.335
	10	70	269	1900	3637	95.7	3275	1.719	.001	.031	.095	.309
	11	16	85	2016	3970	99.1	4034	1.719	0.	.013	.105	.34
	12	2	15	2044	4079	100.	4207	1.719	0.	.013	.105	.34
qh1484	2	4	5	0	0	.2	0	1.144	.634	.697	.477	.697
	3	14	22	0	0	.9	1	1.006	.205	.67	.43	.67
	4	47	86	0	0	3.3	5	1.024	.07	.493	.308	.493
	5	142	280	6	1	10.8	60	1.044	.01	.15	.064	.15
	6	363	781	67	16	30.8	124	1.029	.01	.261	.076	.261
	7	353	907	730	663	66.7	536	1.024	.005	.196	.027	.196
	8	33	143	1417	2264	97.	1128	1.029	.002	.101	.028	.219
	9	0	0	1482	2488	100.	1186	1.029	.002	.068	.028	.221
sherman4	2	4	1	0	0	.2	0	1.38	.346	1.	.908	.908
	3	16	4	0	0	.8	0	1.026	.075	.297	.155	.297
	4	59	9	2	0	2.7	1	1.054	.035	.373	.217	.373
	5	146	16	48	0	8.2	1	1.008	.017	.338	.248	.46
	6	109	26	261	1	15.4	1	1.002	.007	.254	.206	.46
	7	56	36	419	15	20.5	1	1.004	.001	.049	.016	.049
	8	173	511	541	86	51.	87	1.004	.001	.174	.017	.174
	9	94	384	909	728	82.2	495	1.004	0.	.173	.013	.191

Table 6.4: *continued*

Graph	$\mathcal{L}$	$\mathcal{C}$	$\mathcal{I}$	$\mathcal{N}$	$\mathcal{E}$	$\mathcal{VW}$	$\mathcal{X}$	$\mathcal{AR}$	$\mathcal{MD}$	$\mathcal{ME}$	$\mathcal{MM}$	$\mathcal{MA}$
	10	0	0	1104	1468	100.	1040	1.004	0.	.127	.008	.127
plat1919	2	4	6	0	0	.1	1	1.585	.464	.663	.393	.663
	3	12	24	0	0	.2	4	1.281	.112	.397	.241	.397
	4	28	65	1	0	.5	10	1.487	.07	.498	.325	.498
	5	86	251	4	0	2.	49	1.455	.027	.377	.353	.506
	6	281	1151	15	1	8.4	861	1.478	.016	.375	.173	.375
	7	587	4180	377	617	33.6	11450	1.471	.002	.057	.023	.063
	8	252	2886	1397	8216	74.3	49949	1.463	.001	.052	.017	.052
	9	44	826	1831	13846	96.4	88356	1.466	.001	.026	.008	.026
	10	12	253	1895	14848	99.1	95641	1.466	.001	.026	.008	.026
	11	0	0	1919	15240	100.	98153	1.466	0.	.019	.006	.019
dw8192	2	4	5	0	0	0.	0	2.055	.455	.613	.409	.613
	3	12	22	0	0	.1	1	1.564	.158	.593	.354	.593
	4	34	73	1	0	.4	3	1.721	.07	.518	.33	.518
	5	107	265	2	0	1.5	8	1.809	.019	.274	.179	.274
	6	378	1015	9	0	5.5	32	1.867	.015	.418	.274	.418
	7	1401	3915	56	8	21.	241	1.842	.007	.381	.188	.381
	8	2454	7279	2148	2317	55.5	1804	1.852	0.	.043	.045	.111
	9	1329	4769	5442	8619	78.8	5113	1.849	0.	.014	.06	.122
	10	598	2846	6986	12832	90.9	8617	1.849	0.	.014	.059	.122
	11	193	1099	7806	15755	97.1	10978	1.849	0.	.014	.02	.041
	12	4	31	8184	17371	100.	12149	1.849	0.	.008	.02	.041
bcspwr09	2	4	4	0	0	.2	0	1.122	.754	.914	.747	.914
	3	14	22	0	0	.9	0	1.005	.153	.49	.295	.49
	4	42	85	2	0	3.1	5	1.061	.043	.301	.275	.46
	5	123	252	12	0	9.4	17	1.18	.016	.239	.128	.239
	6	322	731	86	9	27.9	77	1.168	.008	.237	.152	.38
	7	460	1161	587	351	62.2	581	1.166	.005	.194	.112	.433
	8	135	422	1449	1652	88.9	1589	1.166	.004	.187	.07	.279
	9	0	0	1723	2394	100.	2604	1.164	.004	.175	.07	.28
bcspwr10	2	4	3	0	0	.1	0	2.063	.336	.709	.616	.709
	3	10	12	1	0	.2	0	1.385	.125	.551	.422	.619
	4	23	42	1	0	.5	6	1.382	.04	.298	.131	.298
	5	62	137	2	0	1.5	27	1.406	.016	.233	.083	.233
	6	176	465	9	0	4.8	155	1.389	.011	.274	.063	.306
	7	495	1567	72	12	15.8	1245	1.362	.004	.151	.019	.151
	8	1055	3554	519	239	39.5	5095	1.369	.001	.056	.012	.143
	9	1213	4024	2072	1739	66.7	10592	1.362	.001	.066	.006	.093
	10	586	2182	4053	5022	87.3	17248	1.363	0.	.042	.004	.068
	11	67	293	5165	7885	98.8	24978	1.364	.001	.053	.004	.07
	12	2	11	5296	8260	100.	25770	1.364	0.	.027	.004	.071
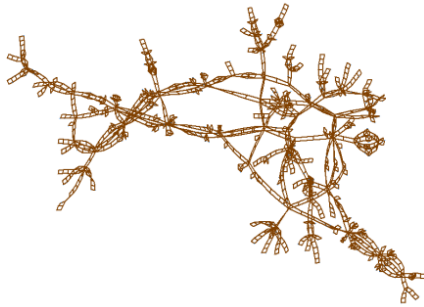
Figure 6.66: Plain `qh1484` graph view
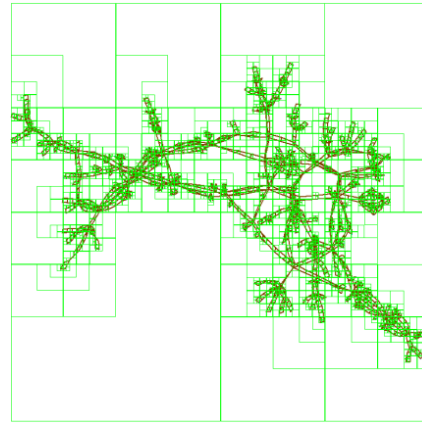
Figure 6.67: View of `qh1484` with quadtree
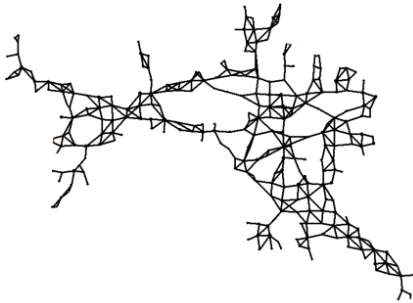
Figure 6.68: Visual Précis of a 5th Level Horizon of `qh1484`
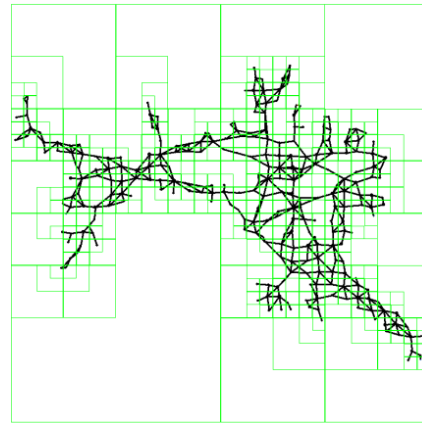
Figure 6.69: Visual Précis of a 5th Level Horizon of `qh1484` with quadtree
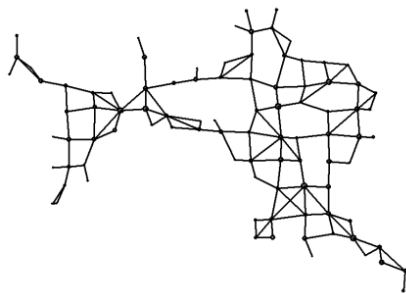
Figure 6.70: Visual Précis of a 4th Level Horizon of `qh1484`
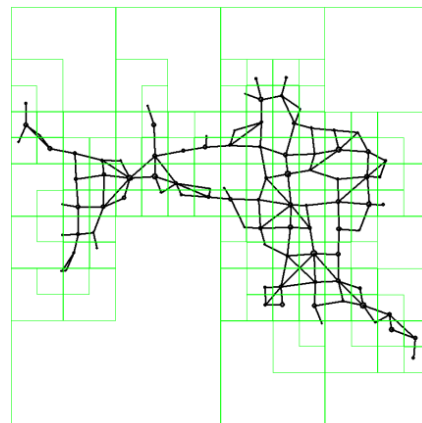
Figure 6.71: Visual Précis of a 4th Level Horizon of `qh1484` with quadtree
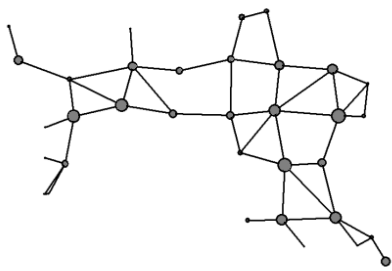
Figure 6.72: Visual Précis of a 3rd Level Horizon of `qh1484`
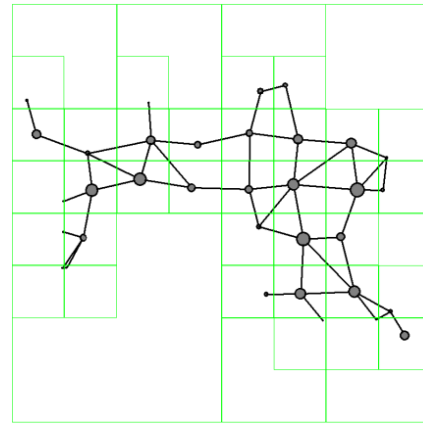
Figure 6.73: Visual Précis of a 3rd Level Horizon of `qh1484` with quadtree
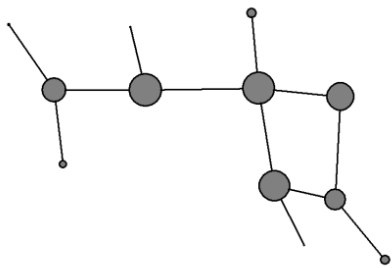
Figure 6.74: Visual Précis of a 2nd Level Horizon of `qh1484`
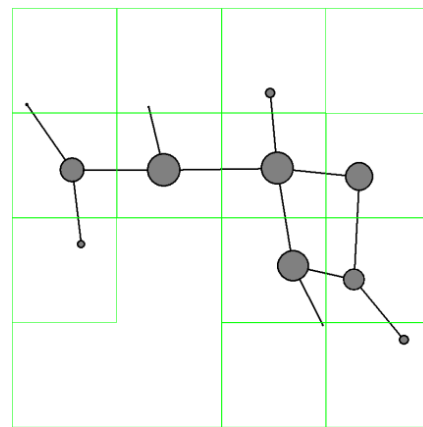
Figure 6.75: Visual Précis of a 2nd Level Horizon of `qh1484` with quadtree
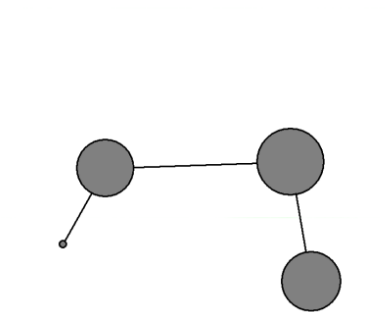
Figure 6.76: Visual Précis of a 1st Level Horizon of `qh1484`
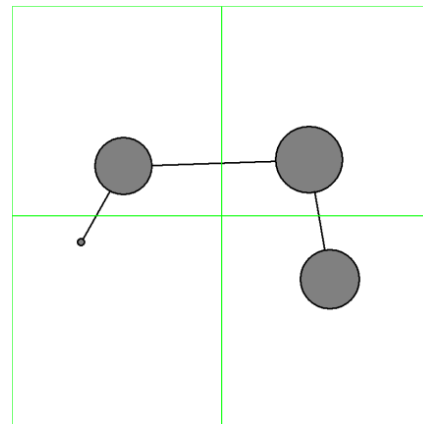
Figure 6.77: Visual Précis of a 1st Level Horizon of `qh1484` with quadtree
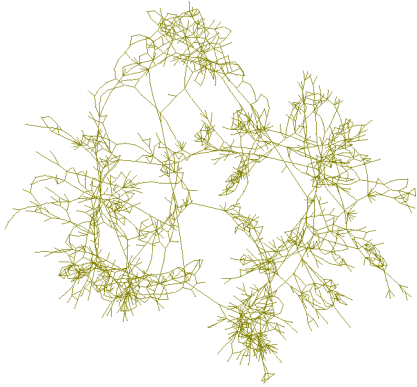
Figure 6.78: Plain `bcspwr09` graph view
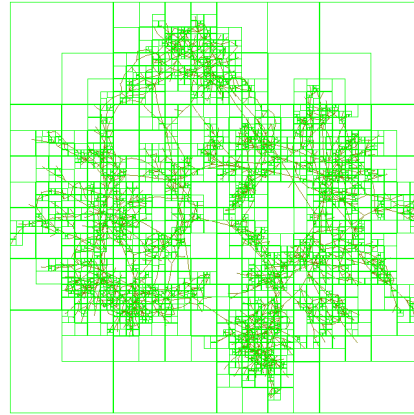
Figure 6.79: View of `bcspwr09` with quadtree
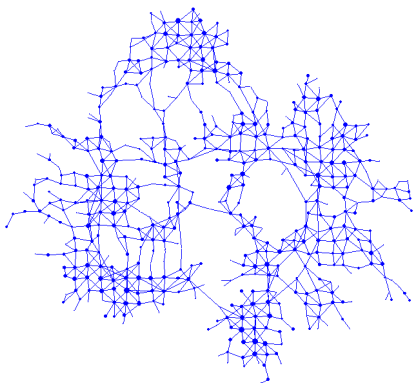
Figure 6.80: Visual Précis of a 5th Level Horizon of `bcspwr09`
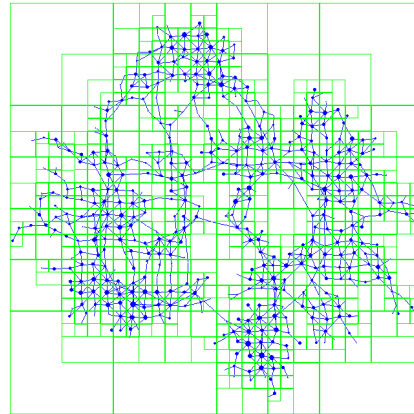
Figure 6.81: Visual Précis of a 5th Level Horizon of `bcspwr09` with quadtree
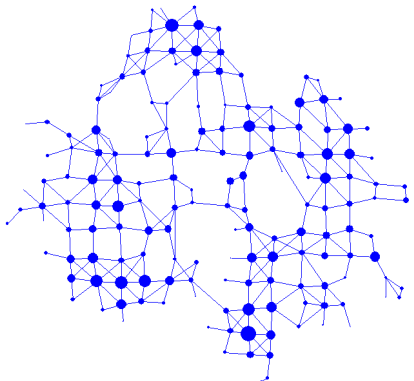
Figure 6.82: Visual Précis of a 4th Level Horizon of `bcspwr09`
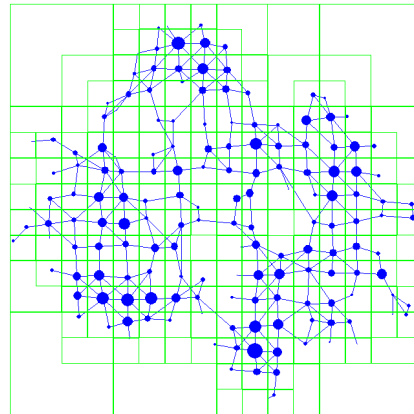
Figure 6.83: Visual Précis of a 4th Level Horizon of `bcspwr09` with quadtree
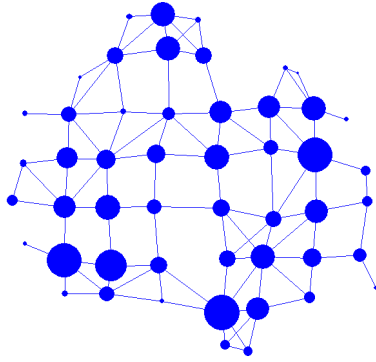
Figure 6.84: Visual Précis of a 3rd Level Horizon of `bcspwr09`
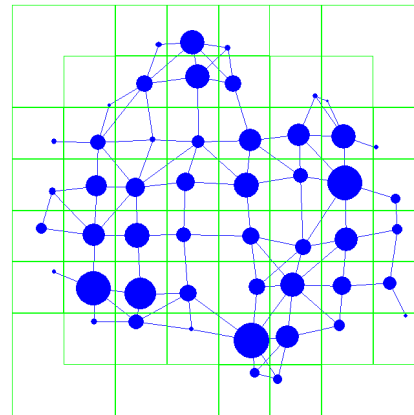
Figure 6.85: Visual Précis of a 3rd Level Horizon of `bcspwr09` with quadtree
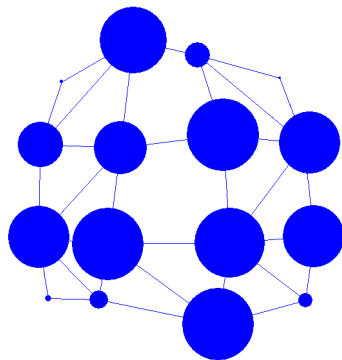
Figure 6.86: Visual Précis of a 2nd Level Horizon of `bcspwr09`
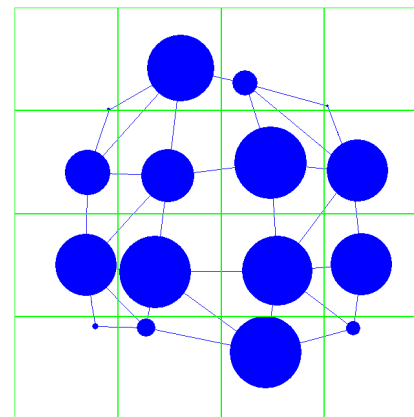
Figure 6.87: Visual Précis of a 2nd Level Horizon of `bcspwr09` with quadtree
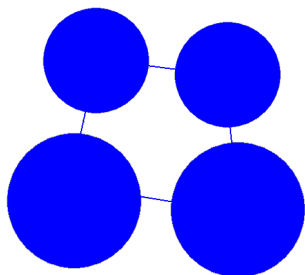
Figure 6.88: Visual Précis of a 1st Level Horizon of `bcspwr09`
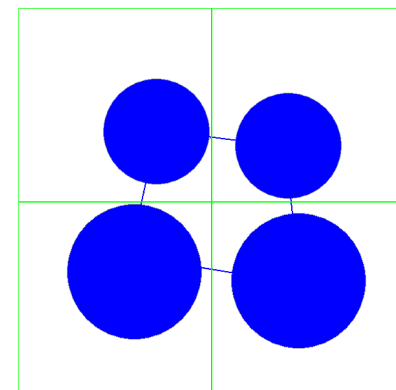
Figure 6.89: Visual Précis of a 1st Level Horizon of `bcspwr09` with quadtree

6.7.1 Discussion

The quality of these horizon drawings are measured using a variety graph drawing aesthetic measures. These abstract representations should also adhere to the multilevel aesthetic measures as described in Section 2.2.2 and previously discussed in Section 5.6.1.

These results indicate that the visual abstractions used in the FADE paradigm greatly reduce the number of edge crossings. However it is also visually apparent in some visual précis that due to the nature of the regular structured graph data in this case study, the two dimensional précis consist of pseudonodes with many non-adjacent nodes. The result of the micro relational structures being elided in such drawings, is that the overall regular structure of the drawing is not apparent in the visual précis.

Some remarks are in order.

- For clustered data, the higher level précis do contain the natural clusters of the graphs which are well displayed by the visualization.

- The use of a post-processing algorithm, such as an inertial bisection method [212], to determine the principal axis of inertia may improve the regularity of the decomposition and hence the visual précis formed.

- The three dimensional visual précis display an approximate superstructure or backbone for the graph drawing.

- The visual weight of high level visual précis from the hierarchical compound graphs produced by both FADE2D and FADE3D, are a small fraction of the number of nodes of edges in the underlying drawing.

We can deduce a clear guideline for graph drawing systems using the FADE paradigm: the use of two dimensional regular space decomposition, for the creation of a hierarchical compound graph, can produce visual précis that do not display any regular repeating structures found in the underlying drawings. However, for graphs with clustered regions, the visual précis formed do provide a good approximation to the overall structure of the layout.

6.8   Results: Time and Error Performance

Here we present the results of a performance and error analysis of the FADE2D algorithm with a $Min\text{-}d$ cell opening criterion and varying values of θ on the range matrices in this case study.

Figure 6.90 to Figure 6.93 compare the percentage of approximate force calculations to direct calculations, for a range of θ values, against the error in the force calculation. The following two measures appear in each chart:

- The percentage of approximate to direct calculations for FADE2D is shown as the purple line .

- The approximate as compared with the direct *node-to-node* nonedge force calculation is shown as the blue line . These results are averaged in two ways. First the error, for a given θ, is averaged over a sampling of the application of FADE2D from the first 400 runs. Second, the initial layout is randomized 100 times and the above average is computed again for each initial layout. The average across these 100 runs is the value shown.

Figure 6.94 to Figure 6.99 show charts of the performance versus error tables, for some of the graphs in this case study. Further charts are shown in Appendix B.

As in the case study in Chapter 5, the following three measures appear in each chart:

- The time per iteration of FADE2D is shown as the blue line .

- The nonedge error as compared with the direct *node-to-node* nonedge force calculation is shown as the orange line . These results are averaged in two ways. First the error, for a given θ, is averaged over a sampling of the application of FADE2D from the first 400 runs. Second, the initial layout is randomized 100 times and the above average is computed again for each initial layout. The average across these 100 runs is the value shown.

- The brown line corresponds to the averaged error for the first 400 iterations of FADE2D.
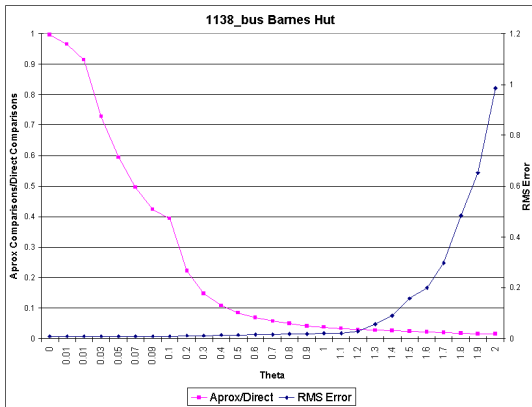
Figure 6.90: Percentage of Force Calculation versus Error of FADE2D for `1138bus`
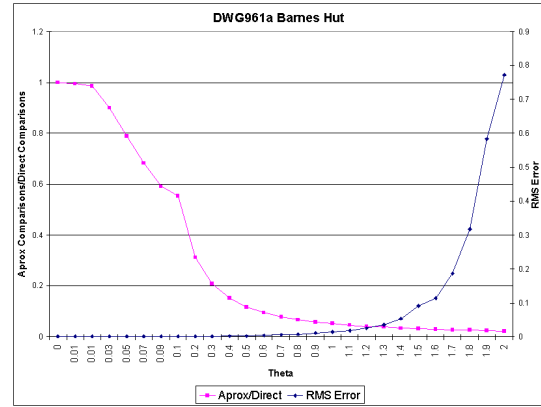
Figure 6.91: Percentage of Force Calculation versus Error of FADE2D for `dwg961a`
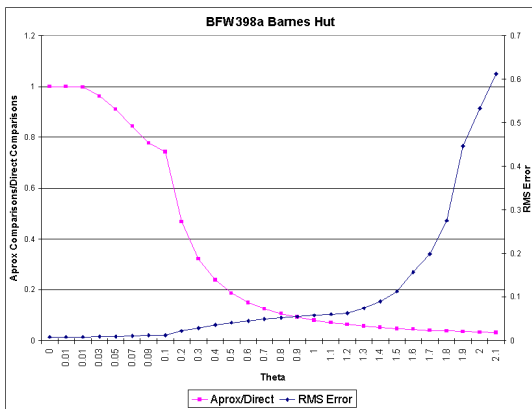
Figure 6.92: Percentage of Force Calculation versus Error of FADE2D for `bfw398a`
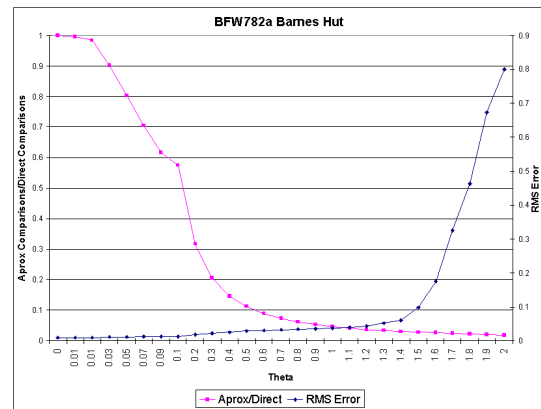
Figure 6.93: Percentage of Force Calculation versus Error of FADE2D for `bfw782a`
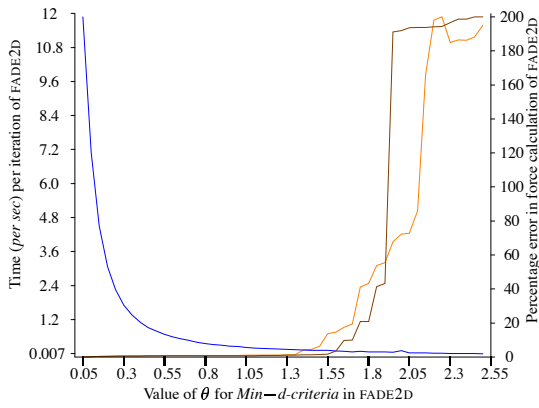
Figure 6.94: Performance versus Error of FADE2D for `dw2048`
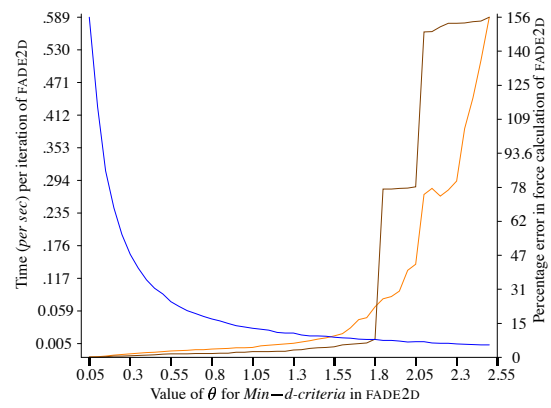
Figure 6.95: Performance versus Error of FADE2D for `rdb450`
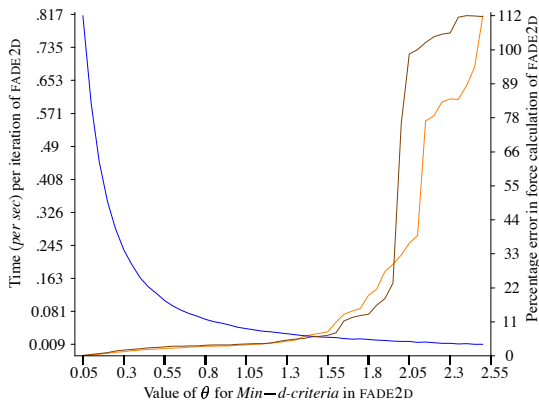
Figure 6.96: Performance versus Error of FADE2D for `dwa512`
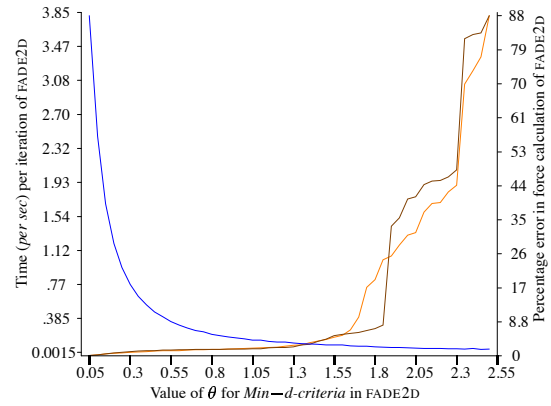
Figure 6.97: Performance versus Error of FADE2D for `1138bus`
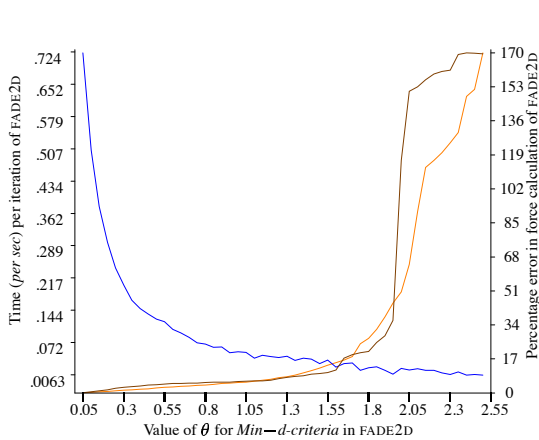
Figure 6.98: Performance versus Error of FADE2D for `bfw398b`
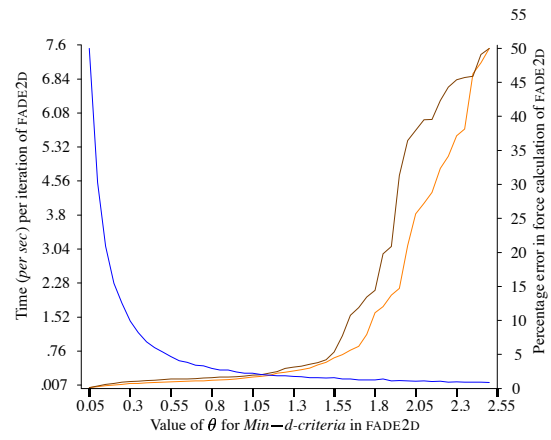
Figure 6.99: Performance versus Error of FADE2D for `qh1484`

6.8.1   Discussion

As in the previous case study the performance of FADE2D improves as the value for θ decreases. And the error in the force calculation increases as θ increases. These values are in line with the predicted values from particle simulation work. Again these results indicate that FADE is not suitable for small graphs. As in the previous case study the results indicate that a value of θ in the range 1.3-1.8 is permissible given that the edge forces dampen these nonedge force errors.

The regular graphs in this case study are drawn approximately uniformly distributed at each step in the progressive cycle, as such the performance of FADE2D in the initial and averaged runs are approximately equal.

6.9   Results: Clustering Measures

Here we present the results of applying the hierarchical compound graph quality measures, introduced in Section 3.5, to the layouts produced across the first 300 iterations of the FADE2D algorithm. Here we use the $Min\text{-}d$ cell opening criterion and a value of $\theta = 0.8$.

Figure 6.100 to Figure 6.111 show charts of the normalized clustering measures versus crossings, the remaining charts are shown in Appendix B.

As in the previous case study these results are averaged over 40 sets of runs of the FADE2D algorithm. In each case, the graph is given an initial random layout.

The values of hierarchical compound graph quality measures are indicated as normalised values according to the right hand axis. Each measure is colour coded as follows:

- The Implied Edge Precision ($\mathcal{IEP}$) measure is drawn with an orange line.

- The Lowest Common Ancestor ($\mathcal{LCA}$) measure is drawn with an blue line.

- The Coupling and Cohesion ($\mathcal{COCO}$) measure is drawn with an red line.

- The Node Neighbourhood Similarity ($\mathcal{NNS}$) measure is drawn with a brown line.

The number of edge crossing in the layout is indicated on the left hand axis and is drawn as a green line in each chart.
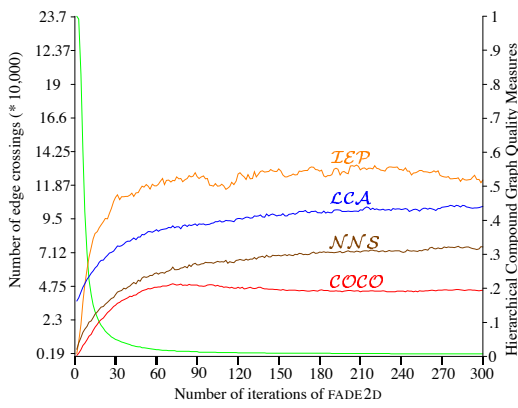
Figure 6.100: *HCGQM* versus Crossings for `bcspwr09`
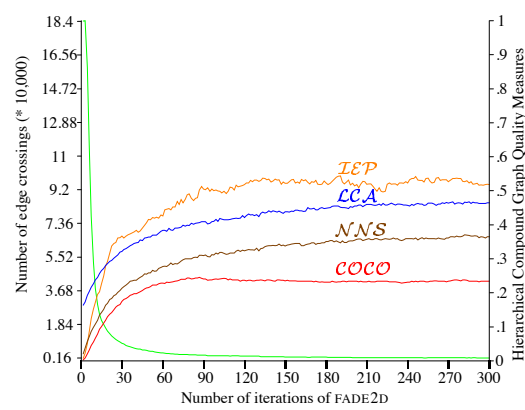
Figure 6.101: *HCGQM* versus Crossings for `bcspwr07`
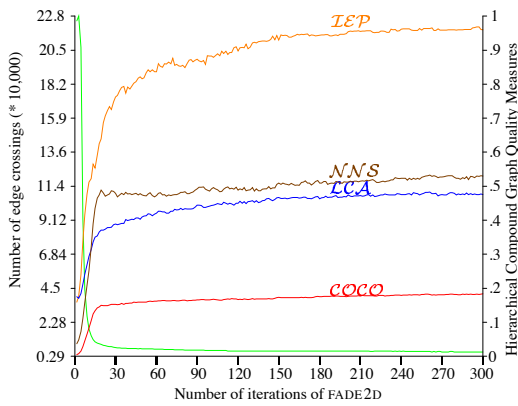
Figure 6.102: *HCGQM* versus Crossings for `rdb968`
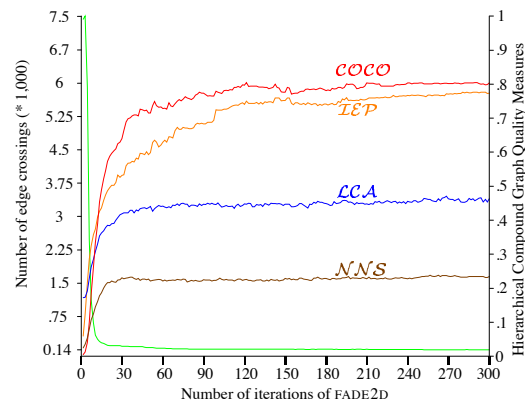
Figure 6.103: *HCGQM* versus Crossings for `sherman4`
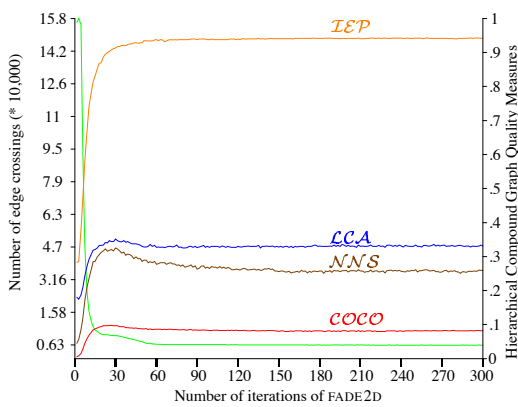
Figure 6.104: *HCGQM* versus Crossings for `nos7`
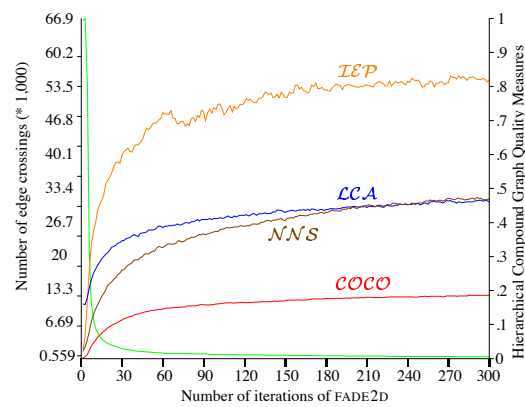
Figure 6.105: *HCGQM* versus Crossings for `dw2048`
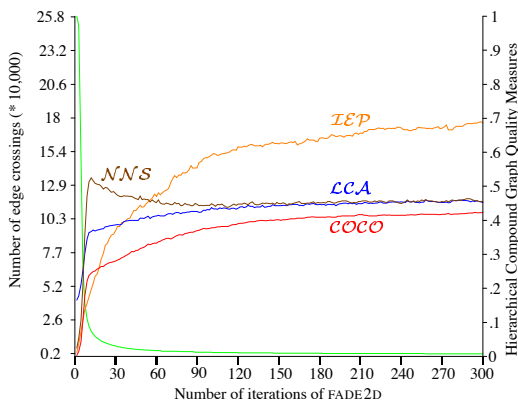
Figure 6.106: *HCGQM* versus Crossings for `qh1484`
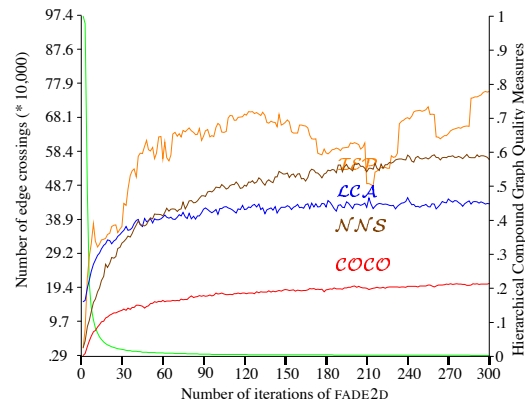
Figure 6.107: *HCGQM* versus Crossings for`plsk1919`
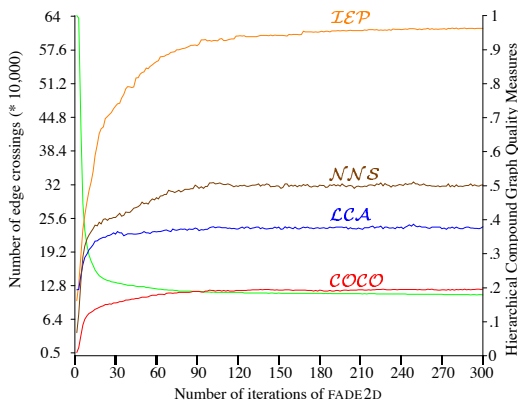
Figure 6.108: *HCGQM* versus Crossings for `bfw398b`
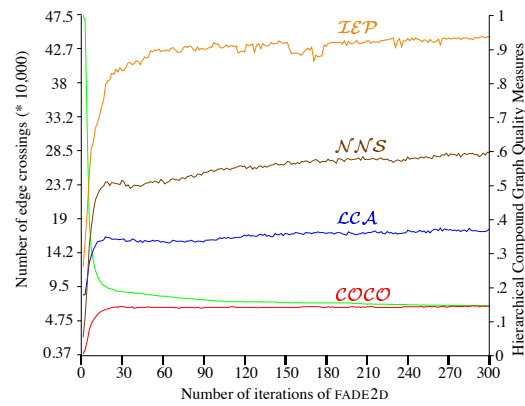
Figure 6.109: *HCGQM* versus Crossings for `bfw782a`
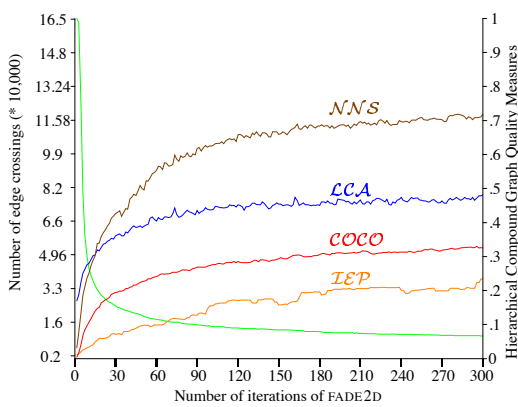
Figure 6.110: *HCGQM* versus Crossings for `olm1000`
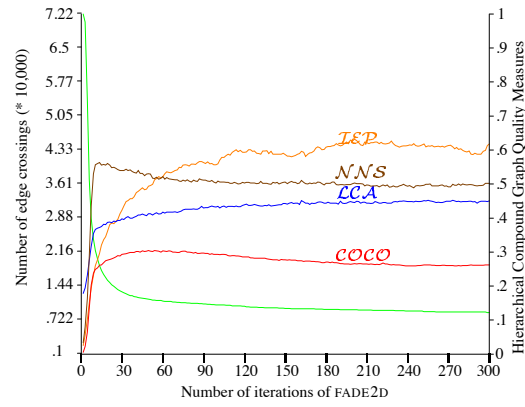
Figure 6.111: *HCGQM* versus Crossings for `qh768`

6.9.1 Discussion

The hierarchical compound graph quality measures have been applied during the first 300 iterations of FADE2D in the progressive cycle. In this discussion our aim is to evaluate, in general terms, the usefulness of our hierarchical compound graph quality measures for these graphs, when much of the data contains little or no clustering, in the traditional sense. Each chart also shows that the number of edge crossings in the layout steadily decreases as the progressive cycle iterates.

Much of this data is structured and semi-structured, with similar cohesion and coupling, our classical geometric Coupling and Cohesion measure ($\mathcal{COCO}$) provides a good indication as to relative amount of clustering that is visually apparent. The results of the $\mathcal{COCO}$ measure in this case study, are in stark contrast to the results in the software visualization case study. Here the underlying data often contains no structural clusters and as such exhibits low values for the $\mathcal{COCO}$ measure.

The $\mathcal{IEP}$ often provides a better indication as to the progress of the progressive cycle. Here the $\mathcal{IEP}$ measure is often a good indicator as the relative stability of the hierarchical compound graph and its suitability for rendering as visual précis.

The $\mathcal{LCA}$ measure is difficult to interpret. It should provide an indication as to the clustering depth of each edge in the hierarchical compound graph, that is, on average how far up the tree do edges cause implied edges. As in the previous case study these results indicate that the $\mathcal{LCA}$ does improve as the layout improves but that the measure itself is on an ordinal scale due to the poor normalization.

For several of the structured graphs the $\mathcal{NNS}$ provides a measure for how strongly interconnected nodes are within an area.

Overall, the measures steady increase to a plateau which supports our hypothesis that as the quality of the drawing improves (as measured by edge crossings here), the quality of the clustering exhibited by the layout also improves. Some of these measures are absolute, others are ordinal due to the poor normalization used.

6.10   Remarks

This chapter has presented a matrix market visualization case study using the FADE paradigm for large graph drawing. These results indicate that the FADE paradigm is very suitable for drawing large graphs from these application domains. The use of three dimensional visual précis is also quite promising and the $\mathcal{IEP}$ tends to indicate highly accurate hierarchical compound graphs are created by the geometric clustering method used in FADE.

The progressive cycle proceeds by making small iterative changes to the layout of the graph which improves the values of the hierarchical compound graph quality measures. Drawings of such layouts tend to exhibit patterns in the edge strengths, natural clusterings, macro and repeating micro-structures, and overall symmetries in the graph.

Overall, we have demonstrated the suitability of the FADE paradigm in providing fast layouts of these graphs coupled with accurate abstract representations.

Conclusions and Future Work

"In the final analysis, a drawing simply is no longer a drawing, no matter how self-sufficient its execution may be. It is a symbol, and the more profoundly the imaginary lines of projection meet higher dimensions, the better." - Paul Klee

7.1 Conclusions

We began by identifying four related problems which are inherent when dealing with the visualization of large graphs: time to compute a layout, the use of screen space, the cognitive load on the user, and the time to render the picture. We introduced and evaluated the hierarchical compound graph model and the FADE visualization paradigm. The model itself is both a geometric representation of a graph layout coupled with a hierarchical geometric clustering of the nodes of the graph. FADE marries rapid graph drawing, geometric clustering, visual abstraction and measurement using a single graph model. This visualization paradigm supports the needs inherent in large scale relational information visualization.

The use of this model helps to address the time taken to compute a high quality layout of the graph. Part of the hierarchical compound graph model consists of an inclusion tree of the nodes of the graph. This inclusion tree is used by force directed algorithms to approximate the nonedge forces, which results in a more efficient layout algorithm. In general, the performance improvement of the layout algorithms in the FADE paradigm comes from computing nonedge forces using a recursive approximation of groups of nodes, rather than all the *node-to-node* nonedge forces directly. The time performance versus the

error in the force computation of these algorithms are evaluated in our case studies.

It was noted how this model allows us to make more effective use of screen space. Extracted from this model are précis which we render as visual précis in two and three dimensions. High level précis form very approximate views of the underlying graph but generally have a visual weight which is a small fraction of the underlying graph drawing. This approach allows for the drawing of abstract representations with good resolution. For high level views, the nodes and edges of the visual précis can be clearly identified in the drawing. The aesthetics of the drawings and a range of visual précis generated from the FADE paradigm are evaluated in our case studies.

By reducing the size of the graph and drawing more abstract views on screen we have reduced the direct cognitive load on the user. If the précis accurately reflects the structures and connectivity in the underlying graph then the cost in comprehending this abstract representation is minimized. Example drawings along with the visual weights of a range of visual précis are presented in our case studies.

The smaller visual précis, which can represent many thousands of nodes and edges are also computationally inexpensive to render in two and three dimensions. Other forms of visual précis, which show the local and global neighbourhoods of a node or set of nodes were also introduced.

We have married the hierarchical compound graph with the methods introduced to address the four problems of; computation, screen space, cognitive load, and rendering. Each of these issues have associated hard measures that we evaluate against application domains in our case studies. Our general conclusion is that large scale relational visualization is both feasible and practical.

7.2 Future Work

There are still many challenges in large scale relational information visualization. The work in this thesis suggests several future research directions of both theoretical and practical significance.

- FADE currently is suitable for the visualization of simple undirected graphs with

singularly attributed nodes and edges. It may be possible to extend this work to more complex graphs model, in particular, it may be possible to visualize directed graphs, where the direction of flow within the graph or precedence relationships in the graph are important. Further, it may be possible to handle more highly attributed graphs with the use of more long range forces or short range forces for *edge-to-edge* and *node-to-node* interactions, within the FADE paradigm.

- The FADE paradigm uses regular space decompositions in creating the hierarchical compound graphs. Other methods which generate regular and non-regular inclusion trees of the nodes of the graph, can be integrated into the FADE paradigm. To accommodate this only the formulation of the width of the cell or the introduction of new cell opening criterion is required, such as those outlined in Chapter 4. Research based on the FADE paradigm, with recursive Voronoi decomposition of space is currently underway [228].

- The layout algorithms in the FADE paradigm can be extended to use multipole expansion, rather than monopole calculations. The Fast Multipole Method uses large pseudo-nodes (high θ) and is an elegant refinement to the basic tree code method in FADE. As in all tree code methods, limiting the error is crucial, whereas in FADE the errors are dampened by the edge forces. Multipole methods have an asymptotic complexity of $O(n)$. Integrating such methods along with non-regular space decompositions may improve both the performance in the force calculation and the quality of the hierarchical compound graphs generated.

- The reduction in cognitive load in using visual précis to abstractly represent large graph drawings has the cost of removing detail. The use of visual précis and the FADE paradigm in any application requires a formal HCI study. This study should measure both the appropriateness and effectiveness of this paradigm, in accurately representing and abstracting large amounts of relational information.

- Although the FADE paradigm currently deals with the visualization of static data, the underlying hierarchical compound graph model is suitable for dynamic graph envi-

ronments. Streaming data, such as network traffic, might be visualized with a modi-
fied FADE paradigm which accommodates nodes being added, deleted, or collapsed
along with edge creation and deletion. The high level visual précis may likewise be
suitable in such environments, where micro changes to the underlying data are not
as important as the overall change to the graph structure and node distribution.

- A comparative study, in terms of computational speed, iterations required, and the
  quality of the drawings produced by the layout methods presented in this thesis along
  with those in [98], [115],and [283] should prove useful. Currently the method de-
  scribed in [115] does not have a publicly available implementation.

- As part of Koschke's research [163], a manual analysis of the resource flow graphs
  used in Chapter 5 was undertaken. The result of this analysis produced a set of refer-
  ence atomic components, that are used in the comparison of different automatic and
  semi-automatic component identification techniques. Integrating this result data into
  a visualization of the graphs allows us to compare the natural clusters identified with
  the actual human identified software components in the system. An investigation of
  correlations here would be useful.

- On a more practical note, as noted in Section 3.11 the visual précis offered by the
  hierarchical compound graph model need to be integrated into an interactive visual-
  ization environment, if they are to be of practical use. Animation, morphing, fading
  or other suitable visualization techniques can be used to allow a user to move be-
  tween visual précis. With appropriate visualization, actions such as moving between
  horizon layers, viewing sets of nodes with cut views, or digging down into sections
  of the hierarchical compound graph can be supported. Further, integrating multiple
  views of the graph with surface views may allow users to navigate and explore large
  graphs in new ways.

Bibliography

[1] Adamic, L. A., "The small world web," *Proc. 3rd European Conf. Research and Advanced Technology for Digital Libraries, ECDL*, edited by S. Abiteboul and A. Vercoustre, Vol. 1696 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–22 Sep. 1999, pp. 443–452.

[2] Alberts, D., Gutwenger, C., Mutzel, P., and Näher, S., "AGD-library: A library of algorithms for graph drawing," Research Report MPI-I-97-1-019, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, Sep. 1997.

[3] Ali, J. and Nishinaka, Y., "Using 3d visualization for understanding java class libraries," *Proc. Software Visualization Work.*, edited by A. Quigley, Department of Computer Science, University of Technology, Sydney, Australia, Dec 1999, pp. 17–22.

[4] Anderberg, M. R., *Cluster Analysis for applications*, Academic Press, 1973.

[5] Anderson, R. J., "Tree data structures and n-body simulation," *SIAM J. Comput.*, Vol. 28, No. 6, June 1999, pp. 1923–1940.

[6] Anderson, W. N. and Morley, T. D., "Eigenvalues of the Laplacian of a graph," *Linear and Multilinear Algebra*, Vol. 18, 1985, pp. 141–145.

[7] Appel, A., "An efficient program for many-body simulation," *SIAM J. Sci. Statist. Comput.*, Vol. 6, 1985, pp. 85–103.

[8] Arabie, P., Hubert, L. J., and Soete, G. D., *Clustering and Classification*, World Scientific Press, 1996.

[9] Armstrong, M. N. and Trudeau, C., "Evaluating architectural extraction tools," *Working Conference on Reverse Engineering*, IEEE Computer Society, IEEE Computer Society Press, Hawai, USA, Oct 1998, pp. 30–39.

[10] Arulananthan, A., Johnson, S., McManus, K., Walshaw, C., and Cross, M., "A generic strategy for dynamic load balancing of distributed memory parallel computational mechanics using unstructured meshes," *Parallel Computational Fluid Dynamics: Recent Developments and Advances Using Parallel Computers*, edited by D. R. Emerson et al., Elsevier, Amsterdam, 1998, pp. 43–50, (Proc. Parallel CFD'97, Manchester, 1997).

[11] Baecker, R., DiGiano, C., and Marcus, A., "Software visualization for debugging," *Communications of the Association for Computing Machinery*, Vol. 40, No. 4, Apr. 1997, pp. 44–54.

[12] Ball, T. and Eick, S. G., "Software visualization in the large," *Computer*, Vol. 29, No. 4, Apr. 1996, pp. 33–43.

[13] Baraff, D. and Witkin, A., "Large steps in cloth simulation," *SIGGRAPH 98 Conference Proceedings*, edited by M. Cohen, Annual Conference Series, ACM SIGGRAPH, Addison Wesley, Jul. 1998, pp. 43–54, ISBN 0-89791-999-8.

[14] Barnes, J. and Hut, P., "A hierarchical $o(n \log n)$ force-calculation algorithm," *Nature*, Vol. 324, No. 4, December 1986, pp. 446–449.

[15] Barnes, J. E., *The Formation of Galaxies*, chap. 12, Cambridge University Press, Cambridge, 1995, p. 399.

[16] Batagelj, V., Mrvar, A., and Zaveršnik, M., "Partitioning approach to visualization of large graphs," *Proc. 7th Int. Symp. Graph Drawing, GD*, edited by J. Kratochvíl, Vol. 1731 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 15–19 Sep. 1999, pp. 90–97.

[17] Battista, G., Patrignani, M., and Vargiu, F., "A split and push approach to 3d orthogonal drawing," *Proc. 6th Int. Symp. Graph Drawing, GD*, edited by S. H. White-

sides, Vol. 1547 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 13–15 Aug. 1998, pp. 87–101.

[18] Becker, B. and Hotz, G., "On the optimal layout of planar graphs with fixed boundary," *SIAM Journal on Scientific Computing*, Vol. 16, No. 15, 1987.

[19] Benzi, M., Meyer, C. D., and Tuma, M., "A sparse approximate inverse preconditioner for the conjugate gradient method," *SIAM Journal of Scientific Computing*, Vol. 17, 1996, pp. 1135–1149.

[20] Berg, M. D., Groot, M. D., and Overmars, M., "New results on binary space partitions in the plane," *Computational Geometry Theory Applications*, Vol. 8, 1997, pp. 317–333.

[21] Berg, M. D., van Kreveld, M., Overmars, M., and Schwarzkopf, O., *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 2nd ed., 2000, rev. ed.

[22] Bern, M., Eppstein, D., and J.Gilbert, "Provably good mesh generation," *Computer System Sciences*, Vol. 48, 1994, pp. 384–409.

[23] Bertault, F., "A force-directed algorithm that preserves edge crossing properties," *Proc. 7th Int. Symp. Graph Drawing, GD*, edited by J. Kratochvíl, Vol. 1731 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 15–19 Sep. 1999, pp. 351–358.

[24] Bertault, F. and Miller, M., "Extended graph model for software visualization," *Proc. Software Visualization Work.*, edited by A. Quigley, Department of Computer Science, University of Technology, Sydney, Australia, Dec 1999, pp. 75–81.

[25] Biedl, T. and Kant, G., "A better heuristic for orthogonal graph drawings," *2nd European Symposium on Algorithms*, Vol. 855 of *Lecture Notes in Computer Science*, Springer-Verlag, 1994, pp. 124–135.

[26] Biedl, T. C., "Three approaches to 3D-Orthogonal box-drawings," *Proc. 6th Int.*

Symp. Graph Drawing, GD, edited by S. H. Whitesides, Vol. 1547 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 13–15 Aug. 1998, pp. 30–43.

[27] Biedl, T. C., Madden, B. P., and Tollis, I. G., "The three-phase method: A unified approach to orthogonal graph drawing," *Proc. 5th Int. Symp. Graph Drawing, GD*, edited by G. Di Battista, Vol. 1353 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1997, pp. 391–402.

[28] Blellcoh, G. E. and Narlikar, G. J., "A practical comparison of n-body algorithms," Technical report, Wright Laboratory, 1991.

[29] Boisvert, R. F., Pozo, R., Remington, K., Barrett, R., and Dongarra, J. J., "The Matrix Market: A web resource for test matrix collections," *Quality of Numerical Software, Assessment and Enhancement*, edited by R. F. Boisvert, Chapman Hall, London, 1997, pp. 125–137.

[30] Bosak, J., *Decompositions of Graphs*, Vol. 47 of *Mathematics and Its Applications-East European Series*, Kluwer Academic Publishers, 1st ed., October 1990, ISBN: 079230747X.

[31] Bothun, G., "Fabry-perot images of interacting galaxies," http://zebu.uoregon.edu/mihos.html.

[32] Brandenburg, F., Himsolt, M., and Rohrer, C., "An experimental comparison of force-directed and randomized graph drawing algorithms," *Symposium on Graph Drawing, GD'95*, edited by F. J. Brandenburg, Vol. 1027 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–22 Sep. 1995, pp. 76–87.

[33] Brandenburg, F. J., "Graph clustering I: Cycles of cliques," *Proc. 5th Int. Symp. Graph Drawing, GD*, edited by G. Di Battista, Vol. 1353 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1997, pp. 158–168.

[34] Brandes, U. and Wagner, D., "A bayesian paradigm for dynamic graph layout," *Proc. 5th Int. Symp. Graph Drawing, GD*, edited by G. Di Battista, Vol. 1353 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1997, pp. 236–247.

[35] Brassard, G. and Bratley, P., *Fundamentals of Algorithms*, Prentice Hall, Englewood Cliffs, New Jersey 07632, 1996.

[36] Bridgeman, S. and Tamassia, R., "Difference metrics for interactive orthogonal graph drawing algorithms," *Proc. 6th Int. Symp. Graph Drawing, GD*, edited by S. H. Whitesides, Vol. 1547 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 13–15 Aug. 1998, pp. 57–71.

[37] Brown, M., Domingue, J., Price, B., and Stasko, J., "Software visualization," *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, Vol. 2 of *WORKSHOPS*, 1994, p. 463.

[38] Brown, M. H., Meehan, J. R., and Sarkar, M., "Browsing graphs using a fisheye view," *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems*, Formal Video Programme: Visualization, 1993, p. 516.

[39] Bruß, I. and Frick, A., "Fast interactive 3-d graph visualization," *Symposium on Graph Drawing, GD'95*, edited by F. J. Brandenburg, Vol. 1027 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–22 Sep. 1995, pp. 99–110.

[40] Buchheim, C., Jünger, M., and Leipert, S., "A fast layout algorithm for k-level graphs," *Proc. 8th Int. Symp. Graph Drawing, GD*, edited by J. Marks, Vol. 1984 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–23 Sep. 2000, pp. 229–240.

[41] Bui, T. N., "Improving the performance of the kernigan-lin and simulated annealing graph bisection algorithms," *Proceedings of the ACM/IEEE Design Automation Conference*, ACM Press, 1989, pp. 775–778.

[42] Canfora, G., Sansone, L., and Visaggio, G., "Data flow diagrams: Reverse engineering production and animation," *CSM'92: Proceedings of the 1992 Conference on Software Maintenance, (Orlando, Florida; November 9-12, 1992)*, IEEE Computer Society Press (Order Number 2980), November 1992, pp. 366–375.

[43] Carrier, J., Greengard, L., and Rokhlin, V., "A fast adaptive multipole algorithm for particle simulations," *SIAM Journal on Scientific Computing*, Vol. 9, No. 4, July 1988, pp. 669–686.

[44] Chalmers, M. and Chitson, P., "Bead: Explorations in information visualization," *Proceedings of SIGIR '92, published as a special issue of the SIGIR forum*, ACM Press, June 1992, pp. 330–337.

[45] Chan, T. F., Ciarlet Jr., P., and Szeto, W. K., "On the optimality of the median cut spectral bisection graph partitioning method," *SIAM Journal on Scientific Computing*, Vol. 18, No. 3, May 1997, pp. 943–948.

[46] Chapman, P., Wills, D., Brookes, G., and Stevens, P., "Visualizing underwater environments using multifrequency sonar," *IEEE Computer Graphics and Applications*, Vol. 19, No. 5, Sep. 1999, pp. 61–65.

[47] Charles, A. and So-Yen, Y., "Spectral partititioning: the more eigenvectors, the better," Technical Report 940036, University of California, Los Angeles, Computer Science, Oct. 1994.

[48] Chen, Y.-F. R., Fowler, G. S., Koutsofios, E., and Wallach, R. S., "Ciao: A graphical navigator for software and document repositories," *Proceedings: Second Working Conference on Reverse Engineering: July 14–16, 1995, Toronto, Ontario, Canada*, edited by L. Wills, P. Newcomb, and E. Chickofsky, Working Conference on Reverse Engineering 1995, 2nd, IEEE Computer Society Press, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1995, pp. 66–75.

[49] Chikofsky, E. J. and Cross, II, J. H., "Reverse engineering and design recovery: A taxonomy," *Software Reengineering*, edited by R. S. Arnold, IEEE Computer Society Press, 1992, pp. 54–58.

[50] Christian Duncan, M. G. and Kobourov, S., "Balanced aspect ratio trees and their use for drawing very large graphs," *Proc. 6th Int. Symp. Graph Drawing, GD*, edited by

S. H. Whitesides, Vol. 1547 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 13–15 Aug. 1998, pp. 111–124.

[51] Cohen, R. F., Eades, P., Lin, T., and Ruskey, F., "Three-Dimensional graph drawing," *Proc. DIMACS Int. Work. Graph Drawing, GD 94*, edited by R. Tamassia and I. G. Tollis, Vol. 894 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 10–12 Oct. 1994, pp. 1–11.

[52] Coleman, M. K. and Parker, D. S., "Aesthetics-based graph layout for human consumption," *Software Practice and Experience*, Vol. 26, No. 12, Dec. 1996, pp. 1415–1438.

[53] Collatz, L. and Maas, C., "On early papers on the eigenvalues of graphs," Tech. Rep. Reihe A, Preprint 6, Institut für Angewandte Mathematik der Universität Hamburg, Jun. 1987.

[54] Cruz, I. F. and Twarog, J. P., "3d graph drawing with simulated annealing," *Symposium on Graph Drawing, GD'95*, edited by F. J. Brandenburg, Vol. 1027 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–22 Sep. 1995, pp. 162–165.

[55] Csinger, A., "The psychology of visualization," Technical report, Department of Computer Science, British Columbia, 1992.

[56] Das, R., Mavriplis, D. J., Saltz, J., Gupta, S., and Ponnusamy, R., "The design and implementation of a parallel unstructured euler solver using software primitives," *Proceedings of AIAA 30th Aerospace Sciences Meeting*, 1992, Paper AIAA920562.

[57] Davidson, R. and Harel, D., "Drawing graphs nicely using simulated annealing." Technical Report CS89-13, Weizmann Institute of Science, Faculty of Mathematics and Computer Sciences, 1989.

[58] DeCougny, H. L., Devine, D. D., Flaherty, J. E., Loy, R. M., Özturan, C., and Shephard, M. S., "Load balancing for the parallel adaptive solution of partial differential equations," *Applied Numerical Mathematics*, Vol. 16, 1994, pp. 157–182.

[59] DeRose, T., Kass, M., and Truong, T., "Subdivision surfaces in chracter animation," *SIGGRAPH 98 Conference Proceedings*, edited by M. Cohen, Annual Conference Series, ACM SIGGRAPH, Addison Wesley, Jul. 1998, pp. 85–94, ISBN 0-89791-999-8.

[60] Di Battista, G., Eades, P., Tamassia, R., and Tollis, I. G., *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, Upper Saddle River, New Jersey 07458, U.S.A, Jul. 1999.

[61] Dieckmann, R., Monien, B., and Preis, R., "Load balancing strategies for distributed memory machines," Tech. Rep. tr-rsfb-97-050, SFB 376, Universität-GH Paderborn, Sep. 1997.

[62] Diniz, P., Plimpton, S., Hendrickson, B., and Leland, R., "Parallel algorithms for dynamically partitioning unstructured grids," *Proceedings of the 7th SIAM Conference of Parallel Processing in Scientific Computing*, SIAM, Feb. 1995, pp. 615–620.

[63] Dongen, S. V., "A new cluster algorithm for graphs," *281*, Centrum voor Wiskunde en Informatica (CWI), ISSN 1386-3681, Dec. 31 1998, p. 42, INS (Information Systems (INS)).

[64] Drineas, P., Frieze, A., Kannan, R., Vempala, S., and Vinay, V., "Clustering in large graphs and matrices," *Proceeding of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM, Jan 1999, pp. 291–299.

[65] Duda, R. O. and Hart, P. E., *Pattern Classification and Scene Analysis*, Wiley-Interscience, 1973.

[66] Duncan, C., Goodrich, M., and Kobourov, S., "Balanced aspect ratio trees and their use for drawing very large graphs," *Journal of Graph Algorithms and Applications*, Vol. 4, No. 3, 2000, pp. 19–46.

[67] Duran, B. S. and Odell., P. L., "Cluster analysis: A survey." 1974.

[68] Durand, D. and Kahn, P., "Mapa," *Proc. Ninth ACM Conference on Hypertext and Hypermedia (Hypertext '98)*, 1998, pp. 11–20.

[69] Eades, P., "A heuristic for graph drawing," *Congresses Numerantium*, Vol. 42, 1984, pp. 149–160.

[70] Eades, P., "Metro maps," Pan-Syndey Area Workshop On Visual Information Processing.

[71] Eades, P., Cohen, R. F., and Huang, M. L., "Online animated graph drawing for web navigation," *Proc. 5th Int. Symp. Graph Drawing, GD*, edited by G. Di Battista, Vol. 1353 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1997, pp. 330–335.

[72] Eades, P. and Feng, Q.-W., "Multilevel visualization of clustered graphs," *Proc. 5th Int. Graph Drawing, GD*, edited by S. North, Vol. 1190 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1996, pp. 101–112.

[73] Eades, P., Feng, Q.-W., and Lin, X., "Straight-line drawing algorithms for hierarchical and clustered graphs," *Proc. 5th Int. Graph Drawing, GD*, edited by S. North, Vol. 1190 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1996, pp. 113–128.

[74] Eades, P., Lai, W., Misue, K., and Sugiyama, K., "Preserving the mental map of a diagram," Research Report IIAS-RR-91-16E, Fujitsu Laboratories Ltd., 140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan, Aug. 1991.

[75] Eades, P. and Lin, X., "Spring algorithms and symmetry," *Theoretical Computer Science*, Vol. 240, No. 2, Jun. 2000, pp. 379–405.

[76] Eades, P. and Mutzel, P., "Graph drawing algorithms," *Algorithms and Theory of Computation Handbook*, CRC Press, 1999, pp. 200–280.

[77] Eades, P., Symvonis, A., and Whitesides, S., "Two algorithms for three dimensional orthogonal graph drawing," *Proc. 5th Int. Graph Drawing, GD*, edited by

S. North, Vol. 1190 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1996, pp. 139–154.

[78] Eades, P., Symvonis, A., and Whitesides, S., "Three dimensional orthogonal graph drawing algorithms," *Discrete Applied Mathematics*, Vol. 103, 2000, pp. 55–87.

[79] Edachery, J., Sen, A., and Brandenburg, F. J., "Graph clustering using distance-k cliques," *Proc. 7th Int. Symp. Graph Drawing, GD*, edited by J. Kratochvíl, Vol. 1731 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 15–19 Sep. 1999, pp. 98–106.

[80] Elsner, U., "Graph partitioning: A survey," Tech. Rep. tr-rsfb-97-27, SFB 393, Technische Universität Chemnitz, Dec. 1997.

[81] Eppstein, D., "Fast hierarchical clustering and other applications of dynamic closest pairs," *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, California, 25–27 Jan. 1998, pp. 619–628.

[82] Erhard, G., *Advances in System Analysis Vol. 4, Graphs as Structural Models: The Application of Graphs and Multigraphs in Cluster Analysis*, Vieweg, 1988.

[83] Falkner, J., Rendl, F., and Wolkowicz, H., "A computational study of graph partitioning," Tech. rep., University of Waterloo, Canada, Nov. 1994, optimization area; to appear in Math Progr.

[84] Farhat, C., "A simple and efficient automatic FEM domain decomposer," *Computers and Structures*, Vol. 28, No. 5, 1988, pp. 579–602.

[85] Feijs, L., Krikhaar, R., and van Ommering, R., "A relational approach to support software architecture analysis," *Software - Practice and Experience*, Vol. 28, No. 4, April 1998, pp. 371–400.

[86] Feng, Q., *Algorithms for Drawing Clustered Graphs*, Ph.D. thesis, Department of Computer Science and Software Engineering, University of Newcastle, Apr. 1997.

[87] Fiduccia, C. M. and Mattheyses, R. M., "A linear time heuristic for improving network partitions," *ACM-IEEE 19th Design Automation Conference*, IEEE Press, 1982, pp. 175–181.

[88] Finkel, R. and Bentley, J., "Quad trees: a data structure for retrieval on composite keys," *Acta Informatica*, Vol. 4, No. 1, 1974, pp. 1–9.

[89] Flanders, V., "Web pages that suck.com," http://www.webpagesthatsuck.com.

[90] Flesch, R., *The Art of Readable Writing*, Harper and Row, 1949.

[91] Fortune, S., "Voronoi diagrams and delaunay triangulations," *Computing in Euclidean Geometry*, edited by D.-Z. D. F. Hwang, World Scientfic, 1992, pp. 193–233.

[92] Franck, G. and Ware, C., "Viewing a graph in a virtual reality display is three times as good as a 2d diagram," *IEEE Conference on Visual Languages*, IEEE, St. Louis, Missouri, USA, October 1994, pp. 182–183.

[93] Frick, A., Keskin, C., and Vogelmann, V., "Integration of declarative approaches (system demonstration)," *Proc. 5th Int. Graph Drawing, GD*, edited by S. North, Vol. 1190 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1996, pp. 184–192.

[94] Frick, A., Ludwig, A., and Mehldau, H., "A fast adaptive layout algorithm for undirected graphs," *Proc. DIMACS Int. Work. Graph Drawing, GD 94*, edited by R. Tamassia and I. G. Tollis, Vol. 894 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 10–12 Oct. 1994, pp. 388–403.

[95] Frick, A., Sander, G., and Wang, K., "Simulating graphs as physical systems," *Dr. Dobb's Journal of Software Tools*, Vol. 24, No. 8, Aug. 1999, pp. 58, 60–64.

[96] Fröhlich, M. and Werner, M., "Demonstration of the interactive graph-visualization system davinci," *Proc. DIMACS Int. Work. Graph Drawing, GD 94*, edited by

R. Tamassia and I. G. Tollis, Vol. 894 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 10–12 Oct. 1994, pp. 266–269.

[97] Fruchterman, T. M. J. and Reingold, E. M., "Graph drawing by force-directed placement," *Software-Practice and Experience*, Vol. 21, No. 11, November 1991, pp. 1129–1164.

[98] Gajer, P., Goodrich, M. T., and Kobourov, S. G., "A fast multi-dimensional algorithm for drawing large graphs," *Proc. 8th Int. Symp. Graph Drawing, GD*, edited by J. Marks, Vol. 1984 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–23 Sep. 2000, pp. 211–221.

[99] Gajer, P. and Kobourov, S. G., "Grip: Graph drawing with intellident placement," *Proc. 8th Int. Symp. Graph Drawing, GD*, edited by J. Marks, Vol. 1984 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–23 Sep. 2000, pp. 222–228.

[100] Galick, A., Kerhoven, T., and Ravaioli, U., "Iterative solution of the eigenvalue problem for a dielectric waveguide," *IEEE Trans. Micro. Theory Tech.*, Vol. 40, 1992, pp. 699–705.

[101] Gallagher, K. and O'Brien, L., "Reducing visualization complexity using decomposition slices," *Proc. Software Visualization Work.*, Department of Computer Science, Flinders University, Adelaide, Australia, Dec 1997, pp. 113–118.

[102] Gansner, E. and North, S., "Improved force directed layouts," *Proc. 6th Int. Symp. Graph Drawing, GD*, edited by S. H. Whitesides, Vol. 1547 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 13–15 Aug. 1998, pp. 364–373.

[103] Gansner, E. R. and North, S. C., "An open graph visualization system and its applications to software engineering," *Software Practice and Experience*, Vol. 30, No. 11, Sep. 2000, pp. 1203–1233.

[104] Gerlhof, C., Kemper, A., Kilger, C., and Moerkotte, G., "Partition-based clustering in object bases: From theory to practice," Tech. Rep. 92-34, RWTH Aachen, Dec. 1992.

[105] Gerlhof, C. A., Kemper, A., Kilger, C., and Moerkotte, G., "Partition-based clustering in object bases: From theory to practice," *Intl. Conf. on Foundations of Data Organization and Algorithms*, Vol. 730 of *Lecture Notes in Computer Science (LNCS)*, Springer, 1993, pp. 301–316.

[106] Gilbert, J. R., Miller, G. L., and Teng, S.-H., "Geometric mesh partitioning: Implementation and experiments," *SIAM Journal on Scientific Computing*, Vol. 19, No. 6, 1998, pp. 2091–2110.

[107] Girard, J.-F. and Würthner, M., "Evaluating the accessor classification approach to detect abstract data types," *Program Comprehension*, IEEE Computer Society Press, Washington D.C., Jul. 2000, pp. 87–99.

[108] Goebel, M. and Gruenwald, L., "A survey of data mining software tools," *SIGKDD Explorations — Newsletter of the Special Interest Group on Knowledge Discovery & Data Mining*, Vol. 1, No. 1, Jun. 1999, pp. 20–33.

[109] Goldstein, C. I., "A finite element method for solving Helmholtz type equations in waveguides and other unbounded domains," *Mathematics of Computation*, Vol. 39, No. 160, Oct. 1982, pp. 309–324.

[110] Greengard, L. and Rokhlin, V., "A fast algorithm for partcile simulations," *Computers in Physics*, Vol. 73, 1987, pp. 325–348.

[111] Griswold, W. G., Chen, M. I., Bowdidge, R. W., Cabaniss, J. L., Nguyen, V. B., and Morgenthaler, J. D., "Tool support for planning the restructuring of data abstractions in large systems," *IEEE Transactions on Software Engineering*, Vol. 24, No. 7, Jul. 1998, pp. 534–558, Special Section: Symposium on Foundations in Software Engineering (FSE-4).

[112] Grumann, J. and Welch, P., "Graph method for technical documentation and reengineering of dp-applications," *Software Reuse and Reverse Engineering in Practice*, edited by P. Hall, Chapman Hall, 1992, pp. 321–353.

[113] Hadany, R. and Harel, D., "A multi-scale method for drawing graphs nicely," *25th International Workshop on Graph-Theoretic Concepts in Computer Science, WG'99*, Vol. 1665, 2000, pp. 262–277.

[114] Harel, D., "On the aesthetics of diagrams," *MPC: 4th International Conference on Mathematics of Program Construction*, LNCS, Springer-Verlag, 1998, pp. 1–2.

[115] Harel, D. and Koren, Y., "A fast multi-scale method for drawing large graphs," Technical Report MCS99-21, Weizmann Institute of Science, Faculty of Mathematics and Computer Science, Nov. 1999.

[116] Harel, D. and Koren, Y., "A fast multi-scale method for drawing large graphs," *Proc. 8th Int. Symp. Graph Drawing, GD*, edited by J. Marks, Vol. 1984 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–23 Sep. 2000, pp. 183–196.

[117] Hartigan, J., *Clustering algorithms*, Wiley, 1975.

[118] He, W. and Marriott, K., "Constrained graph layout," *Proc. 5th Int. Graph Drawing, GD*, edited by S. North, Vol. 1190 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1996, pp. 217–232.

[119] Hendley, R., Drew, N., Wood, A., and R.Beale, "Narcissus: Visualising information," *Proceedings of the IEEE Symposium on Information Visualization*, 1995, pp. 90–96.

[120] Hendrickson, B., "Graph partitioning and parallel solvers: Has the emperor no clothes?" *Lecture Notes in Computer Science*, Vol. 1457, 1998, pp. 218–232.

[121] Herman, Melançon, G., and Marshall, M. S., "Graph visualization and navigation in information visualization: A survey," *IEEE Transactions on Visualization and Computer Graphics*, edited by H. Hagen, Vol. 6 (1), IEEE Computer Society, 2000, pp. 24–43.

[122] Herman, I., "Latour - a tree visualization system," *Proc. 7th Int. Symp. Graph Drawing, GD*, edited by J. Kratochvíl, Vol. 1731 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 15–19 Sep. 1999, pp. 392–399.

[123] Herman, I., "Skeletal images as visual cues in graph visualization," *Proceedings of the Joint Eurographics - IEEE TCCG Symposium on Visualization*, edited by H. L. E. Gröller and W. Ribarsky, Springer, Vienna, Austria, 1999, pp. 13–22.

[124] Hernquist, L., "Performance characteristics of tree codes," *Astrophysics J. Supp.*, Vol. 64, 1987, pp. 715–734.

[125] Hernquist, L., "Hierarchical n-body methods," *Computational Physics Communications*, Vol. 48, 1988, pp. 107–115.

[126] Hightower, R. R., Ring, L. T., Helfman, J. I., Bederson, B. B., and Hollan, J. D., "Graphical multiscale web histories: A study of padprints," *Proceedings of the Ninth ACM Conference on Hypertext*, Mapping and Visualizing Navigation, 1998, pp. 58–65.

[127] Himsolt, M., "A view to graph drawing algorithms through GraphEd," *Proc. ALCOM Int. Work. Graph Drawing, GD*, edited by G. Di Battista, P. Eades, H. d. Fraysseix, P. Rosenstiehl, and R. Tamassia, Centre D'Analyse et de Mathématique Sociales, Paris Sorbonne, 26–29 Sep. 1993, pp. 117–118.

[128] Himsolt, M., "The graphlet system," *Proc. 5th Int. Graph Drawing, GD*, edited by S. North, Vol. 1190 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1996, pp. 233–240.

[129] Himsolt, M., "Graphlet: design and implementation of a graph editor," *Software Practice and Experience*, Vol. 30, No. 11, Sep. 2000, pp. 1303–1324.

[130] Hitz, M. and Montazeri, B., "Measure coupling and cohesion in object-oriented systems," *Proceedings of International Symposium on Applied Corporate Computing (ISAAC95)*, October 1995, pp. 9–16.

[131] Hockney, R. W. and Eastwood, J. W., *Computer simulations using particles*, Adam Hilger, 1998.

[132] Hoffmeyer, S., editor, *Proceedings of the Computer Graphics Conference 2000*, New York, Jul. 23–28 2000, ACMPress.

[133] Hoshen, J., "A graph theoretical method for the management and synchronization of large software updates," *Software - Practice and Experience*, Vol. 28, No. 8, July 1998, pp. 845–857.

[134] Houle, M. E. and Webber, R., "Approximation algorithms for finding best viewpoints," *Proc. 6th Int. Symp. Graph Drawing, GD*, edited by S. H. Whitesides, Vol. 1547 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 13–15 Aug. 1998, pp. 210–223.

[135] Huang, M., *On-line Animated Visualization of Huge Graphs*, Ph.D. thesis, Department of Computer Science and Software Engineering, University of Newcastle, Callaghan 2308, Australia, 1999.

[136] Huang, M. and Eades, P., "A fully animated interactive system for clustering and navigating huge graphs," *Proc. 6th Int. Symp. Graph Drawing, GD*, edited by S. H. Whitesides, Vol. 1547 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 13–15 Aug. 1998, pp. 374–383.

[137] Huang, M. L., Eades, P., and Cohen, R. F., "Online visualization of huge distributed software," *Proc. Software Visualization Work.*, Department of Computer Science, Flinders University, Adelaide, Australia, Dec 1997, pp. 105–112.

[138] Hutchens, D. H. and Basili, V. R., "System structure analysis: Clustering with data bindings," *IEEE Transactions on Software Engineering*, Vol. 11, No. 8, August 1985, pp. 749–757.

[139] Ignatowicz, J., "Drawing force-directed graphs using optigraph," *Symposium on Graph Drawing, GD'95*, edited by F. J. Brandenburg, Vol. 1027 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–22 Sep. 1995, pp. 333–336.

[140] Jain, A., Murty, M., and Flynn, P., "Data clustering: A reviews," *ACM Computing Surveys*, Vol. 31, No. 3, Sep 1999, pp. 264–321.

[141] Jerding, D. and Rugaber, S., "Using visualization for architectural localization and extraction," *Proceedings of the Fourth Working Conference on Reverse Engineering*, edited by I. Baxter, A. Quilici, and C. Verhoef, IEEE Computer Society Press Los Alamitos California, 1997, pp. 267–284.

[142] Jerding, D. F. and Stasko, J. T., "The information mural: A technique for displaying and navigating large information spaces," *Proceedings of the IEEE Symposium on Information Visualization*, Nov. 1995, pp. 257–271.

[143] Jerding, D. F., Stasko, J. T., and Ball, T., "Visualizing interactions in program executions," *Proceedings of the 19th International Conference on Software Engineering (ICSE '97)*, ACM, NY, May 17–23 1997, pp. 360–371.

[144] Jones, B., "Knowledge nation manifesto, australian labour party," http://www.alp.com.au.

[145] Jones, C., *Vertex and Edges partitions of graphs*, Phd thesis, Pennsylvania State University, 1992.

[146] Kamada, T., *Visualizing Abstract Objects and Relations: A Constraint-Based Approach*, Vol. 5 of *Series in Computer Science*, World Scientific, Singapore, Dec. 1989.

[147] Kamada, T. and Kawai, S., "An algorithm for drawing general undirected graphs," *Information Processing Letters*, Vol. 31, 1991, pp. 7–15.

[148] Kamps, T., Kleinz, J., and Read, J., "Constraint-based spring-model algorithm for graph layout," *Symposium on Graph Drawing, GD'95*, edited by F. J. Brandenburg, Vol. 1027 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–22 Sep. 1995, pp. 349–360.

[149] Karypis, G., Aggarwal, R., Kumar, V., and Shekhar, S., "Multilevel hypergraph partitioning: Applications in VLSI domain," Tech. rep., University of Minnesota, Department of Computer Science, 1997, short version in 34th Design Automation Conference.

[150] Karypis, G. and Kumar, V., "Analysis of multilevel graph partitioning," Tech. Rep. TR 95-037, University of Minnesota, Department of Computer Science, 1995.

[151] Karypis, G. and Kumar, V., *MeTis: Unstrctured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0*, Aug. 1995.

[152] Karypis, G. and Kumar, V., "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, 1998.

[153] Karypis, G. and Kumar, V., "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," *Journal of Parallel and Distributed Computing*, 1998.

[154] Karypis, G., Schloegel, K., and Kumar, V., *ParMeTis: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 2.0*, University of Minnesota, Dept. of Computer Science, Sep. 1999.

[155] Kaufman, A. E., "Volume visualization," *ACM Computing Surveys*, Vol. 28, No. 1, Mar. 1996, pp. 165–167.

[156] Kaufman, L. and Rousseeuw, P. J., *Finding Groups in Data - An Introduction to Cluster Analysis*, Probability and Mathemathematical Statistics, Wiley, 1990.

[157] Kaugars, K., Renfelds, J., and Brazma, A., "A simple algorithm for drawing large graphs on small screens," *Proc. DIMACS Int. Work. Graph Drawing, GD 94*, edited by R. Tamassia and I. G. Tollis, Vol. 894 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 10–12 Oct. 1994, pp. 194–205.

[158] Kernighan, B. W., *Some Graph Partitioning Problems Related to Program Segmentation*, Ph.d. dissertation, Princeton Univ., 1969.

[159] Kernighan, B. W. and Lin, S., "An efficient heuristic procedure for partitioning graphs," *The Bell System Techinical Journal*, Vol. 29, No. 2, 1970, pp. 291–307.

[160] Koike, H., "Fractal views: A fractal-based method for controlling information display," *ACM Transactions on Information Systems*, Vol. 100, No. 100, 1999, pp. 1–19.

[161] Koike, H. and Chu, H.-C., "How does 3-D visualization work in software engineering?: Empirical study of a 3-D version/module visualization system," *Proceedings of the 1998 International Conference on Software Engineering*, IEEE Computer Society Press / ACM Press, 1998, pp. 516–519.

[162] Kosak, C., Marks, J., and Shieber, S., "Automating the layout of network diagrams with specified visual organization," *III Transactions on Systems, Man, and Cybernetics*, Vol. 24, No. 3, 1994, pp. 440–454.

[163] Koschke, R., *Atomic Architectural Component Recovery for Program Understadning and Evolution*, Phd. thesis, Institut für Informatik, Universität Stuttgart, October 1999.

[164] Koutsofios, E., "Editing graphs with *dotty*," Technical report, AT&T Bell Laboratories, Murray Hill, NJ, USA, Jul. 1994.

[165] Krone, M. and Snelting, G., "On the inference of configuration structures from source code," *Proceedings of the 16th International Conference on Software Engineering*, IEEE Computer Society Press, May 1994, pp. 49–58.

[166] Kruyt, N. P., "A conjugate gradient method for the spectral partitioning of graphs," *Parallel Computing*, Vol. 22, No. 11, Jan. 1997, pp. 1493–1502.

[167] Kumar, A. and Fowler, R., "A spring modeling algorithm to position nodes of an undirected graph in three dimensions," Technical report, Department of Computer Science University of Texas - Pan American, Edinburg, Texas, October 1996.

[168] Lamping, J. and Rao, R., "Visualizing large trees using the hyperbolic browser," *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, Vol. 2 of *VIDEOS: Visualization*, 1996, pp. 388–389.

[169] Lanza, M., "Combining metrics and graphs for object oriented reverse engineering," 1999.

[170] Lee, A., Dobkin, D., Sweldens, W., and Schröder, P., "Multiresolution mesh morphing," *Siggraph 1999, Computer Graphics Proceedings*, edited by A. Rockwood, Annual Conference Series, Addison Wesley Longman, Los Angeles, 1999, pp. 343–350.

[171] Lee, A., Moreton, H., and Hoppe, H., "Displaced subdivision surfaces," *Proceedings of the Computer Graphics Conference 2000*, edited by S. Hoffmeyer, ACMPress, New York, Jul. 23–28 2000, pp. 85–94.

[172] Lee, A. W., Sweldens, W., Schröder, P., Cowsar, L., and Dobkin, D., "Maps: Multiresolution adaptive parameterization of surfaces," *SIGGRAPH 98 Conference Proceedings*, edited by M. Cohen, Annual Conference Series, ACM SIGGRAPH, Addison Wesley, Jul. 1998, pp. 95–104, ISBN 0-89791-999-8.

[173] Lesh, N., Marks, J., and Patrignnani, M., "Interactive partitioning," *Proc. 8th Int. Symp. Graph Drawing, GD*, edited by J. Marks, Vol. 1984 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–23 Sep. 2000, pp. 31–36.

[174] Lin, T. and O'Brien, L., "FEPSS: A flexible and extensible program comprehension support system," *Working Conference on Reverse Engineering*, IEEE Computer Society, IEEE Computer Society Press, Hawai, USA, Oct. 1998, pp. 40–49.

[175] Lindley, C. A., "Ray tracing and the POV-Ray toolkit," *Dr. Dobb's Journal of Software Tools*, Vol. 19, No. 7, Jul. 1994, pp. 68, 70, 72, 74, 76, 103.

[176] Linos, P., "A tool for maintaining hybrid c++ programs," *Journal of Software Maintenance*, Vol. 8, December 1996, pp. 389–419.

[177] Lisitsyn, I. A. and Kasyanov, V. N., "Higres visualization system for clustered graphs and graph algorithms," *Proc. 7th Int. Symp. Graph Drawing, GD*, edited by J. Kratochvíl, Vol. 1731 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 15–19 Sep. 1999, pp. 82–89.

[178] Lorr, M., *Cluster Analysis for Social Scientists*, Jossey-Bass Ltd., 1987.

[179] Lynn, B. Y. S., Symvonis, A., and Wood, D., "Refinement of three-dimensional orthogonal graph drawings," *Proc. 8th Int. Symp. Graph Drawing, GD*, edited by J. Marks, Vol. 1984 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–23 Sep. 2000, pp. 308–320.

[180] Lyons, K., "Cluster busting in achored graph drawing," *Proceeding of the 1992 CAS Conference*, 1992, pp. 7–16.

[181] Maden, B., "Tom sawyer software, graph layout toolkit(glt)," http://www.tomsawyer.com.

[182] Mancoridis, S., Mitchell, B. S., Chen, Y., and Gansner, E. R., "Bunch: A clustering tool for the recovery and maintenance of software system structures," *IEEE Proceedings of the1998 International Conference on Software Maintenance (ICSM'98)*, 1998, pp. 50–59.

[183] Mancoridis, S., Mitchell, B. S., Rorres, C., Chen, Y., and Gansner, E. R., "Using automatic clustering to produce high-level system organizations of source code," *IEEE Proceedings of the 1998 Int. Workshop on Program Understanding (IWPC'98)*, 1998, pp. 77–88.

[184] Manning, J., Atallah, M., Cudjoe, K., Lozito, J., and Pacheco, R., "A system for drawing graphs with geometric symmetry," *Proc. DIMACS Int. Work. Graph Drawing, GD 94*, edited by R. Tamassia and I. G. Tollis, Vol. 894 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 10–12 Oct. 1994, pp. 262–265.

[185] Mehlhorn, K., *Muti-dimensional Searching and Computational Geometry*, Springer-Verlag, Berlin, Germany, 1st ed., 1984, English Translation of Effiziente Allgorithmen(1977).

[186] Mehlhorn, K. and Näher, S., "LEDA: A platform for combinatorial and geometric computing," *Communications of the ACM*, Vol. 38, No. 1, Jan. 1995, pp. 96–102.

[187] Mehlhorn, K. and Näher, S., *LEDA, a Platform for Combinatorial and Geometric Computing*, Cambridge University Press, 1999.

[188] Mendelzon, A. and Sametinger, J., "Reverse engineering by visualizing and querying," *Software−Concepts and Tools*, Vol. 16, No. 4, 1995, pp. 170–182.

[189] Mendonca, X., "A layout system for information system diagrams," Technical Report 94-01, Department of Computer Science, University of Newcastle, Callaghan 2308, Australia, 1994.

[190] Merris, R., "Laplacians matrices of graphs: A survey," *Linear Algebra and its Applications*, Vol. 197,198, 1994, pp. 143–176.

[191] Merris, R., "A survey of graph Laplacians," *Linear and Multilinear Algebra*, Vol. 39, 1995, pp. 19–31.

[192] Mirkin, B., *Mathematical Classification and Clustering*, Vol. 11 of *Nonconvex Optimization and Its Applications*, Kluwer Academic Publishers, June 1996.

[193] Misue, K., Eades, P., Lai, W., and Sugiyama, K., "Layout adjustment and the mental map," *J. Visual Languages and Computing*, Vol. 6, Jun. 1995, pp. 183–210.

[194] Misue, K., Nitta, K., Sugiyama, K., Koshiba, T., and Inder, R., "Enhancing D-ABDUCTOR towards a diagrammatic user interface platform," *Proc. 2nd Int. Conf. Knowledge-Based Intelligent Electronic Systems*, Apr. 1998, pp. 359–368.

[195] Misue, K. and Sugiyama, K., "Support of human thinking processes with D-ABDUCTOR," Research Report ISIS-RR-94-1E, Fujitsu Laboratories Ltd., 140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan, Jan. 1994.

[196] Mohar, B., "Laplace eigenvalues of graphs – a survey," *Discrete Mathematics*, Vol. 109, 1992, pp. 171–183.

[197] Monien, B., Ramme, F., and Salmen, H., "A parallel simulated annealing algorithm for generating 3d layouts of undirected graphs," *Symposium on Graph Drawing, GD'95*, edited by F. J. Brandenburg, Vol. 1027 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–22 Sep. 1995, pp. 396–408.

[198] Mukherjea, S. and Stasko, J. T., "Applying algorithm animation techniques for program tracing, debugging, and understanding," *Proceedings of the 15th International Conference on Software Engineering*, IEEE Computer Society Press, Apr. 1993, pp. 456–467.

[199] Müller, H. A., Jahnke, J. H., Smith, D. B., Storey, M.-A. D., Tilley, S. R., and Wong, K., "Reverse engineering," *Proceedings of the 22th International Conference on Software Engineering (ICSE-00)*, ACM Press, NY, Jun. 4–11 2000, pp. 47–60.

[200] Munzner, T., "Drawing large graphs with H3Viewer and site manager," *Proc. 6th Int. Symp. Graph Drawing, GD*, edited by S. H. Whitesides, Vol. 1547 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 13–15 Aug. 1998, pp. 384–393.

[201] Munzner, T., "Exploring large graphs in 3D hyperbolic space," *IEEE Computer Graphics and Applications*, Vol. 18, No. 4, Jul./Aug. 1998, pp. 18–23.

[202] Nelson, M. L., "A survey of reverse engineering and program comprehension," .

[203] Neves, F. D., "The aleph: A tool to spatially represent user knowledge about the WWW docuverse," *Proceedings of the Eighth ACM Conference on Hypertext*, Visualization, 1997, pp. 197–207.

[204] Newbery, F. J., "Edge concentration: A method for clustering directed graphs," *In Proceedings of the 2nd International Workshop on Software Configuration Management*, 1989, pp. 76–85.

[205] Nielson, G. M., Kaufman, A., and Post, F., "Data visualization," *Computer Graphics Forum*, Vol. 12, No. 3, 1993, pp. C509–C523.

[206] Noguchi, T. and Tanaka, J., "New automatic layout method based on magnetic spring model for object diagrams of omt," *Proceedings of International Symposium on Future Software Technology 1998 (ISFST'98)*, October 1998, pp. 33–47.

[207] Noik, E. G., "Exploring large hyperdocuments: Fisheye views of nested networks," *Proceedings of the 5th Conference on Hypertext*, ACM Press, New York, NY, USA, Nov. 1993, pp. 192–205.

[208] Noik, E. G., "Layout-independent fisheye views of nested graphs," *Proc. IEEE Symp. Visual Languages, VL*, edited by E. P. Glinert and K. A. Olsen, IEEE Press, 24–27 Aug. 1993, pp. 336–341.

[209] Noik, E. G., "Encoding presentation emphasis algorithms for graphs," *Proc. DI-MACS Int. Work. Graph Drawing, GD 94*, edited by R. Tamassia and I. G. Tollis, Vol. 894 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 10–12 Oct. 1994, pp. 428–435.

[210] North, S. C., "Drawing ranked digraphs with recursive clusters," *Graph Drawing '93, ALCOM International Workshop PARIS 1993 on Graph Drawing and Topological Graph Algorithms*, 1993, p. 75.

[211] North, S. C., "Visualizing graph models of software," *Software Visualization: Programming as a Multimedia Experience*, edited by J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, chap. 5, M.I.T. Press, Feb. 1998, pp. 63–72.

[212] Nour-Omid, B., Raefsky, A., and Lyzenga, G., "Solving finite element equations on concurrent computers," *Parallel Computations and their impact on Mechanics*, edited by A. Noor, American Society Mechanical Engineers, New York, 1986, pp. 209–207.

[213] Oca, C. M. D. and Carver, D. L., "A visual representation model for software subsystem decomposition," *Working Conference on Reverse Engineering*, IEEE Computer Society, IEEE Computer Society Press, Hawai, USA, Oct. 1998, pp. 231–240.

[214] Ostry, D. I., *Some Three-Dimensional Graph Drawing Algorithms*, Master's thesis, Department of Computer Science and Software Engineering, University of Newcastle, Callaghan 2308, Australia, October 1996.

[215] Papakostas and Tollis, "Algorithms for area-efficient orthogonal drawings," *CGTA: Computational Geometry: Theory and Applications*, Vol. 9, 1998.

[216] Papakostas, A. and Tollis, I. G., "Improved algorithms and bounds for orthogonal drawings," *Proc. DIMACS Int. Work. Graph Drawing, GD 94*, edited by R. Tamassia and I. G. Tollis, Vol. 894 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 10–12 Oct. 1994, pp. 40–51.

[217] Papakostas, A. and Tollis, I. G., "Issues in interactive orthogonal graph drawing," *Symposium on Graph Drawing, GD'95*, edited by F. J. Brandenburg, Vol. 1027 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–22 Sep. 1995, pp. 419–430.

[218] Papakostas, A. and Tollis, I. G., "Incremental orthogonal graph drawing in three dimensions," *Proc. 5th Int. Symp. Graph Drawing, GD*, edited by G. Di Battista, Vol. 1353 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1997, pp. 52–63.

[219] Parker, G., Franck, G., and Ware, C., "Visualization of large nested graphs in 3d," *Journal of Visual Languages and Computing*, Vol. 9, Sep. 1998, pp. 299–317.

[220] Parker, J. L., "Information retrieval with large-scale geographic data bases," *ACM SIGFIDET , Codd(ed)*, 1971.

[221] Penteado, R., Masiero, P. C., and do Prado, A. F., "Reengineering of legacy systems based on transformation using the object-oriented paradigm," *Working Conference*

on Reverse Engineering, IEEE Computer Society, IEEE Computer Society Press, Hawai, USA, Oct. 1998, pp. 144–153.

[222] Pfalzner, S. and Gibbon, P., *Many-Body Tree Methods in Physics*, Cambridge University Press, 1996.

[223] Pothen, A., "Graph partitioning algorithms with applications to scientific computing," *Parallel Numerical Algorithms*, edited by D. E. Keyes, A. H. Sameh, and V. Venkatakrishnan, Kluwer Academic Press, 1995, p. 46 pages.

[224] Pothen, A., Simon, H., and Liou, K.-P., "Partitioning sparse matrices with eigenvectors of graphs," *SIAM Journal on Matrix Analysis and Applications*, Vol. 11, No. 3, Jul. 1990, pp. 430–452.

[225] Powers, D. L., "Graph partitioning by eigenvectors," *Linear Algebra and its Applications*, Vol. 101, 1988, pp. 121–133.

[226] Preparata, F. P. and Shamos, M. I., *Computational Geometry: An Introduction*, Springer-Verlag, Berlin, Germany, 1985.

[227] Price, B., Baecker, R., and Small, I., "A principled taxonomy of software visualization," *Journal of Visual Languages and Computing*, Vol. 4, No. 3, 1993, pp. 211–266.

[228] Pulo, K., "Recursive space decompositions in force-directed graph drawing algorithms," *InVis.au, Australian Symposium on Information Visualization*, edited by P. Eades, University of Sydney, Australia, 3–4 Dec. 2001, p. (to appear).

[229] Purchase, H., "Which aesthetic has the greatest effect on human understanding?" *Proc. 5th Int. Symp. Graph Drawing, GD*, edited by G. Di Battista, Vol. 1353 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1997, pp. 248–261.

[230] Purchase, H. C., Allder, J.-A., and Carrington, D., "User preference of graph layout aesthetics: a UML study," *Proc. 8th Int. Symp. Graph Drawing, GD*, edited by

J. Marks, Vol. 1984 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–23 Sep. 2000, pp. 5–18.

[231] Purchase, H. C., Cohen, R. F., and James, M., "Validating graph drawing aesthetics," *Symposium on Graph Drawing, GD'95*, edited by F. J. Brandenburg, Vol. 1027 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–22 Sep. 1995, pp. 435–446.

[232] Purchase, H. C. and Leonard, D., "Graph drawing aesthetic metrics," Technical Report 361, Department of Computer Science, University of Queensland, 4072, Australia, 2 May 1996.

[233] Quigley, A., "Automated tool support for a large scale diagramming tool," *2nd Australian Workshop on Constructing Software Engineering Tools, (AWCSET'99)*, edited by J. Grundy, Macquarie University Sydney, Australia, Oct. 1999, pp. 55–58.

[234] Quigley, A., "Graphical liquid miro," Dagsthul Software Visualization Seminar.

[235] Quigley, A. and Eades, P., "Visualizaing a reverse engineered system structure with dynamic 3-d clustered graph drawings," *Proc. Software Visualization Work.*, Department of Computer Science, Flinders University, Sydney, Australia, Dec 1997, pp. 25–29.

[236] Quigley, A. and Eades, P., "Proveda: A scheme for progressive visualization and exploratory data analysis of clusters," *Proc. Software Visualization Work.*, Department of Computer Science, UTS, Sydney, Australia, Dec 1999, pp. 67–74.

[237] Quigley, A. and Eades, P., "FADE : Graph drawing, clustering, and visual abstraction," *Proc. 8th Int. Symp. Graph Drawing, GD*, edited by J. Marks, Vol. 1984 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–23 Sep. 2000, pp. 197–210.

[238] Quigley, A., Postema, M., and Schmidt, H., "Revis: Reverse engineering by clustering and visual object classification," *Proceedings of the 2000 Australian Software*

Engineering Conference, edited by D. D. Grant, IEEE Press, Los Alamitos, California, April 2000, pp. 119–125.

[239] Quigley, A. J., "Large scale 3d clustering and abstraction," *Proceedings of the 2000 Pan-Syndey Area Workshop On Visual Information Processing*, edited by J. Jin and P. Eades, University of Sydney, Australia, Dec. 2000, pp. 101–103.

[240] Quinn, N. and Breuer, M. A., "A forced directed component placement procedure for printed circuit boards," *IEEE Transactions on Circuits and Systems*, Vol. CAS-26, No. 6, 1979, pp. 377–388.

[241] Reingold, E. M. and Tilford, J. S., "Tidier drawing of trees," *IEEE Trans. Software Engineering*, Vol. SE-7, No. 2, Mar. 1981, pp. 223–228.

[242] Reiss, S. P., "Software visualization in the desert environment," *ACM SIGPLAN Notices*, Vol. 33, No. 7, Jul. 1998, pp. 59–66.

[243] Reiss, S. P., "Software visualization in the Desert environment," *Proceedings of the ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, 1998, pp. 59–66.

[244] Roxborough, T. and Sen, A., "Graph clustering using multiway ratio cut," *Proc. 5th Int. Symp. Graph Drawing, GD*, edited by G. Di Battista, Vol. 1353 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1997, pp. 291–296.

[245] Rubin, H. A., "Metric systems – software economics 101," *CIO*, 1997.

[246] Rugaber, S., "A tool suite for evolving legacy software," *Proceedings; IEEE International Conference on Software Maintenance*, IEEE Computer Society Press, 1999, pp. 33–39.

[247] Rugaber, S. and White, J., "Restoring A legacy: Lessons learned," *IEEE Software*, Vol. 15, No. 4, Jul./Aug. 1998, pp. 28–33.

[248] Ryall, K., Biedl, T., and Whitesides, S., "Graph multidrawing: Finding nice drawings without defining nice," *Proc. 6th Int. Symp. Graph Drawing, GD*, edited by

S. H. Whitesides, Vol. 1547 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 13–15 Aug. 1998, pp. 347–355.

[249] Sablowski, R. and Frick, A., "Automatic graph clustering (system demonstration)," *Proc. 5th Int. Graph Drawing, GD*, edited by S. North, Vol. 1190 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1996, pp. 395–400.

[250] Sajeev, A. S. M., Wang, W., Quigley, A., and Eades, P., "Towards visualizing size measures of large systems," *Proc. Australian Conference on Software Metrics*, Nov 2000, pp. 87–95.

[251] Salmon, J. K. and Warren, M. S., "Skeletons from the treecode closet," *Journal of Computational Physics*, Vol. 111, No. 1, 1994, pp. 136–155.

[252] Samet, H., "The quadtree and related hierarchical data structures," *acms*, Vol. 16, No. 2, June 1984, pp. 187–260.

[253] Samet, H., *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, Addison-Wesley, Reading, MA, 1990.

[254] Samet, H., *The design and analysis of spatial data structures*, Addison-Wesley, 1990.

[255] Sander, G., "Graph layout through the VCG tool," *Proc. DIMACS Int. Work. Graph Drawing, GD 94*, edited by R. Tamassia and I. G. Tollis, Vol. 894 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 10–12 Oct. 1994, pp. 194–205.

[256] Sarkar, M. and Brown, M., "Graphical fisheye views of graphs," *ACM SIGCHI '92 Conference on Human Factors in Computing Systems*, March 1992, pp. 83–84.

[257] Schaffer, D., Zuo, Z., Greenberg, S., Bartram, L., Dill, J., Dubs, S., and Roseman, M., "Navigating hierarchically clustered networks through fisheye and full-zoom methods," *ACM Trans. Computer-Human Interaction*, Vol. 3, No. 2, Jun. 1996, pp. 162–188.

[258] Schultz, B., *Bounded waveguide Eigenmodes, finite element method solution*, Master's thesis, Department of Electrical Engineering, University of Kentucky, 1994.

[259] Spath, H., *Cluster Analysis Algorithms for data reduction and classification of objects*, Horwood, 1980.

[260] Stasko, J., Domingue, J., Brown, M. H., and Price, B. A., editors, *Software Visualization: Programming as a Multimedia Experience*, M.I.T. Press, Feb. 1998.

[261] Storer, J. A., "The node cost measure for embedding graphs on the planar grid (extended abstract)," *Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, Los Angeles, California, 28–30 Apr. 1980, pp. 201–210.

[262] Storey, M.-A. D., Mueller, H., and Wong, K., "Manipulating and documenting software structures," *Software Visualization*, edited by P. Eades and K. Zhang, Software Engineering and Knowledge Engineering, World Scientific, Feb. 1996, pp. 244–263.

[263] Storey, M. A. D. and Müller, H., "Manipulating and documenting software structures using shrimp views," *International Conference in Software Maintenance*, IEEE Computer Society Press, 1995, pp. 275–285.

[264] Storey, M.-A. D. and Müller, H. A., "Graph layout adjustment strategies," *Symposium on Graph Drawing, GD'95*, edited by F. J. Brandenburg, Vol. 1027 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–22 Sep. 1995, pp. 487–499.

[265] Storey, M.-A. D., Wong, K., and Müller, H. A., "Rigi: A visualization environment for reverse engineering," *Proceedings of the 1997 International Conference on Software Engineering*, ACM Press, 1997, pp. 606–607.

[266] Strothotte, T., *Computational Visualization. Graphics, Abstraction, and Interactivity*, Springer-Verlag, Heidelberg, 1998.

[267] Suaris, P. R. and Kedem, G., "An algorithm for quadrisection and its application to standard cell placement," *IEEE Transactions on Circuits and Systems*, Vol. 35, No. 3, Mar. 1988, pp. 294–303.

[268] Sugiyama, K. and Misue, K., "Multi-viewpoint perspective display methods: Formulation and application to compound graphs," *Proceedings of the 4th International Conference on HCI*, Elsevier, 1991, pp. 834–838.

[269] Sugiyama, K. and Misue, K., "A simple and unified method for drawing graphs: Magnetic-spring algorithm," *Proc. DIMACS Int. Work. Graph Drawing, GD 94*, edited by R. Tamassia and I. G. Tollis, Vol. 894 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 10–12 Oct. 1994, pp. 364–375.

[270] Sugiyama, K. and Misue, K., "Graph drawing by the magnetic spring model," *J. Visual Languages and Computing*, Vol. 6, No. 3, 1995, pp. 217–232.

[271] Sugiyama, K., Tagawa, S., and Toda, M., "Methods for visual understanding of hierarchical system structures," *IEEE Transactions on Systems, Man, & Cybernetics*, Vol. 11, No. 2, Feb. 1981, pp. 109–125.

[272] Tamassia, R., "Graph drawing," *CRC Handbook of Discrete and Computational Geometry*, edited by J. E. Goodman and J. O'Rourke, chap. 44, CRC Press, 1997, pp. 202–250.

[273] Tamassia, R., Battista, G. D., and Batini, C., "Automatic graph drawing and readability of diagrams," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 18, No. 1, Jan 1988, pp. 61–79.

[274] Teng, S., *Points, Spheres, and Separators: A Unified Geometric Approach to Graph Partitioning*, Ph.D. thesis, School of Computer Science, Carnegir Mellon University, 1991.

[275] Teng, S.-H., "Provably good partitioning and load balancing algorithms for parallel adaptive n-body simulation," *SIAM J. Sci. Comput.*, Vol. 19, No. 2, March 1998, pp. 635–656.

[276] Terveen, L., Hill, W., and Amento, B., "Constructing, organizing, and visualizing collections of topically related Web resources," *ACM Transactions on Computer-Human Interaction*, Vol. 6, No. 1, Mar. 1999, pp. 67–94.

[277] Tufte, E. R., *Envisioning Information*, Graphics Press, Box 430, Cheshire, CT 06410, USA, 1990.

[278] Tufte, E. R., *Visual Explanations: Images and Quantities, Evidence and Narrative*, Graphics Press, Box 430, Cheshire, CT 06410, USA, 1997.

[279] Tunkelang, D., "A practical approach to drawing undirected graphs," Tech. rep., Carnegie Mellon University, 1994.

[280] Tunkelang, D., "JIGGLE: Java interactive general graph layout environment," *Proceedings of the 6th International Symposium on Graph Drawing*, edited by S. Whitesides, Vol. 1547 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1998, pp. 413–422.

[281] Tzerpos, V. and Holt, R. C., "MoJo: A distance metric for software clusterings," *In Proc. Working Conf. on Reverse Engineering*, 1999, pp. 187–195.

[282] Voisard, A. and Juergens, M., "Geospatial information extraction: Querying or quarrying?" Tech. Rep. TR-98-010, Berkeley, CA, 1998.

[283] Walshaw, C., "A multilevel algorithm for force-directed graph drawing," *Proc. 8th Int. Symp. Graph Drawing, GD*, edited by J. Marks, Vol. 1984 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–23 Sep. 2000, pp. 171–182.

[284] Walshaw, C., "A multilevel algorithm for force-directed graph drawing," Tech. Rep. Mathematics Research Report 00/IM/60, School of Computing and Mathematical Sciences, University of Greenwich, Apr 2000.

[285] Walshaw, C. and Berzins, M., "Dynamic load balancing for PDE solvers on adaptive unstructured meshes," Research Report Series 92.32, University of Leeds School of Computer Studies, Dec. 1992.

[286] Wang, W., Yang, J., and Muntz, R., "Sting: A statistical information grid approach to spatial data mining," *23rd International Conference on Very Large Data Bases(VLDB)*, IEEE, 1997, pp. 186–195.

[287] Wang, W., Yang, J., and Muntz, R., "Sting+: An approach to active spatial data mining," *Proceedings of the 15th International Conference on Data Engineering*, IEEE, 1999, pp. 116–125.

[288] Wang, X. and Miyamoto, I., "Generating customized layouts," *Symposium on Graph Drawing, GD'95*, edited by F. J. Brandenburg, Vol. 1027 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 20–22 Sep. 1995, pp. 504–515.

[289] Ware, C., *Information Visualization: Perception for Design*, The Morgan Kaufmann Series in Interactive Technologies, December 1999.

[290] Warren, M. S. and Salmon, J. K., "Astrophysical N-body simulations using hierarchical tree data structures," *Proceedings of the Conference on Supercomputing*, IEEE Computer Society Press, Los Alamitos, CA, USA, Nov. 1992, pp. 570–577.

[291] Wazwaz, A.-M., "The decomposition method applied to systems of partial differential equations and to the reaction-diffusion Brusselator model," *Applied Mathematics and Computation*, Vol. 110, No. 2–3, Apr. 2000, pp. 251–264.

[292] Webber, R. J., *Finding the Best Viewpoints for Three-Dimensional Graph Drawings*, Ph.D. thesis, Department of Computer Science and Software Engineering, The University of Newcastle, Newcastle, Australia, Jul. 1998.

[293] West, T. G., *In the Mind's Eye : Visual Thinkers, Gifted People With Dyslexia and Other Learning Difficulties, Computer Images and the Ironies of Creativity*, Prometheus Books, September 1997.

[294] Wiggerts, T. A., "Using clustering algorithms in legacy systems remodularization," *Proceedings: 4th Working Conference on Reverse Engineering*, edited by I. D. Baxter, A. Quilici, and C. Verhoef, IEEE Computer Society Press, 1997, pp. 33–43.

[295] Wills, G. J., "NicheWorks — interactive visualization of very large graphs," *Proc. 5th Int. Symp. Graph Drawing, GD*, edited by G. Di Battista, Vol. 1353 of *Lecture Notes in Computer Science, LNCS*, Springer-Verlag, 18–20 Sep. 1997, pp. 403–414.

[296] Wills, G. J., "NicheWorks — interactive visualization of very large graphs," *Journal of Computational and Graphical Statistics*, Vol. 8, No. 1, June 1999, pp. 190–202.

[297] Wood, A., Drew, N., Beale, R., and Hendley, R., "Hyperspace: Web browsing with visualization," *In the Third International WorldWide Web Conference Poster Proceedings*, April 1995, pp. 21–25.

[298] Woodman, E., "Information generation," *EMC news release*, 2000.

[299] Woodwark, J. R., "The explicit quad tree as a structure for computer graphics," *Comput. J.*, Vol. 25, May 1982, pp. 235–238.

[300] Yang, C., Sorensen, D. C., Meiron, D. I., and Wedeman, B., "Numerical computation of the linear stability of the diffusion model for crystal growth simulation," Technical Report TR96-04, Department of Comp. App. Mathematics, Rice University, Houston, TX, 1996.

[301] Young, P., "Program comprehension," Technical report, Centre for Software Maintenance University of Durham, Durham, UK, 1996.

[302] Young, P., "Three dimensional information visualization," Technical report, Centre for Software Maintenance University of Durham, Durham, UK, 1996.

[303] Zhang, T., Ramakrishnan, R., and Livny, M., "Birch: An efficient data clustering method for very large databases," Technical report, Computer Sciences Department, University of Wisconsin-Madison, 1995.

[304] Zupan., J., *Clustering of Large Data Sets*, Chemometrics Research Studies, Research Studies Press, 1982.

Visualizations, Animations, and Models on CD-ROM

It is difficult to accurately show a three dimensional graph drawing on paper since, as it has been noted, rotating a three dimensional graph layout is akin to turning and manipulating an unfamiliar object in ones hands [214, 92, 292] to learn more about it. The CD-ROM accompanying this thesis contains several directories with images of two dimensional drawings, images of three dimensional drawings, video of two dimensional drawings during the progressive cycle, video of three dimensional drawings, and three dimensional models.

In terms of visualization, animation, and models the **directories** on the CD-ROM include:

Case Study I: Software Visualization

⬦ **CASE1/GALLERY** ...... Entire Case Study I Picture Gallery

⬦ **CASE1/2D-VIS** .... Extra FADE2D Software Visualizations

⬦ **CASE1/3D-VIS** .... Extra FADE3D Software Visualizations

⬦ **CASE1/2D-VIDEO** ..... Video of FADE2D Software Visualizations

⬦ **CASE1/3D-VIDEO** ..... Video of FADE3D Software Visualizations

⬦ **CASE1/3D-MODELS** .... Software Visualization, VRML models

◇ **CASE1/DATA** .... Raw View Data

◇ **CASE1/MISC** .... Software Visualization Miscellaneous

Case Study II: Matrix Market Visualizations

◇ **CASE2/GALLERY** ...... Entire Case Study II Picture Gallery

◇ **CASE2/2D-VIS** .... FADE2D Matrix Market Visualizations

◇ **CASE2/3D-VIS** .... FADE3D Matrix Market Visualizations

◇ **CASE2/2D-VIDEO** ..... Video of FADE2D Matrix Market drawings

◇ **CASE2/3D-VIDEO** ..... Video of FADE3D Matrix Market drawings

◇ **CASE2/3D-MODELS** .... Matrix Market, VRML models

◇ **CASE2/DATA** .... Raw Matrix Data

◇ **CASE2/MISC** .... Matrix Market Miscellaneous

Thesis Images & Video

◇ **THESIS/IMAGES** ...... Images Used in Thesis

◇ **THESIS/VIDEO** .... Video Based on Images in Thesis

General Visualizations

◇ **GENERAL/IMAGES** ..... General non-case study drawings

◇ **GENERAL/VIDEO** ...... General Video Clips

◇ **GENERAL/MODELS** ..... General Data Models

◇ **GENERAL/DATA** .... General Raw Data

Results of Measures on CD-ROM

The CD-ROM accompanying thesis contains several directories with all the results from the two case studies.

In terms of results, the **`directories`** on the CD-ROM include:

Case Study I: Software Visualization

✓ **`CASE1/RESULTS/VPAR`** .... Case Study I Visual Précis Aesthetic Results

✓ **`CASE1/RESULTS/PERF`** .... Case Study I Performance Versus Error Results

✓ **`CASE1/RESULTS/HCQM`** .... Case Study I $\mathcal{HCG}$ Quality Measure Results

Case Study II: Matrix Market Visualizations

✓ **`CASE2/RESULTS/VPAR`** .... Case Study II Visual Précis Aesthetic Results

✓ **`CASE2/RESULTS/PERF`** .... Case Study II Performance Versus Error Results

✓ **`CASE2/RESULTS/HCQM`** .... Case Study II $\mathcal{HCG}$ Quality Measure Results