Computer Corporation of America

575 Technology Square
Cambridge
Massachusetts 02139
617-491-3670

LEVEL II

12

Quarterly Research and Development Technical Report

6 Spatial Data Management System

11 30 Nov 79

12 50 p.

9 Quarterly research and development technical rept.

Contract Period Covered by Report: 1 Mar [illegible] - 31 May 1979

Computer Corporation of America

The views and conclusions in this document are those of the authors and should not be interpreted as necessarily representing the official policies, express or implied, of the Advanced Research Projects Agency, or the United States Government.

Report Authors: 10 Christopher F. Herot
David Kramlich
Richard Carling
Mark Friedell
John Thompson

Research Division
Computer Corporation of America
617-491-3670

Sponsor: Defense Advanced Research Projects Agency
Office of Cybernetics Technology

ARPA Order Number: 3487

ARPA Contract Number: MDA903-78-C-0122

15 ARPA Order-3487

Contract Period: 15 February 1978 - 30 November 1979

ORIGINAL CONTAINS COLOR PLATES: ALL DDC REPRODUCTIONS WILL BE IN BLACK AND WHITE.

DISTRIBUTION STATEMENT A
Approved for public release; Distribution Unlimited

387 285

Table of Contents

1. INTRODUCTION

This report describes the sixth quarter of work on the design and implementation of a prototype Spatial Data Management System (SDMS).

Spatial Data Management is a technique for storing and retrieving information which employs pictorial representations of data arranged on a collection of flat surfaces which the user can view on a color, raster-scan display. Tools are provided for interactively entering graphical and symbolic data, including a mechanism for generating graphical representations of existing, shared symbolic databases.

1.1 New Design Issues

Three new features of SDMS which will be implemented in the coming quarter are described in this report. They are the secondary data and subicon facilities in icon creation (chapter two), and the blinking/framing extension to the query language (chapter three).

The availability of secondary data in icon creation allows graphical representations of symbolic data entities to

reflect data that is related to, but not present in, those entities.

Subicons provide a method of displaying information in a graphically nested form.

To illustrate both secondary data and subicons, consider a naval database with a relation containing a tuple for each ship, and another relation containing a tuple for each officer on each ship. The secondary data and sub-icon facilities would allow the nesting of the officer and ship displays so that each ship could be displayed along with the corresponding officers.

The symbolic query language of SDMS will be augmented to include two additional commands: blink and frame. These commands, which take the standard retrieve command arguments, selectively retrieve icons by blinking or framing them in color. This allows a user to easily locate objects of interest in a large data space.

1.2 Completed Work

This quarter, several new features that expand the scope of data which can be manipulated and increase the utility of the system, were implemented. These features are:

1. a mechanism for displaying formatted text

2. extensions to the graphical editor

3. a rapid transit mechanism for moving about in the graphical data space

4. an integrity maintenance facility

The program for display of ascii text is described in chapter four. The user can define a rectangular symbol on the graphical data surface to be a text port. If the user zooms in on this port, the corresponding document appears on the screen. Documents are typically the output of the Unix nroff [OSSANNA] program and are stored as files in the Unix file system. If the document was formatted with a table of contents, as can be easily done with nroff macros, the table of contents can be displayed on one of the auxiliary monitors and used to indicate and change the portion of the document visible on the main display.

Chapter five discusses the interactive graphical editor which has been greatly augmented since last described [HEROT et al.]. Of special interest is a feature which automatically scales down anything drawn by the user and displays it on the world-view map.

The program which allows the user to move about the graphical data space now provides a rapid transit (chapter 6) facility to allow the user to quickly jump from one spot to another without the necessity of displaying all of the intervening space. By pressing the appropriate button on the tablet puck, the usual highlighted rectangle on the world view map is replaced by a cursor which can be moved to the desired destination. Then, by pressing another button, that location becomes the user's new position on the graphical data surface. In addition, a hierarchy map can be displayed to show the nesting of the data surfaces. One such map is shown in Figure 1.1. The rapid transit feature can be used on the hierarchy map to move among data surfaces.

The integrity maintenance facility maintains the correspondence between graphical and symbolic representations of information; its design is discussed the quarterly technical report covering the SDMS contract period December 1, 1978, to February 28, 1979. This facility maintains a record of the dependencies of the graphical representations on the symbolic database. If the symbolic database is updated, as could happen as the result of user interaction or background input from some external, possibly remote, source, the graphical

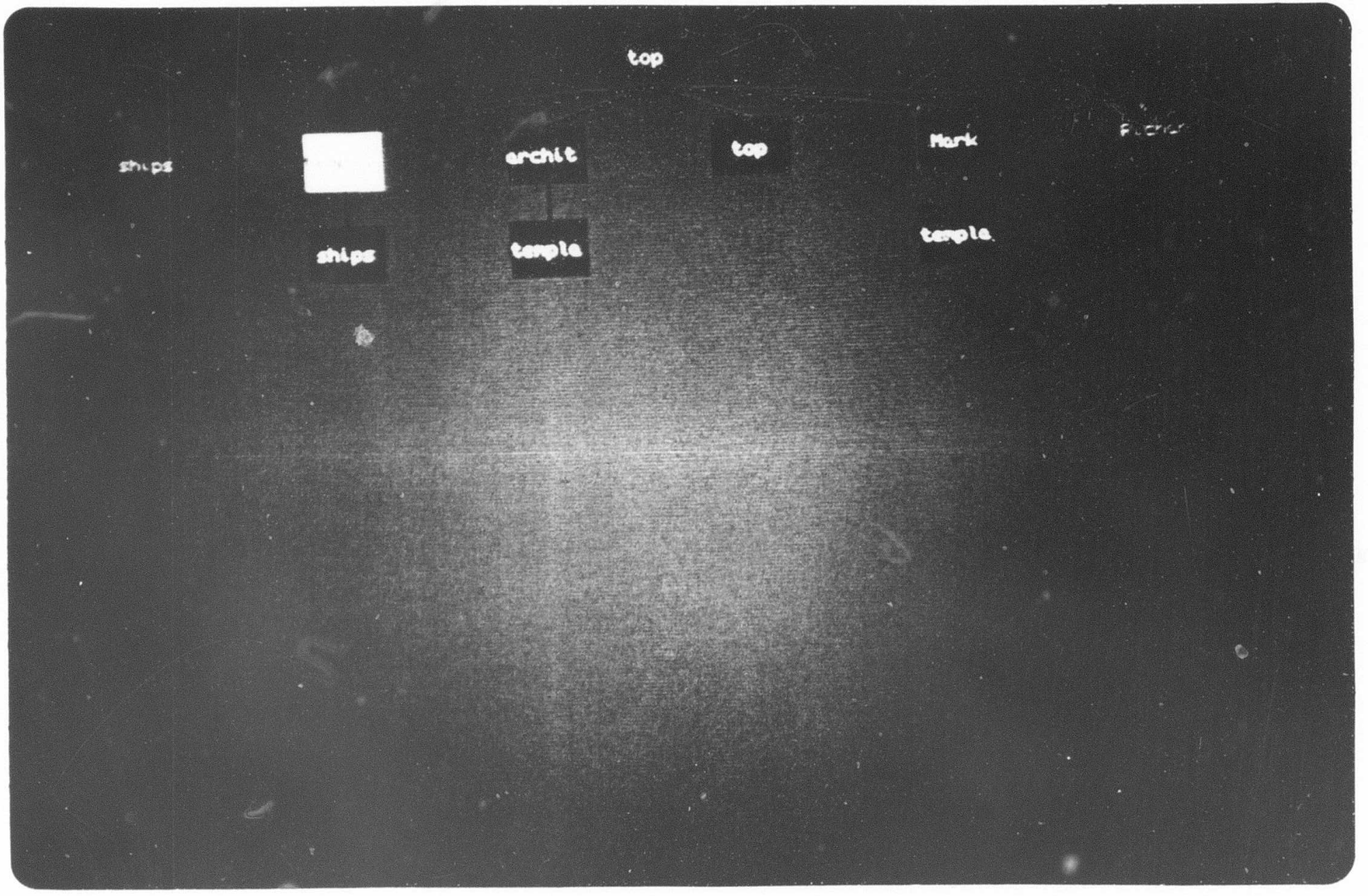

Figure 1.1

Hierarchy map shows ispace organization in the graphical data space.

representations which were derived from that data are re-created to reflect the new data values. In this way, SDMS may be used to examine dynamic databases with the assurance that the display reflects the current values of the data.

2. ENHANCEMENTS TO ICON CREATION

The SDMS database command language SQUEL (Spatial QUEry Language) includes two statements that direct SDMS to graphically represent symbolic data from the host database management system. These are the ASSOCIATE and DISPLAY statements. INGRES [HELD STONEBRAKER WONG] is the host database management system.

The ASSOCIATE statement produces a graphical representation, referred to in SDMS terminology as an "icon", of each tuple in a specified INGRES relation. An important SDMS function is to continuously maintain the correct graphical representation of each associated relation, even if tuples are added, deleted, or modified. This functionality is referred to as graphical integrity maintenance. The quarterly technical report covering the SDMS contract period December 1, 1978, to February 28, 1979, describes graphical integrity maintenance in detail.

The DISPLAY statement produces icons of only those tuples meeting an optionally specified qualification. If no qualification is present, an icon will be created for each tuple in the relation. Graphical integrity maintenance does not apply to icons created via the DISPLAY

statement.

ASSOCIATE and DISPLAY statements must specify an Icon Class Description (ICD). The role of the ICD is to specify the appearance of an icon as a function of the symbolic data it represents; an ICD is a mapping from symbolic data to graphical data. ICDs are written in Icon Class Description Language (ICDL).

Icon creation is the process that actually constructs icons in the graphical data space. Icon creation may be described as an ICDL processor which accepts a tuple as its input and produces an icon as its output.

In response to an ASSOCIATE or DISPLAY command, SDMS activates the association processor, an icon creation executive, that repeatedly invokes icon creation, once for each tuple in the relation.

2.1 Secondary Data

Current SDMS development in the symbolic database - graphical database interface area has focused on providing icon creation with data from INGRES not contained in the tuple being graphically represented, but which may be useful in constructing its icon. This data is referred

to as "secondary data" or "data from secondary tuples". Secondary tuples may or may not be in the relation being associated or displayed.

As an example of how secondary data is useful, consider an INGRES relation of ships and an SDMS user who requires repeated immediate access to data concerning these ships, including the rank of the commanding officer. Suppose also that the ships relation includes the commanding officer's name, but not his or her rank. An associate or display of the ships relation that produces icons specifying the commanding officer's rank is still possible if the secondary data mechanisms of icon creation are employed. Given that a personnel relation exists, icon creation can use the commanding officer's name as a key to look up his or her rank.

2.1.1 Additions to Icon Class Description Language

All relations from which secondary tuples may be retrieved must be declared in ICDL:
SECONDARY DATA FROM RELATION relation_name_list.

ICDL allows relation names to be used in lieu of range variables. Occasions arise, however, in which multiple range variables over the same relation are required.

These additional range variables can be declared as follows:

```
RANGE OF range_variable IS relation_name
```

Retrieving secondary data is allowed in ICDL wherever a simple assignment statement is allowed. A retrieve operation is accomplished by a special case assignment statement:

```
RETRIEVE variable = attribute WHERE qualification.
```

Normal execution of the retrieve statement implies that exactly one tuple meets the qualification. If no tuple meets the qualification, a fatal error condition exists, and the icon creation process is aborted. If multiple tuples meet the qualification, the first tuple retrieved from INGRES is used and a warning message is issued.

2.1.2 Implementation Issues

Input tuples to icon creation are passed from the SQUEL processor to icon creation via a UNIX file, hence no direct interface between icon creation and INGRES was required previous to the addition of secondary data retrieval capabilities.

Adding an INGRES interface to icon creation is, unfortunately, rather expensive. A second instance of the

four processes comprising INGRES is necessary because no provision exists for multiple process use of one INGRES. Fortunately, UNIX provides a "shared text" feature which allows multiple instances of the same process to execute from the same set of instructions. This serves to substantially reduce the memory requirements of icon creation. Further, icon creation does not automatically spawn a second INGRES; the first secondary data declaration appearing in an ICD initiates the INGRES interface.

The overhead required to establish the second instance of INGRES, specifically loading the INGRES processes' data areas in memory and initiating the processes in UNIX, is reduced by storing INGRES's load files on the UNIX swap device. This UNIX feature is activated by setting the "sticky" mode of all INGRES load files.

2.1.3 Performance Considerations for the SDMS User

There is a price to be paid for the use of secondary data in icon creation. To summarize the previous section, a cost is incurred in establishing the provisions to use secondary data. These provisions are NOT automatically available; the first secondary data declaration appearing in an ICD initiates them, hence this overhead is incurred

only when secondary data is used.

Significant additional overhead results from the rather slow speed of the INGRES database management system. Each ICDL RETRIEVE statement requires an INGRES retrieve operation. Typically, a retrieve operation can be completed in fifteen seconds or less, although retrieves consuming hours of processing time have been reported in the INGRES literature.

What are the alternatives to the use of secondary data? How should these alternatives be considered in light of the secondary data overhead?

The primary alternative to the use of secondary data in an icon class description is the creation of a new primary relation which contains all the data items required. A less desirable alternative is to simply forgo the use of secondary data. This results in a less informative, but faster executing, ICD.

As discussed previously, SDMS graphical integrity maintenance mechanisms guarantee consistency between associated relations and their graphical representations. This consistency is effected by re-creating icons that reflect modified tuples. Re-creation of an icon consists of erasure, followed by re-invocation of icon creation. The repeated execution of icon creation, which results from

continuous update of an associated relation, stresses the importance of fast ICD execution. It is in this situation that consideration of an alternative to secondary data in the ICD is most appropriate.

2.2 Subicons

Icon class descriptions may include subicons within the area of a regular icon (referred to in this context as the "parent" icon). Subicons may be used to augment the appearance of the parent icon, or, they may be visually disjoint. Subicons are only one iplane deep; however, the impression of a multiple iplane subicon can be achieved through the use of subicons on multiple iplanes. This is analogous to the construction of parent icons from template icons on multiple iplanes. Subicon descriptions are relative to the parent icon; the "Ispace" of a subicon is its parent. Subicons may be a component of the parent icon, or they may be visually disjoint.

Subicons which are used to construct part of the parent icon are useful in specifying the appearance of only that part as a symbolic data function. For example, a flag subicon of a ship parent icon would allow the same icon

class description to present ships of various nationalities.

Visually disjoint subicons are useful when presenting data related to, but not part of, the entity presented by the parent icon. An escort aircraft subicon of a ship parent icon is an example.

2.2.1 Subicon Descriptions in ICDL

Parent icons are described on each iplane by an iplane block. An iplane block consists of a header, followed by iplane statements. The header specifies which template icon to use. Iplane statements specify modifications to the template.

Subicon descriptions are very similar to parent icon descriptions; each iplane of a subicon is described by a subicon block consisting of a header, followed by iplane statements. A subicon block differs from an iplane block in that its header specifies subicon position. Subicon position is given in coordinates relative to the origin of the parent icon.

3. BLINKING AND FRAMING

3.1 Function

Blinking and framing provide a mechanism for the SDMS user to selectively "retrieve" icons which have been created by an associate or a display. In this context, "retrieval" means temporarily modifying the selected icons in the graphical data space (GDS) to make them visually distinctive. This may be accomplished in one of two ways. The first is blinking, the second is framing.

The user causes a subset of icons from an association or display to blink by issuing a BLINK command in SQUEL. The command specifies the relation whose icons are to be blinked and an optional qualification which will select some subset of the associated icons. The consequence of this command will be to cause the selected icons to be blinked wherever they might be in the GDS. Blinking of the selected icons continues until an UNBLINK command is issued.

The user may frame selected icons in a manner analogous to blinking. There are FRAME and UNFRAME commands in

SQUEL which are identical in syntax to the BLINK and UNBLINK commands, respectively. The effect of a FRAME command is to draw a blinking frame around the selected icons. Icons continue to be framed until an UNFRAME command is issued.

The modifications to an icon's appearance resulting from a blink or frame are made only on a copy of the icon's bit pattern in internal memory. The stored image of the icon is in no way modified.

3.2 Implementation

The implementation of blinking and framing in SDMS affects four different processes. Those processes are: (1) the SQUEL interpreter, where the command is parsed; (2) the modification process, where the data structures necessary for blinking and framing are managed; (3) the stager, where the modifications to the icons are performed; and (4) the blinking process, where the actual blinking of the visible icons on the display is done.

This section will describe that portion of each of these processes which pertains to blinking and framing.

3.2.1 SQUEL

The SQUEL interpreter, once it has parsed a blink, frame, unblink, or unframe command, must create a list of icon-ids for the affected icons. This list and a command signifying what modifications are to be made to the icons contained in the list are passed to the modification process. The mechanism for communication between SQUEL and the modification process is a set of pipes with synchronizing locks.

3.2.2 Modification Process

The modification process is responsible for several tasks. It must: (1) process icon lists arriving from the SQUEL interpreter; (2) manage a database of active modifications; (3) generate and manage data structures of modifications to be performed on a per tile basis; and (4) process requests from the stager for explicit information on modifications to be performed.

When the modification process receives a command from the SQUEL interpreter, processing may take either of two branches. Blink and frame commands take one path, unblink and unframe commands take the other.

3.2.2.1 Blink/Frame

To execute a blink or frame command, the modification process must ask the Icon Manager for detailed information about each of the icons in the list that accompanies the command. It must know the following about each icon:

- the I-space in which the icon is located
- the X and Y origin of the icon (universal coordinates)
- the width and height of the icon (universal coordinates)
- the color of the icon

This information about each icon, as well as the icon-id and modification to be performed (blink or frame), is inserted into the modification database. The modification database is implemented as a hash table, the icon-id being hashed upon.

At the same time that the icon records are being inserted into the mod(ification) database, several other data structures must be updated. These data structures are a bitmap of tiles for each iplane in the icon's I-space, and a B*-tree of affected tiles for each iplane in the icon's I-space. These data structures are described in detail in Section 3.3 of this document.

As each icon record is inserted into the mod database, the following processing takes place. For each iplane in the icon's I-space, compute the iplane coordinates and extents of the icon. Compute which tiles are affected in that iplane. Set the corresponding bits in that iplane's tile bitmap. For each affected tile in the iplane, compute the bounds of the area affected within the tile. Insert an entry for the affected tile into that iplane's B*-tree. The entry contains the icon-id of the selected icon, the bounds within the tile to be modified, what kind of modification to make, and what color to use.

3.2.2.2 Unblink/Unframe

When an unblink or unframe command is issued, the SQUEL interpreter passes the modification process a list of the affected icon-ids. The modification process must look up each of these icon-ids in the modification database. Once the appropriate icon record has been found, the icon's I-space is known. For each iplane in the icon's I-space, the icon's iplane coordinates must be calculated. From the iplane coordinates, a list of affected tiles can be generated for the iplane. For each affected tile, the corresponding bit in the iplane's tile bitmap is cleared. Then for each affected tile, the entry in the

iplane's tile B*-tree is removed.

3.2.2.3 Processing Common to Blinking/Framing and Unblinking/Unframing

Any modifications to tiles must also be passed on to the Stager so that tiles which are resident in the core buffer may be updated to reflect the modification. The mechanism for doing this is the modification queue, described in Section 3.3.

The Stager may also send commands to the modification process. These commands may be either a request for information about a particular tile or a notification of an impending change in the current iplane.

The modification process tries to keep as much of current iplane's tile B*-tree in core as possible (this mechanism is described in the section on data structures). This allows requests for information from the Stager to be processed quickly and efficiently. In addition to the B*-tree, the iplane's bitmap is kept in a portion of memory shared by the Stager.

When a request from the Stager for information about a particular tile arrives, the modification process must look up the tile record. Once that has been retrieved,

the information requested is placed in the modification queue in shared memory.

When the Stager changes the current iplane, the modification process must be informed of the change. The modification process must load the bitmap for the new iplane in shared memory. Portions of the new iplane's tile B*-tree will be loaded as requests for information from the Stager are processed.

3.2.3 Stager

The Stager must perform several tasks in support of blinking and framing. These tasks are: (1) determining whether a tile needs to be modified; (2) asking for information if a tile does need modification; (3) performing the relevant modifications; (4) feeding the modified tiles to the display; and (5) telling the modification process when a change in iplane is about to happen. In addition, a separate process, diskio, examines tiles which the Stager has discarded. Tiles which must be written back to disk (i.e., those which have been modified with the GDS editor) first have any modifications caused by blinking or framing stripped. Diskio, while physically a separate process from the Stager, is logically a part

of the it.

The Stager determines whether a tile needs modification by examining a bitmap managed by the modification process in shared memory. Each tile has a bit indicating whether it should be modified. If the tile does need modification, the Stager must ask the modification process for the specifics of the changes.

The Stager sends a request to the modification process specifying the tile-id that it needs to know about. Since the modification process and the Stager are kept in synchronization regarding the current iplane, no other information is required. When the modification process has retrieved the appropriate tile record(s) (there may be more than one modification per tile), the information is returned to the Stager via the modification queue.

Every time the Stager cycles through its run loop, it checks to see whether there are any entries in the modification queue. If an entry is relevant to the current iplane, additional processing must take place. An entry in the modification queue is relevant if it specifies both the current iplane and a tile which is currently loaded in core. If the Stager detects an entry which specifies the current iplane and a tile which is slated to be loaded in core, but is not yet present, it leaves

the entry in the queue to be processed in a later cycle. Entries which satisfy neither of the conditions above are discarded.

When a relevant entry is detected, the Stager must modify the tile in accordance with the information in the modification record. If the modification to be made is a blink, then all of the pixels within the specified bounds of the tile that are of the specified color must be mapped to the blinking version of that color. The mapping is a straight-forward process of turning on a high-order bit in each affected pixel. If the modification to be made is a frame, the Stager must determine what portion of the icon is contained within the tile. The row and/or column of pixels which forms the boundary of the icon is then mapped to the blinking color.

Once the modifications have been made to the tile, the Stager sets a status bit in the tile header indicating that the modification has been completed. At the same time, it sets a bit indicating that the tile has been changed in some fashion and needs to be re-fed to the display. On the next cycle of the run loop, the modified tile(s) will be fed to the display.

When the Stager detects that a change in iplane is about to take place, it must notify the modification process of

the impending change. The modification process must then load the tile bitmap for the new iplane into shared memory. Note that the bitmap for the current iplane is not overwritten, but that the new bitmap is written in a different area. The stager uses this separate, new bitmap when feeding tiles for the new iplane. When the transition to the new iplane is complete, the new bitmap is copied over the old, current one.

Diskio examines tiles which are returned it by the Stager to determine whether they need to be written back to disk. Tiles which have not been updated, need not be written. If a tile has been updated by the GDS editor and has been modified by a blink or frame, the affected pixels must be changed to their normal values before being written. Diskio simply checks for the update bit and the modified bit in the tile header, if both are set then the blinking bit must be turned off in each pixel. If only the update bit is set, the tile may be written immediately. If only the modified bit is set, the tile may be discarded.

3.2.4 Blinker

The blinking process is responsible for changing the video lookup table (vlt) in the main display at a regular interval. The effect of changing the vlt periodically is to cause those pixels on the screen which index that part of the vlt which is changing to appear to blink. N times a second (where N is to be determined empirically) the blinking process toggles the portion of the vlt in the main display which is indexed by the blinking colors.

The rationale for controlling the blinking in a separate process is that a more regular period between blinks can be obtained. If blinking were controlled by the Stager, the rate would be dependent on user motion, etc.

3.3 Data Structures

The modification process maintains three different kinds of data structures in support of blinking and framing. These data structures are: (1) the list of active modifications; (2) the tile B*-tree, indicating what modifications are to be made to each tile; (3) the tile bitmap, indicating whether a tile needs modification; and (4) the modification queue, where information is returned to the

Stager.

The modification process keeps a list of icons that have modifications outstanding on them. The list is actually implemented as a hash table, the icon-id serving as the hash key. This allows rapid and efficient access to any modification record in the case of deletion. This list remains in core at all times. The following items are stored in each modification record:

- icon-id
- operation to be performed
- color to be modified (ignored for frames)
- X and Y origin of icon (universals)
- width and height of icon (universals)
- I-space of icon

For each affected iplane in the GDS, the modification process creates a B*-tree containing one or more records for each tile in the iplane which requires modification. The tile-id serves as the key for the record. Tiles which have more than one modification outstanding have records which are linked together. The format of a tile record is the following:

- icon-id
- operation to be performed

- color to be modified (ignored for frames)
- X and Y origin of icon within tile (tile coordinates)
- width and height of icon within tile (tile coordinates)
- link to other records with same tile-id

The B*-trees normally reside on disk, as they will tend to be large in size for iplanes with a high scale factor. The B*-tree structure was chosen because it is one of the most efficient structures for external store devices. Records are grouped into pages, sorted within the page. The records are organized so that the proper page can be found with a minimum of disk activity. Once the proper page is in internal memory, a rapid search of the records in the page can be made.

The modification process will operate a paging scheme for referencing the pages of a B*-tree. The algorithms which manipulate the B*-tree will issue virtual read and write commands to the pager. The pager keeps many different pages in memory. Physical reads and writes occur only when the desired page is not already resident in memory. Memory buffers are re-used on a least-recently-used (lru) basis. The base page of a B*-tree will remain in memory as long as that iplane is the current one.

The modification process maintains a tile bitmap for each affected iplane in the GDS. These bitmaps normally reside on disk. They are read into core when they need to be modified. The bitmap for the current and next iplanes are loaded into shared memory and remain there until the current or next iplane changes.

Each tile has a bit in the bitmap. The bit is set when the tile has a modification pending. It is cleared if there is no modification pending.

The modification process returns information about modifications to tiles to the Stager via a modification queue in shared memory. Each slot in the queue has the format:

- iplane-id
- tile-id
- X and Y origin of icon within tile (tile coordinates)
- width and height of icon within tile (tile coordinates)
- operation to be performed
- color to be modified (ignored for frames)

4. TEXT IN SDMS

SDMS now has the facility to incorporate textual information into the graphical data space. This feature allows access to on-line textual information such as reports, system documentation, news, and mail. On the iplane these documents appear in an iconic form, such as a picture of the cover of a book. These pictorial representations of documents are called text ports. When the user zooms into a text port, requesting further detail, the actual text of the document appears. If a table of contents is available, it, too, is displayed. The user is able to scroll up or down through the document. Also, different sections can be accessed rapidly by selecting from the table of contents display.

4.1 Activating a text port

Typically, the user will be in an iplane populated by icons representative of different documents. Icon appearance can be used to indicate the type of document. For example, icons for quarterly reports may be green and larger than blue system documentation icons. Zooming

into an icon will give an increasingly magnified view of the icon until the actual text of the document appears. If available, a table of contents will also appear on the right display. A cursor on this display will be to the left of the line containing the title of the section currently on the main monitor. Pressing the joystick forward will cause the text to scroll up with new lines of text appearing at the bottom of the display. Pulling the joystick backward will scroll the text down. The cursor on the table of contents will always indicate the title of the section currently displayed. Figure 4.1 shows an example of a table of contents.

A rapid transit feature is available to immediately display a selected section of text. This feature is activated by pressing the "menu-select" button on the tablet puck. The cursor on the table of contents display will now be controlled by the vertical position of the puck. The cursor is fixed horizontally. After positioning the cursor to the desired section title, pressing the "do-it" button will cause the beginning of the selected section to appear on the main display. Once the table of contents is selected, it, too, may be scrolled. To exit and return to the original iplane, the user zooms out by turning the knob on the joystick counterclockwise.

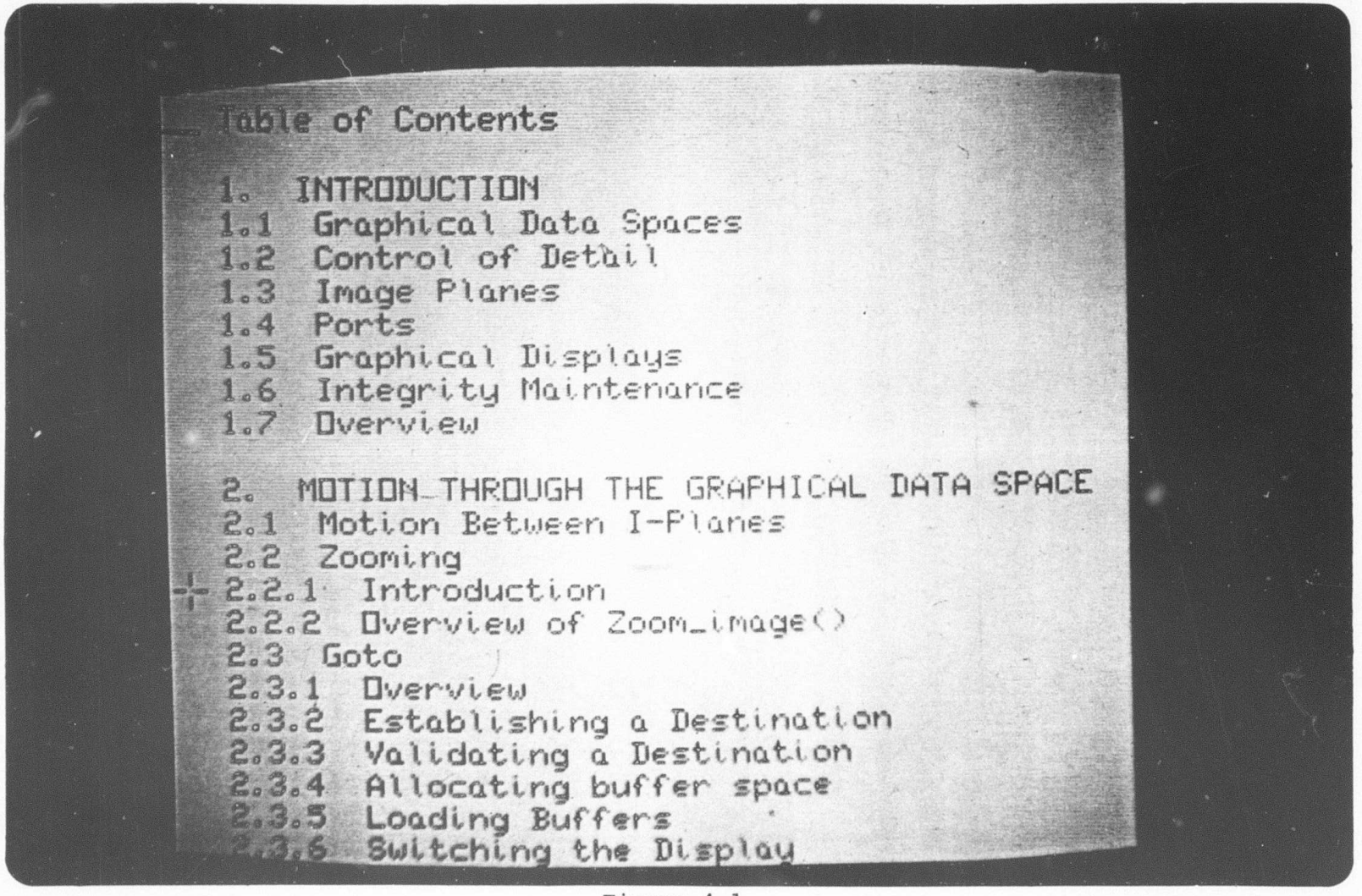

Figure 4.1
Table of contents.

Cursor indicates that text document section 2.2.1
is currently displayed on the main screen.

4.2 Making a text port

The SDMS graphical editor is used to add text ports in the graphical data space. The editor has a general facility to create ports and associate UNIX command strings with them. The associated command string is executed when the user zooms into the port. To create a text port, the editor is used to draw an icon. Then, a command string is attached to it; the result is a text port. The command string consists of the name of the text display program and the name of a text file. When the user zooms into this port, the command string is executed; the text display program is activated and the name of the text file is passed to it. The UNIX text formatter nroff [OSSANNA], is usually used to generate the table of contents for the text file. Rapid transit is not available for text files without a table of contents.

4.3 Options of the text display program

The command string used to execute the text display program can specify a variety of options. They are: the height and width of the main display; text size; whether

to use the existing color table or load a new one for either display; and the text and background colors for either display.

Two text sizes are available in SDMS. The default size, size two, results in forty-six characters per line. Size one allows twice as many characters per line. However, this text is somewhat more difficult to read.

5. Paint

5.1 User Environment

The graphical editing system (paint) can be invoked by the user for use in any Ispace into which he or she can travel and for which he or she has permission to modify. Paint is easily invoked via a function button located on the tablet puck. This causes a menu of interactive input commands to appear on the rightmost monitor (figure 5.1). The main (center) monitor continues to present the currently staged information, with the addition of a color table (palette) across the bottom of the screen. The palette provides a method of easily selecting a new color to be used in editing the iplane.

The user controls the mode of operation by use of the four function buttons provided on the tablet puck. The buttons are defined as follows: the top center button (referred to as the "doit" button) is used to enter information concerning point locations on the screen.

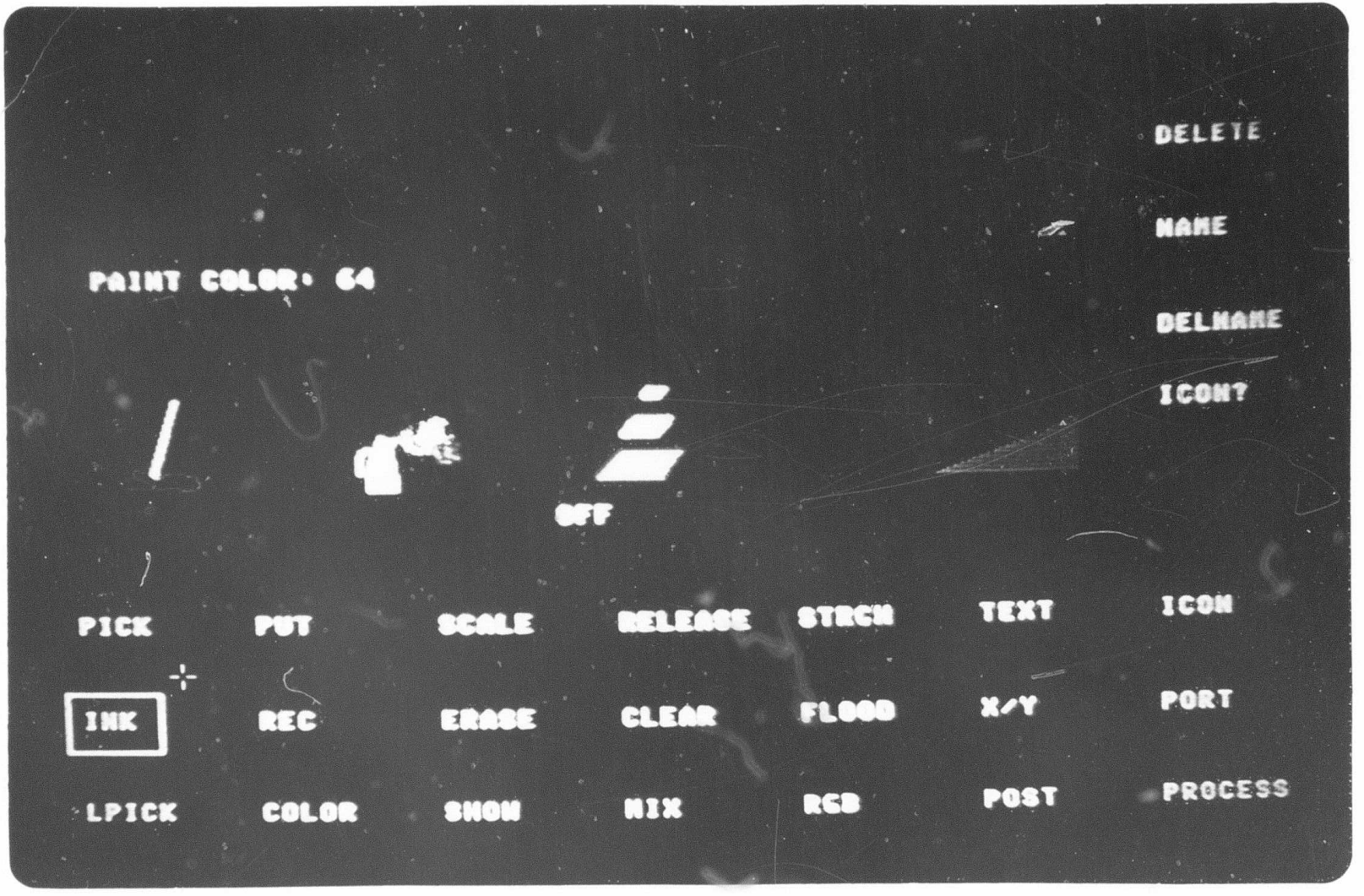

Figure 5.1
Graphical Editor Menu

The left button exits paint and enters rapid transit. The button to the right of center directs the user to the menu. The bottom button is used to abort paint operations such as flood, pick and stretch.

To exit paint, the user need only move the joystick in any desired direction. The menu display darkens and the palette is removed from the staged screen.

Paint provides the user with a scratch buffer enabling him to save a user-defined portion of the current iplane. The user may then maneuver to another iplane (using the joysticks, travelling through ports, etc.), re-enter paint, and place the information at the new location. The user may optionally scale the information when it is placed.

Paint also provides the user with the ability to automatically scale down (miniaturize) any modifications to the current iplane onto all iplanes of lower levels of detail in the same Ispace. This capability is provided by a background daemon which is sent a list of modified tiles whenever the user exits paint. Miniaturization may be enabled or disabled at any time.

5.2 User Interaction

Upon entering paint the user is provided with a set of tools for generating and modifying any currently staged area in the iplane. These tools allow a user to work with simple geometric shapes such as lines and rectangles or with specific objects previously created.

Paint also includes a set of operations for defining and deleting objects known to the icon manager. These include: the ability to create icons, ports, and process ports; to name ports and icons; and to delete icons or their names.

The graphically oriented functions are summarized as follows:

LINE - draw a line.

width - optionally selected to enter a new line width specification.
color - optionally selected to pick a new color.

operation:
Whenever the "doit" button is depressed a line is drawn on the screen (as if the ink flowed from the cursor).

STRETCH - draw a rubber band (straight) line.

width - optionally selected to enter a new line width specification.
color - optionally selected to pick a new color.

operation:
The user moves a cursor to one endpoint

of the line and activates the "doit" button. Now, as the user moves the cursor a straight line from the initial point selected to the current cursor location is drawn in highlight. When the cursor is located at the desired second point, pressing the "doit" button causes a straight line of the desired color to be drawn.

RECTANGLE - draw a rectangle

Color - optionally selected to pick a new color.

operation:
The user moves a cursor to the desired location of the rectangle and activates the "doit" button. Now, as the user moves the cursor a rectangular shape follows, leaving one corner fixed at the initial position indicated. When the user locates the second point, pressing the "doit" button causes a rectangle of the desired color to be drawn.

Many of the following commands use this method of selecting diagonally opposing corners to enter a rectangular area.

ERASE - draw a filled rectangle

Color - optionally selected to pick a new color.

operation:
A rectangular area is entered by selecting diagonally opposing corners. The inclusive area defined by this rectangle is set to the desired color.

CLEAR - clear a tile to a specified color

Color - optionally selected to pick a new color.

operation:
A cursor is provided. Each time the user activates the "doit" button, the tile in which the cursor is located is set to the current color. This provides a method for clearing a large area relatively quickly.

FLOOD - fill an area with a specified color.

Color - optionally selected to pick a new color.

operation:
The cursor is positioned within the area to be flooded and the "doit" button activated.

PICK - defines an area to be copied.

operation:
A rectangle is defined by selecting diagonally opposing corners. The inclusive area defined by this rectangle is loaded into the pick buffer.

PUT - defines a location for the copied area to be placed.

operation:
The size of the current pick buffer is displayed via a rectangular cursor. When the user has it positioned as desired the "doit" button is activated and the object is placed in the image plane.

ISCALE - defines a location and size for the copied area to be placed in.

operation:
the user defines a rectangular area. The object in the pick buffer is scaled into the area defined.

RELEASE - allows the user to release the current pick buffer

operation: The memory allocated for the pick buffer is returned to the memory management system. This operation automatically occurs when a user puts something new in the pick buffer. This command automatically puts the user back into the previous command.

TEXT - allows the user to insert text.

Color - optionally selected to pick a new color.

operation:
The user enters the text and its scale on the keyboard. Once entered, the size of

the text field is displayed via a rectangular cursor. When the user has it positioned, the "doit" button is activated and the text appears as desired.

GRID - places a grid on the screen for use in alignment of objects.

Width - selected to define a new size grid to be used. The user enters two points defining the grid scale.

operation:
A grid appears on the image plane (non-permanent). When placing or drawing objects, positioning the cursor near a grid boundary causes the cursor to 'snap' to the boundary for easy alignment of independent objects on the iplane.

SNAP - enables or disables the snap effect of the grid.

MIX - allows the user to mix colors in the palette.

Color - optionally selected to pick a new color to mix.

operation:
The user uses the three slide controls to mix the given slot to the color desired. The three controls are defined to be intensity, hue, and saturation.

RGB - allows the user to mix colors in the palette using RGB.

Color - optionally selected to pick a new color to mix.

operation:
The user uses the three slide controls to mix the given slot to the color desired. The three controls are defined to be the intensities of red, green, and blue, respectively.

SHOW - allows the user to re-display the staged image.

operation:
The screen is redrawn and the user placed back into his previous mode. This is used to verify what the current state of the staged image is.

X/Y - allows the user to obtain coordinates of any point in the staged image.

operation:
Each time the user activates the "doit" button the coordinates (iplane and universal) of the cursor's position are displayed on the menu monitor.

The remaining functions define an image area's characteristics.

MAKE ICON - defines a particular icon.

operation:
the rectangular area is defined using the puck and the object becomes known to the system as an icon.

MAKE PORT - defines a rectangular area thru which the user may pass to enter a new Ispace.

operation:
Two independent operations are required, the first is to define the port boundaries, the second is to define the port destination. The port boundaries are defined by entering a rectangular area. The port destination is then entered as Ispace, iplane, scale, and x,y location. The user may immediately zoom thru the port.

MAKE PROCESS PORT - defines a rectangular area thru which the user may pass to enter an arbitrary process.

operation:
the process port's location is defined by entering a rectangular area on the currently staged iplane. To associate the port with a process the user types the process path name and any arguments at the keyboard.

NAME ICON/PORT - defines a name by which the user may reference an icon or port.

operation:
The user points to the object to be named. If the object is an icon or a port, the user types in the name desired. The name is then associated with the object in the icon manager's data base.

DELETE ICON/PORT NAME - deletes the name used to reference an icon or port.

operation:
The user points to the object which is to have its name deleted. If the object is an icon or a port, its name (if there is one) is deleted.

DELETE - allows a user to delete icons, ports and processes ports.

operation:
The user points to the object to be deleted. If the object is an icon or a port, it is deleted from the icon manager's data base. The graphical representation can then be erased or the object redefined if desired.

FIND ICON - allows a user to interrogate spatially for icons and ports.

operation:
The user points to a location. If that point is contained in an icon or a port, the boundaries of the icon/port are highlighted and its name and id are displayed on the menu monitor.

6. RAPID TRANSIT

6.1 Function

Rapid transit allows the user to move about the graphical data space without the necessity of staging intervening data in which the user is not interested. Two types of rapid transit are available. One is a "local" rapid transit, which allows the user to move about the currently visited iplane. The other is an "inter-Ispace" rapid transit, which allows the user to move rapidly between Ispaces.

Rapid transit is invoked by activating the left button on the tablet puck. After rapid transit is activated, a cursor appears on the navigational aid monitor which tracks the location of the tablet puck. When the cursor is positioned over the desired destination, the user activates the "doit" button. The user is quickly transported to the selected area.

The mode of rapid transit invoked is dependent upon the type of navigational aid displayed. When the user has the world view map displayed, rapid transit will be of the

local variety. When the hierarchy map is displayed, rapid transit will be of the inter-Ispace variety. The user may switch between modes of operation while in rapid transit.

6.2 Implementation

The implementation of rapid transit required modifications to the Stager and Hierarchy Map processes. The extent and nature of those modifications are the subjects of this section.

6.2.1 Stager modifications

The stager was modified to call the rapid transit subroutine when it detects that the left button on the tablet puck has been activated. It is the responsibility of this subroutine to tell the stager what the destination coordinates should be.

When the world view map is displayed, rapid transit is of the local variety. The subroutine tracks the position of the tablet puck and displays it as a cursor on the world view map. The location of the cursor determines where in the Ispace the destination is. Activating the "doit"

button, causes the stager to stage data for the desired location. Once the data is staged, the main display is updated with the new information, and the the highlighted area of the world view map is redrawn to reflect the new location.

When the hierarchy map is displayed, rapid transit is of the inter-space variety. The subroutine sends a command to the hierarchy map process telling it to invoke rapid transit. The details of the implementation of rapid transit in the hierarchy map process is the subject of the following section.

6.2.2 Rapid Transit in the Hierarchy Map

As in the local mode of rapid transit, a cursor appears on the navigational aid monitor. The hierarchy map process tracks the location of the tablet puck and displays it as a cursor on the map. When the user moves the cursor to an Ispace node on the map (a rectangular solid with the name of the Ispace displayed in it) and activates the "doit" button, rapid transit to that Ispace is performed. The center of the new Ispace on iplane 1 is chosen as the destination.

At any time the user may switch to the local mode of rapid transit by activating the map select button. This button controls which map (world view or hierarchy) is currently selected.

References

[HEROT et al.]
Herot, C.F.; Kramlich, D.; Carling, R.; Friedell, M.; and Farrell, J., "Quarterly Research and Development Technical Report, Spatial Data Management System", Computer Corporation of America, 575 Technology Square, Cambridge, Mass., 02139 (December 1978).

[HELD STONEBRAKER WONG]
Held, G.D.; Stonebraker, M.R.; Wong, E., "INGRES - A relational data base system", AFIPS Proceedings, Volume 44.

[OSSANNA]
Ossanna, Joseph F., "NROFF Users' Manual" in *Documents for Use with the UNIX Time-Sharing System*, Sixth Edition, Bell Telephone Laboratories, May, 1975.