Seeing the Forest for the Trees: Hierarchical Display of Hypertext Structure

Steven Feiner

Department of Computer Science
Columbia University
New York, NY 10027
feiner@cs.columbia.edu

ABSTRACT

Most recent hypertext systems support hierarchy only as a restricted subset of directed graph structure. Consequently they do not provide many of the capabilities for graphical information hiding and structure manipulation that a tree makes possible. This paper describes display techniques developed for IGD, a hypertext system that supports the creation of large graphical documents whose arbitrary directed graph structure is embedded in a strict hierarchy. IGD offers the full generality of arbitrary keyworded links, while simultaneously allowing hierarchies to be easily manipulated and displayed with much of their structural detail selectively abstracted.

Keywords: hypertext, hypermedia, directed-graph display, graph browser, display decluttering, electronic documents

1. Introduction

Hypertext systems make it possible to create documents that contain direct links to other parts of themselves or to different documents [NELS81]. If both the source and destination of a link are thought of as individual objects, then it is natural to conceive of a hypertext document's structure as a directed graph. The graph's nodes are documents or parts of documents and its arcs are the links that join them. This model is sufficiently appealing that a number of researchers have implemented systems that display a hypertext document's underlying structure as a browseable directed graph [FEIN82; MEYR86; DELI86; HALA87; SMIT87]. Developing effective ways to display large graph structures has, however, proven difficult. Unfortunately, straightforward approaches that are well suited for depicting small, sparsely connected graphs do not scale up. There are two basic problems: graphs may have an arbitrarily large number of nodes, and each node may be connected to any number of other nodes.

The first problem, that of sheer size, is shared with virtually any kind of large information space to be displayed. Hardware display resolution and size constraints and inherent limitations in the human visual system place an upper bound on the number of resolvable, let alone recognizable, objects that can be displayed simultaneously. A number of user interface techniques have been applied to this problem. A scrollable window may be used to inspect a large data structure piecemeal, letting the user pan over it and zoom in and out to examine parts of the data structure in more or less detail. A "world view" map [DONE78] can show the full data structure to be traversed in a separate window. A "you are here" rectangle overlaid on the world view indicates how the enlarged subsection shown in the scrollable window relates to the world view. The world view map can also make possible speedy travel from one part of the data structure to another in the enlarged view by moving the "you are here" icon in the world view.

The second problem, that of arbitrary interconnection structure, is peculiar to graphs. A graph arc is typically represented as a continuous line connecting two nodes. This has several ramifications. Unlike nodes, arcs are typically not recognizable in isolation. An arc's two nodes (or some representation of each) may have to be visible along with the arc itself for the arc's full identity to be grasped. Arcs may also overlap nodes and other arcs, making the display difficult to interpret. Even if only one arc is allowed between each pair of nodes, n nodes may be connected by up to $n(n-1)/2$ arcs, discounting self adjacencies. The large number of arcs that may have to be displayed can impair legibility, even if overlap is not considered a problem. If multiple arcs are allowed between a pair of nodes, for example to signify different types of links, the situation gets worse.

Graph layout algorithms offer a partial solution to the graph display problem insofar as they allow the user to forgo the task of positioning nodes and arcs by hand [SUGI81; ROBI87; ROWE87; TRIC88]. These algorithms use a variety of heuristics to accomplish goals such as minimizing the number of edge crossings and keeping closely connected nodes together. Although problems of aesthetics and run-time may eventually be overcome, a fundamental difficulty remains. Although nodes may always be constrained not to overlap each other, the same is not true of arcs. There are nonplanar graphs, as

small as the undirected graphs of Figure 1, that cannot be embedded in a plane without having an arc cross another arc or node [GROS87]. Thus, the size, shape, and position of the nodes and arcs determine only in part whether intersections will occur.

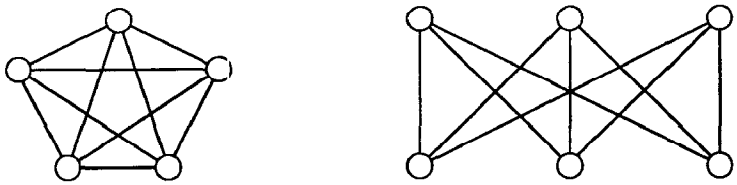


Figure 1: *Two nonplanar graphs.*

Selective display of nodes and arcs can reduce the number of visible objects, independent of the spatial windowing techniques discussed earlier. For example, if nodes or arcs have associated types, then only those satisfying a specified query request on those types may be displayed. If one or more nodes or arcs are to be considered focal points, then "fisheye views" [FURN86] can be composed. These are created by selectively varying the size, position, and graphical style of nodes and arcs, according to a degree of interest function that depends on how close each is to a focal point and on some measure of the object's intrinsic importance. Closer and more important objects are emphasized, while those below some threshold may not be displayed at all. Depending on the rationale behind how objects are selected for display, however, it may be unclear as to whether objects have been suppressed or do not exist. As well, changing focal point in a fisheye view may have an intentionally drastic global effect on the graph's appearance. The use of automated layout techniques with either selective display approach will further differentiate the displays produced, which may make it difficult for the viewer to relate two displays that were created with different focal points or selection criteria.

A number of these issues have been explored in graph browsing systems such as GRAB [ROWE87], ISI Grapher [ROBI87], and BONSAI [HELF87]. One issue that these systems have not addressed is that of combining strict tree hierarchy with arbitrary directed graph structure.

2. Hypertext and Hierarchy

Some of the earliest hypertext systems, such as NLS [ENGE68], HES [CARM69], and FRESS [VAND71], provided an ordered tree hierarchy in the form of nested, optionally numbered, sections, which were maintained separately from the directed graph structure. Their users could selectively display and edit parts of the hierarchy. For example, visibility could be controlled by section level or a section moved elsewhere in the hierarchy as a single unit.

In contrast, many recent hypertext systems that support graph browsing, such as Intermedia [MEYR86] and Neptune [DESL86], provide hierarchy only insofar as it is a subset of general graph structure. Of those that expressly encourage hierarchy, WE [SMIT87] models the document creation process in part as the coercion of an initial directed graph network into the preferred final form: a single tree. Notecards [HALA87] provides hierarchy through "file boxes", but they are implemented and displayed with the same mechanism used for regular links.

One disadvantage of supporting hierarchy through facilities designed for arbitrary directed graph structure is that the powerful information hiding capabilities that a hierarchy can provide are not exploited. Although WE does take advantage of these capabilities, it only offers them for that part of the document whose graph structure has already been transformed into a tree. It is arbitrary graph structure, however, that can often benefit most from detail suppression.

3. Hierarchy in IGD

IGD (Interactive Graphical Documents) [FEIN82; FEIN85; YANK85], also known as the Electronic Document System, was an experimental hypertext environment that supported the creation and display of arbitrary keyworded graph structure, embedded in a tree hierarchy.

The nodes in IGD's directed graph structure were called *pages*, modeled in part after the screen-sized pages of PROMIS [SCHU79], but emphasizing graphics, rather than text. Pages were the leaves in the tree. They contained pictures, text, pickable buttons, executable actions, and indexing keywords, all specified with the system's graphics and text editors. At any time the reader of a document was presented with a single current page. The tree's internal nodes were called *chapters*, and could contain both pages and recursively nested chapters. The highest level of the document was itself a chapter. Thus each chapter could be considered to be a subdocument. Graph structure was provided by a set of links that connected pages. Pages were therefore both the leaves of the tree and the nodes of the directed graph. Unlike the hierarchies of NLS, HES,

and FRESS, which modeled the ordered subsections of a book, IGD's chapters and pages had no implied order. Each chapter's contents formed an unordered set, which was ordered only by the links that stitched together its pages.

Documents were created with a direct manipulation editor that operated on an iconic representation of the document's structure. Chapters and pages were represented as rectangles on the display. Chapters had black borders, while pages had white borders. Links were presented as arrows between pages.

4. Notations for Displaying Graphs and Trees

When designing IGD, we considered the standard graphical notations for displaying trees listed by Knuth [KNUT73]: graph trees, indentation, and nested sets. Several ways to depict the same small graph together with an associated tree hierarchy are sketched in Figures 2-4. Internal tree nodes are labeled for convenience in comparing the notations. Figure 2 uses a graph tree notation for the hierarchy to complement the display of a full directed graph. The nodes and arcs of the graph and the tree are distinguished by their graphical style: solid lines for the directed graph, stippled lines for the tree. If desired, placement of the nodes comprising the hierarchy may be more arbitrary, trading off compactness for clarity. In general, this approach allows arbitrary placement of graph nodes, provided that arc crossings are not considered a problem. It increases the number of nodes and arcs displayed, however, and complicates the appearance of the display if two different styles of arcs and nodes are used.

The indentation approach lays out the graph with an indentation rule that restricts the placement of a tree node's children, as shown in Figure 3. Not only does it eliminate the need for explicit tree arcs, but it provides a clear visual distinction between hierarchy and arbitrary graph structure. The indentation approach shown here strongly suggests a linear ordering of each node's children, however, and therefore may be better suited for representing oriented trees, such as those defined by numbered sections.

Nested set notation, as shown in Figure 4, expresses tree structure through spatial containment. Each node is nested inside its parent node. The nested set paradigm offers visually distinct methods for showing hierarchy and graph structure, while not constraining a node's placement beyond requiring that it be nested in its parent. Thus, the nodes inside another node can be positioned so as to best elucidate the graph structure that connects them. As well, a node and its contents can be easily manipulated as a unit. Note that nested set notation may also be merged into the graph tree and indentation approaches. For example, the graph node leaves of Figure 2 may be nested inside their tree parents or the internal tree node bars of Figure 3 may be expanded downward to enclose their children.

We adopted a variant of nested set notation for use in IGD. Authors created IGD documents using a multiple window graphical editor, as shown in the IGD screens of Figures 5-7. Each window could display the subtree of the hierarchy anchored at a selected chapter or page. The chapter hierarchy provides a convenient means for performing information hiding and display decluttering. IGD used three basic techniques: subtree detail suppression, subtree display selection, and

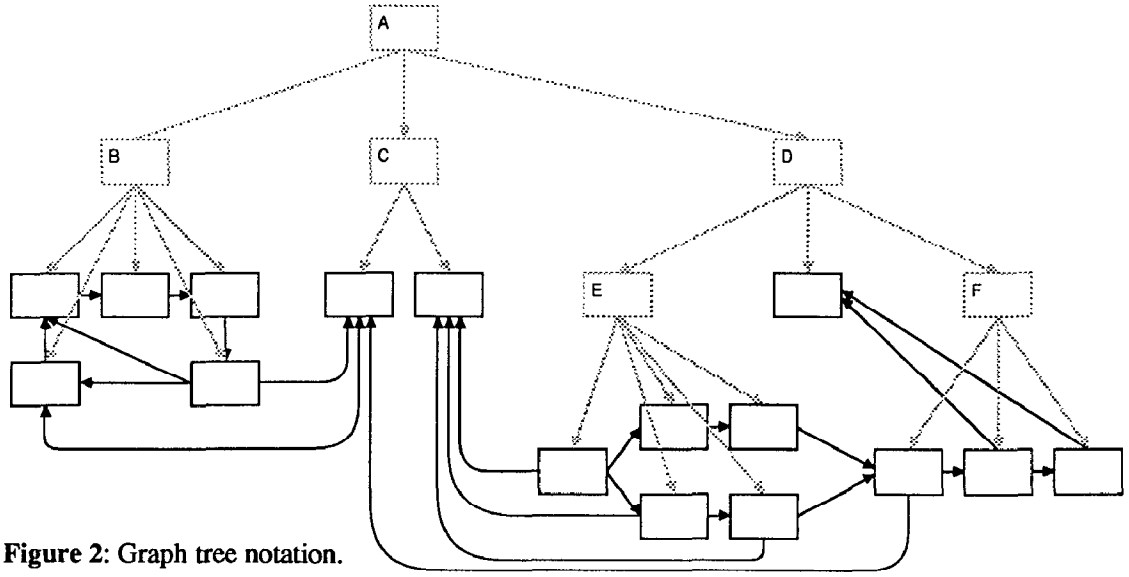


Figure 2: Graph tree notation.

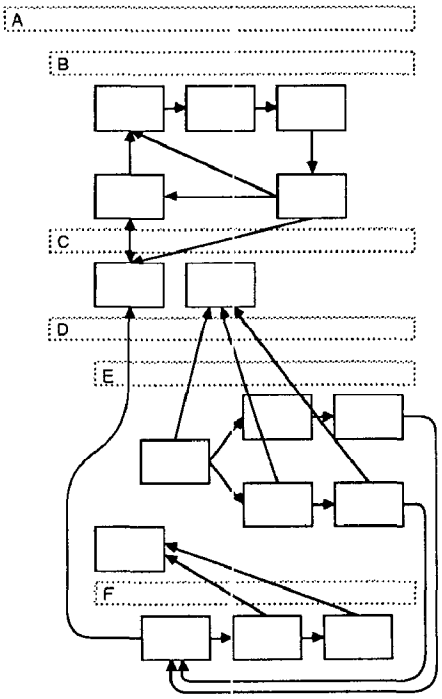
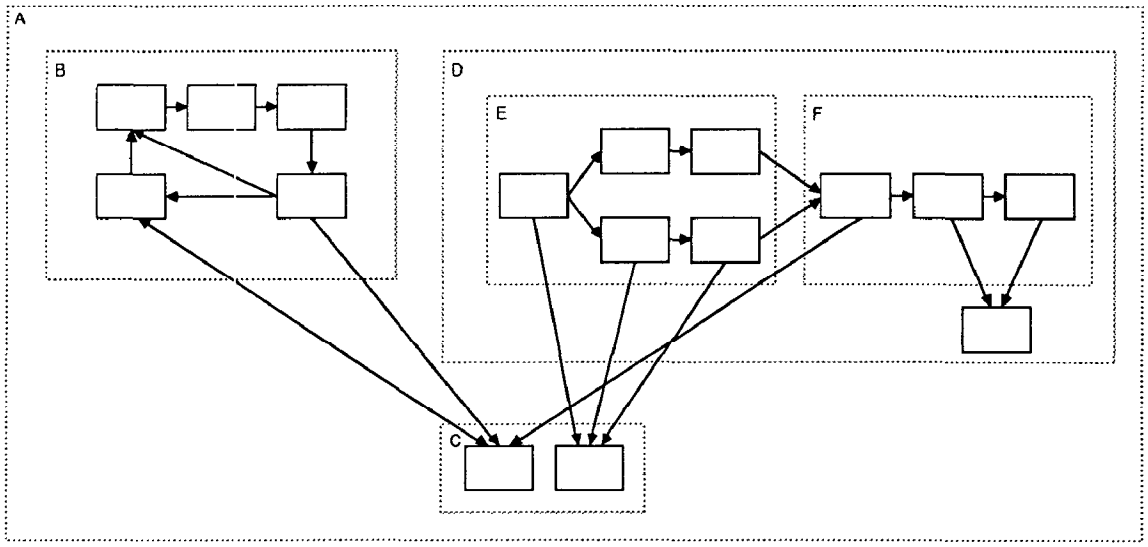


Figure 3: Indentation.

Figure 4: Nested set notation.



link inheritance, all described below. Of these three, link inheritance is the only one that hides nonhierarchical details of the graph structure.

Subtree Detail Suppression

One method of information hiding suppresses the display of any structure inside selected chapters. The large window at the lower right of Figure 5 shows a single chapter, named REFERENCE MANUAL, whose three child chapters chapters are displayed as solid rectangles. Figure 6 shows additional detail revealed in these three chapters at the user's request.

Subtree Display Selection

The user could move up or down the hierarchy in each window independently. Movement down was accomplished by selecting a visible node at any level inside the outermost chapter (Figure 6), which would then fill its window (Figure 7).

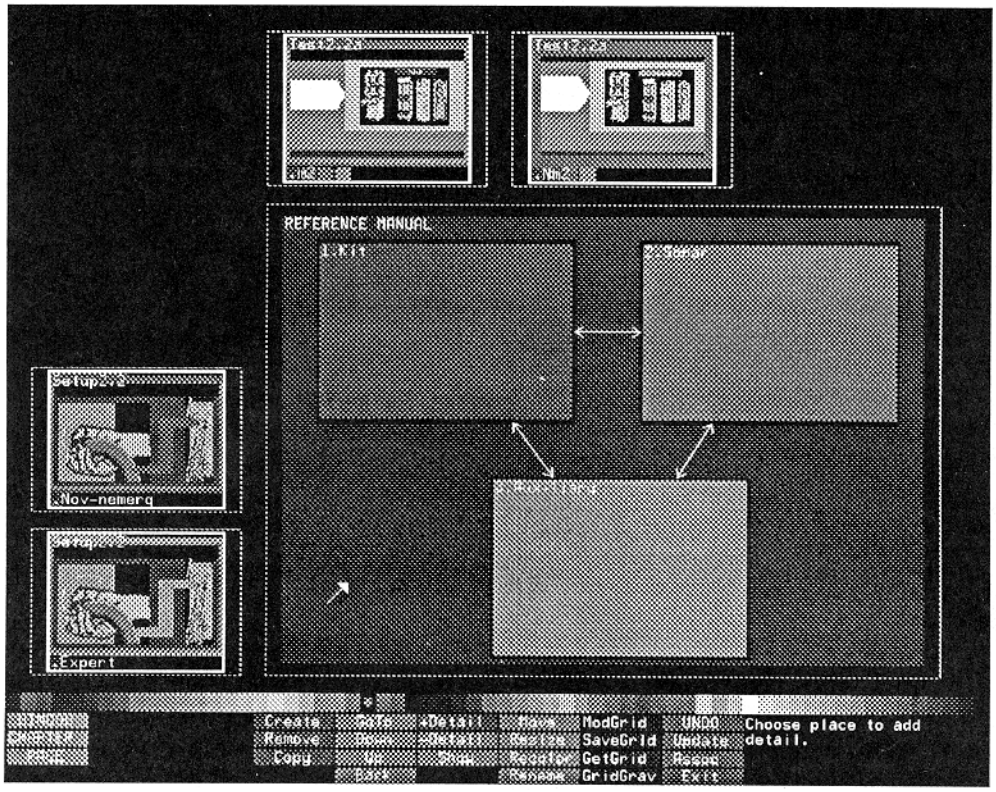


Figure 5: The document layout system. A chapter showing detail suppression.

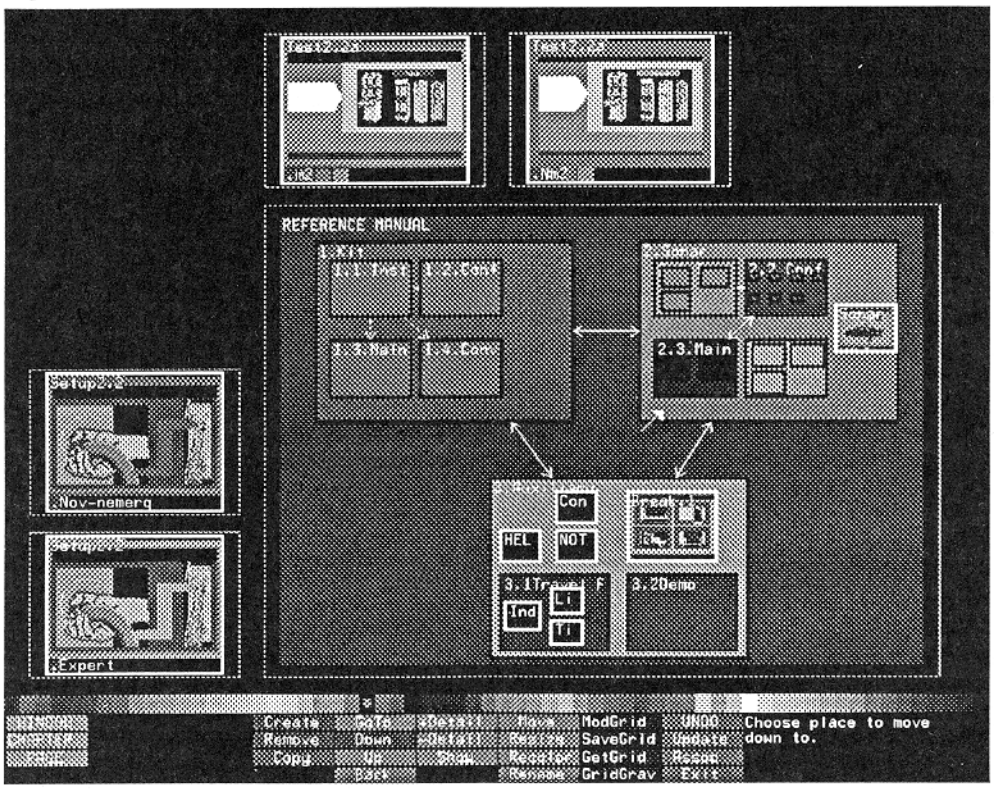


Figure 6: The document layout system. Additional detail in the three chapters is revealed.

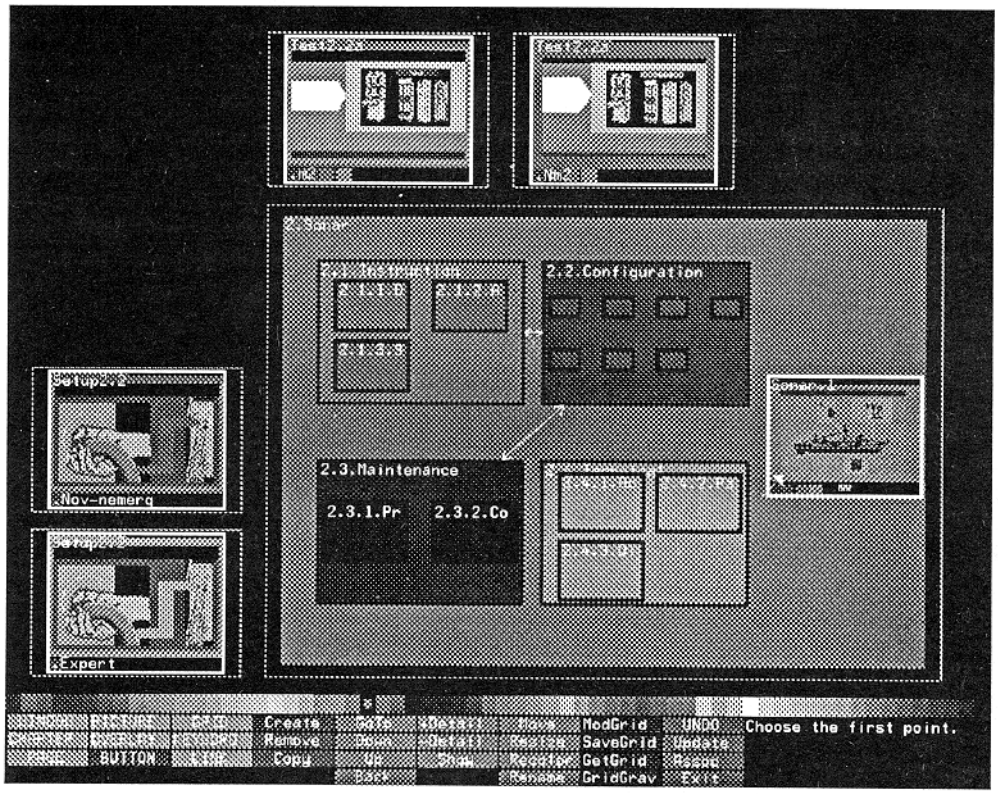


Figure 7: The document layout system. The user moves down the chapter hierarchy.

Movement up was accomplished by selecting a chapter from a displayed list of the chapter's or page's parents. As well, the user could select from a separate list the name of any other page or chapter in the document. This allowed the viewer to see many parts of the document at once and to create links between vastly separated parts of the document without having to search for the destination or source before resolving the link.

Link Inheritance

Much graph structure is created between pages embedded in the same chapter. Indeed, the chapter hierarchy is often chosen with this locality of reference in mind. Links may be made between arbitrary pages, however, without regard for chapter bounds. When all links are shown, as in Figure 4, the result may be an overwhelmingly dense graph, such as the Intermedia global map reproduced in [CONK87]. The use of a separate hierarchy allows the following link abstraction strategy, developed for IGD, which we call *link inheritance*. Links are shown directly only when they connect pages in the same parent chapter. When preparing to display a link between two nodes, IGD traces the chapter ancestries of both nodes to find their first common ancestor, which in the limit is the document itself. An arc is drawn between the two children of the common ancestor that are on the path to the two original nodes. In IGD, we also chose to suppress self-links entirely.

Figure 8 shows the links that are displayed when the link inheritance algorithm is applied to Figure 4. In IGD, at most one arc is drawn between any two chapters or pages, with arrowheads in one or both directions as appropriate. Thus, a myriad of connections between the children of one chapter and those of another may be represented by a single arc. The IGD screens in Figures 5-7 all show the results of link inheritance. For example, the double headed arrows between the subchapters in the chapter labeled REFERENCE MANUAL in Figures 5 and 6 indicate that at least one page in each subchapter references a page in the others. IGD also allowed the desktop on which the windows were placed to optionally be treated as a common parent of the chapters and pages at the outermost level of the windows. Links would then be drawn between the windows themselves.

Since links are only drawn between children of the same parent, visible arcs never cross out of the extent of their single containing chapter. By making each window contain the entirety of a single chapter or page, and not providing scrolling, we eliminated the problem of arc sources or destinations being clipped by the window boundaries. The graph display problem is thus localized through the divide and conquer power of the tree. The author must create sufficient hierarchy, however, to

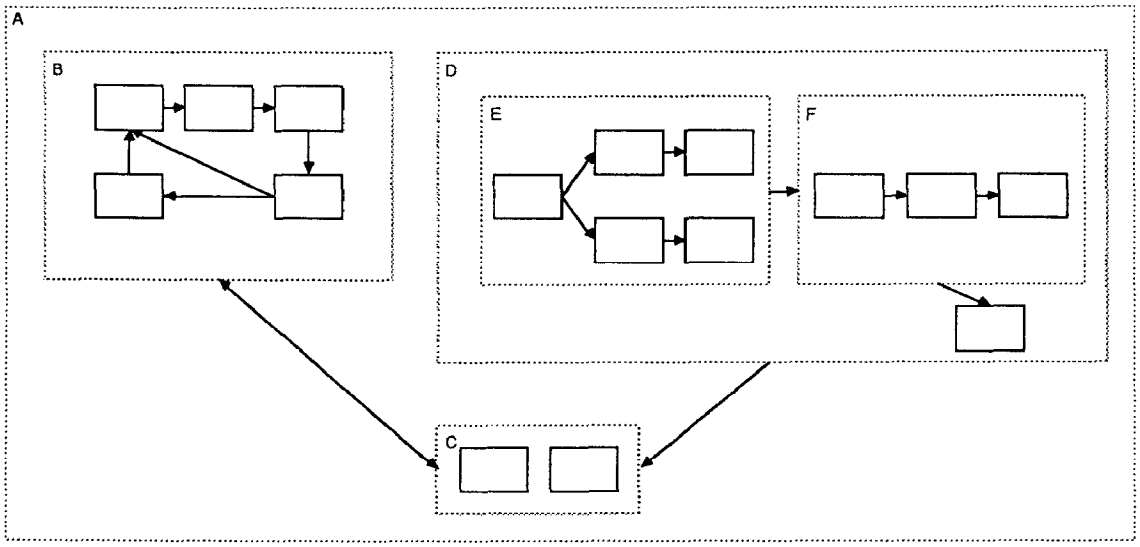


Figure 8: Nested set notation with link inheritance.

ensure that each chapter does not have more children than can be legibly depicted.

Although link inheritance hides information, at times it hides too much. IGD provided the ability to display explicitly either all links emanating from or all links terminating on a selected page or chapter. These were shown as arcs stretching directly between the linked nodes, drawn without regard to chapter or window boundaries.

IGD also allowed document authors to build a variety of displays that showed the user's place in the document. Some of these were graphical menus, constructed "by hand" using the system's editors. Authors who were programmers could write code that queried the document database at run-time to produce displays automatically. Examples of automatically produced keyword index, timeline, and neighbors displays that depict local views of the graph structure and chapter hierarchy are shown in [FEIN82].

5. Conclusions and Future Work

IGD embedded an arbitrary directed graph in a separately maintained hierarchy, as did several earlier hypertext systems. In contrast, IGD's document editor allowed authors to view and directly edit a hierarchical graph representation of a document's structure. A variety of information hiding and decluttering techniques were employed that relied on the hierarchy to reduce the amount of detail displayed, including that of the directed graph structure itself.

IGD used these techniques to modify the displayed structure in accordance with the combined directed graph and nested set display paradigms. Thus, the only changes that could be effected on the naive, fully detailed display were the deletion and creation of nodes and arcs and the modification of their attributes, such as position or graphical style. The only nodes that IGD deleted (for purposes of display) were those outside the outermost node in a window or inside a chapter whose detail was suppressed. No new nodes were created. Arcs were also removed by detail suppression and display selection. In addition, IGD suppressed arcs with the link inheritance algorithm and those associated with keywords that did not match a keyword filter that could be optionally placed on a window. For each arc deleted by the link inheritance algorithm, at most one new arc was created. Not counting the created arcs, no node or arc had its attributes modified.

We are currently investigating techniques, based in part on those used in IGD, for culling unneeded detail from directed-graph displays. Unlike the approach used in IGD, we are looking at heuristics for inserting extra levels of hierarchy automatically in places that would benefit from the simplified displays that they could produce. For example, in order to better concentrate on the relationships among a few of a node's children, a new child could be added to the node which could adopt the remaining siblings. The new child would become the sole other sibling of the children being studied, potentially reducing the number of arcs emanating from them and decluttering the parent node. The new node introduces a single visual abstraction for the set of remaining siblings. Alternatively, the children being studied might themselves be adopted by a newly created node.

The addition of new internal nodes will require local reformatting of the graph, complicated by the possibility that nodes being subsumed are not spatially contiguous. At very least, a new internal node could simply replace one of the nodes that it

was to adopt, while all of the adopted nodes could be scaled to fit inside it. In contrast to creating new internal nodes, existing ones could be selectively removed from the display (or simply ignored by the link inheritance algorithm) as a companion technique to expose more specific connections to or from the nodes that they contain.

Acknowledgements

IGD was the product of many other members of Brown's Computer Graphics group, including Kurt Fleischer, Imre Kovacs, Sandor Nagy, Joe Pato, Randy Pausch, Will Poole, Joel Reiser, Adam Seidman, Barry Trent, Mark Vickers, and Jerry Weil. David Salesin implemented most of the document layout system. Steve Hansen, Imre Kovacs, Charlie Tompkins, and Nicole Yankelovich used the system to design and edit the document shown in Figures 5-7. Andy van Dam provided the inspiration and support that made the project possible. IGD was funded in part by ONR Contract N00014-78-C-0396 and NSF Grant INT-7302268-A03. The author's current research is supported in part by DARPA Contract N00039-84-C-0165 and NY State Center for Advanced Technology Contract NYSSTF-CAT(87)-5.

REFERENCES

[CARM69] Carmody, S., Gross, W., Nelson, T., Rice, D., and van Dam, A. "A Hypertext Editing System for the /360." In M. Faiman and J. Nievergelt (eds.), *Pertinent Concepts in Computer Graphics*, U. of Illinois Press, Urbana, IL, 1969, 291-330.

[CONK87] Conklin, J. "Hypertext: A Survey and Introduction." *IEEE Computer*, 20:9, September 1987, 17-41.

[DELI86] Delisle, N. and Schwartz, M. "Neptune: A Hypertext System for CAD Applications." *Proc. ACM SIGMOD '86*, Washington, DC, May 28-30, 1986, *ACM SIGMOD Record*, 15:2, June 1986, 132-143.

[DONE78] Donelson. W. "Spatial Management of Information." *Computer Graphics (Proc. ACM SIGGRAPH '78)*, 12:3, August 1978, 203-209.

[ENGE68] Engelbart, D., and English, W. "A Research Center for Augmenting Human Intellect." *Proc. Fall Jt. Comp. Conf.*, 33, AFIPS Press, 1968, 395-410.

[FEIN82] Feiner, S., Nagy, S., and van Dam, A. "An Experimental System for Creating and Presenting Interactive Graphical Documents." *ACM Trans. on Graphics*, Vol. 1, No. 1, January 1982, 59-77.

[FEIN85] Feiner, S. "Interactive Documents." In P. Whitney and C. Kent (eds.), *Design in the Information Environment*, NY: A. Knopf, 1985, 118-132.

[FURN86] Furnas, G. "Generalized Fisheye Views." *Proc. CHI '86 Human Factors in Computing Systems*, Boston, April 13-17, 1986, 16-23.

[GROS87] Gross, J. and Tucker, T. *Topological Graph Theory*, NY: John Wiley, 1987, 28-29, 42-51.

[HALA87] Halasz, F., Moran, T., and Trigg, T. "Notecards in a Nutshell." *Proc. CHI + GI 1987*, April 5-9, 1987, Toronto, 45-52.

[HELF87] Helfman, J. "Bonsai: A Prototype Graph Browser and Graphical Interface Tool." Bell Laboratories Internal Memorandum, October 28, 1987.

[KNUT73] Knuth, D. *The Art of Computer Programming, 2nd Ed.*, Vol. 1, Reading: Addison-Wesley, 309-310.

[MEYR86] Meyrowitz, N. "Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework." *Proc. OOPSLA '86*, September 29 - October 2, 1986, Portland, *SIGPLAN Notices*, 21:11, November 1986, 186-201.

[NELS81] Nelson, T. *Literary Machines*. Swarthmore, PA, 1981.

[ROBE81] Robertson, G., McCracken, D., and Newell, A. "The ZOG Approach to Man-Machine Communication." *Int. J. of Man-Machine Stud.*, 14, 1981, 461-488.

[ROBI87] Robins, G. "The ISI Grapher: a Portable Tool for Displaying Graphs Pictorially." *Proc. Symboliika '87*, Helsinki, August 17-18, 1987, USC ISI Reprint Series ISI/RS-87-196, September 1987.

[ROWE87] Rowe, L., Davis, M., Messinger, E., Meyer C., Spirakis, C., and Tuan, A. "A Browser for Directed Graphs." *Software – Practice and Experience*, 17:1, January 1987, 61-76.

[SMIT87] Smith, J., Weiss, S., and Ferguson, G. "A Hypertext Writing Environment and its Cognitive Basis." *Proc. HYPERTEXT '87*, Nov. 13-15, 1987, Chapel Hill, NC, 195-214.

[SUGI81] Sugiyama, K., Tagawa, S., and Toda, M. "Methods For Visual Understanding of Hierarchical System Structures." *IEEE Trans. on Sys. Man and Cyb.*, SMC-11, 1981, 109-125.

[TRIC88] Trickey, H. "Drag: A Graph Drawing System." To appear in *Proc. EP88 (Int. Conf. on Elec. Pub., Doc. Manip., and Typog.)* Nice, April 20-22, 1988.

[VAND71] van Dam, A., and Rice, D. "On-Line Text Editing: A Survey," *ACM Comp. Surv.*, 3:3, September 1971, 93-114.

[YANK85] Yankelovich, N., Meyrowitz, N., and van Dam, A. "Reading and Writing the Electronic Book," *IEEE Computer*, 18:10, October 1985, 15-30.