ISSN 0280–5316
ISRN LUTFD2/TFRT--5467--SE

Scanned Image Processor for X11/Motif

Per Tunestål

Department of Automatic Control
Lund Institute of Technology
February 1993

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Master Thesis

Date of issue
February 1993

Document Number
ISRN LUTFD2/TFRT--5467--SE

Author(s)
Per Tunestål

Supervisor
Rolf Johansson and Jonas Fredenholm

Sponsoring organisation

Title and subtitle
Scanned Image Processor for X11/Motif.

Abstract

The aim of this master thesis is the development of a software package for showing and manipulating scanned images in a X11/Motif environment.

The main purpose of the software is digitizing of graphs scanned in with a black- and white scanner. The software shall support both manual and automatic identification of graphs and coordinate systems. In the automatic mode the program shall be able to compensate for small non-linearities in the coordinate system as a consequence of for example photo copying. The task of automatic identification is complicated by the relatively high amount of noise present in the scanned images, and the fact that the curves are several pixels thick.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
45

Recipient's notes

Security classification

Acknowledgments

I would like to express my gratitude to Rolf Johansson and Jonas Fredenholm for making this master thesis possible, and for hints and advice during the work.

2. Task

2.1. Background

Flygprestanda AB is a company that performs services for civil airline companies as well as national air forces. The services mainly concern flight performance support and route planning.

The aircraft manufacturers supply diagrams specifying the performance, e.g. engine power, of the aircraft depending on various conditions, e.g. air temperature, air pressure and speed. Both safety and economy can be improved by taking full advantage of this information. Most small and middle size companies however lack the resources and knowledge to do so. This is where Flygprestanda AB enters the picture.

The route planning problem concerns choosing the best way from airport A to airport B. It also involves the procedures when departing from and arriving to an airport. Today these procedures are created manually, but with a new computer program using a terrain data base of the whole world, it will be possible to create these procedures on screen. For this reason there will be a need to digitise maps with altitude curves.

Today all diagrams are digitised using a conventional digitise board and a computer program that stores the digitised points and creates a textfile containing the digitised coordinates. The textfile is of a format recognised by optimising and other calculating programs developed by Flygprestanda AB. This arrangement is not satisfactory for the following reasons.

- The result is highly dependent on how well the user places the points and there is no easy way for the user to compare his work with the original diagram.
- Often diagrams are distorted due to photo copying. There is no way to deal with this in the present solution.
- The procedure is quite time consuming, because every point has to be manually digitised.
- If the diagram is accidentally moved during the session, the remaining points will not be correctly placed.

There is thus a need for better computer support.

The solution to these problems is a computer program where the user works on screen with scanned images of the diagrams. The digitised points could then be displayed on top of the image, and the user could easily check if the points are placed accurately. A magnifying window would further aid the user in placing the points. With the image in the computer's memory the program could also perform image processing for extraction of coordinate system and data.

The purpose of this master thesis is to develop a computer program which implements the above stated functions according to the specification in the following section.

2.2. Specification

The thesis consists of two parts:

- A software package for showing and manipulating scanned images in a X11/Motif environment.

-Documentation.

1. Software description:

The software shall be written in C and use the X11 Xtoolkit.

The following functions shall be implemented:

a) Widget showing a bitmap/pixmap containing the scanned image.

The widget shall be able to

- show a pix/bitmap.

- identify a coordinate system using mouse input.

- select and save points using mouse and a selected coordinate system.

- select filtered data lines and convert lines to an array of points.

- overlay source data (the original pixmap) and retrieved data to check consistency.

b) scrollbar widget for showing a selected part of the child widget.

c) A magnifying widget showing a selected part of a widget.

d) A filter function for extraction of data of a scanned graph. The filter shall be able to separate the coordinate system lines, and the data lines in a graph.

e) A program using the above described functions making a tool for looking at scanned images, and processing data contained in the scanned images.

2. Documentation.

The documentation shall be written in English, and contain a description of algorithms used for processing the image, as well as a description of the widget functions and relationships.

3. The Motif system

3.1. Introduction

Building applications with a graphical interface using pure C programming with a library of low level graphics primitives is difficult. The Motif system is designed to simplify this by providing predefined elements for user interface, such as scroll bars, push buttons, dialog boxes and menus. In fact the Motif system provides a way of object oriented programming with standard C. The objects used in Motif are called widgets which is short for "windowed gadget".

3.2. Widgets

A widget is a programming object which has a window on the screen at its disposal. Each widget is a member of a class. A number of widget classes are predefined by the Motif system. The elements for user interface mentioned above are all examples of predefined widget classes. An application programmer can use widget instances from these classes but can also write his own widgets as subclasses of these predefined classes.

3.3. Widget characteristics

To the application programmer who uses predefined widget classes a widget can be thought of as a "black box" with inputs, outputs and states.

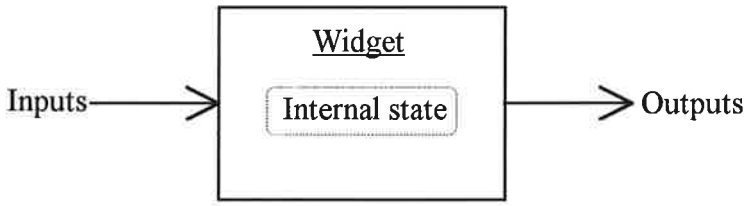


Figure 2.1: Black box visualization of a widget.

The inputs and outputs are called the widget resources and are the public parts of the widget's data structure. The internal state is the part of the widget's data structure that is private to the widget.

In a widget the inputs and outputs are called widget resources and can be set and read by the application that uses the widget. A special kind of resource is the "callback". Most widget classes has one or more callback resources which let the application declare callback functions that will be called as a consequence of some condition being fulfilled. This condition is often a mouse button or a key being pressed. In the callback functions the application can communicate with widgets by setting or getting resources or by calling functions exported by the widget.

3.4. Widgets in applications

In an application widgets are declared as variables of type Widget, and are placed in a widget tree, where the geometry of each child is managed by its parent. The children of a RowColumn

widget for example will be placed in rows or columns according to what is specified in the resources.

3.5. Widget methods

The active part of a Motif application is an event sensitive endless loop where events are dispatched to the various widgets. If you for example click mouse button 1 in a PushButton widget a mouse button event will be dispatched to the PushButton widget, which in turn will change its appearance and call its Activate Callback.

Some events have a special status, and the widget functions that are called as a consequence of these events are called the widget methods. Some of the most important widget methods are:

initialize: Called when the widget is created by the application. Initializes private state variables in accordance with resource values.

expose: Called when part of or the entire widget gets exposed after having been obscured. Restores the widget's appearance.

resize: Called when the widget is being resized. Recalculates the widgets graphical appearance based on the new size.

set_values: Called when a widget resource is set by the application. Recalculates private state variables based on the new resource values.

destroy: Called when the widget is destroyed by the application. Frees dynamically allocated memory space.

In addition to these predefined methods a widget can define action routines which will be called following a specified event sequence.

References

More on X11 and Motif can be found in these books.

Nye, A., and O'Reilly, T., 1992. "X Toolkit Intrinsics, Programming Manual". O'Reilly & Associates, Inc. ISBN: 0-937175-62-5

Open Soft Foundation, 1991. "OSF/Motif Programmer's Guide". Prentice Hall. ISBN: 0-13-640673-4

Nye, A., 1991. "Xlib Programming Manual". O'Reilly & Associates, Inc. ISBN: 0-937175-11-0

4. Image filtering

4.1. Introduction

There are some major difficulties when trying to filter out data lines of mathematical functions from a scanned bitmap image.

1. The data lines are not of zero width.
2. The coordinate system lines intersect the data lines.
3. There is noise present in the image.

In section 6.3. some solutions to these problems will be presented.

4.2. Image data

The source data to perform filtering on is stored as a character array in a structure of the type XImage, defined in XLib. Xlib supplies a number of functions as an interface to this structure. The functions provided for setting and getting pixels are:

```
unsigned long XGetPixel(ximage, x, y)
    XImage      *ximage;
    int         x;
    int         y;
```

The XGetPixel function returnes the pixel value of the pixel with coordinates (x, y) relative to the origin (upper left of the image). In case of a binary image it returns 1 (set) or 0 (not set).

```
int XPutPixel(ximage, x, y, pixel)
    XImage      *ximage;
    int         x;
    int         y;
    unsigned long pixel;
```

The XPutPixel function sets the value of the (x, y) pixel to *pixel*.

However, these functions are far too slow to use in the filtering algorithms presented below. For that reason some specialized functions which operate directly on the image data were designed.

4.3. Filtering Algorithms

4.3.1. Inverting algorithm

Often scanned images are inverted, that is data lines are 0:s and the background consists of 1:s. For this reason a function

```
void GraphInvert(w)
Widget      w;
```

is exported. This function goes through the original and the filtered bitmaps bytewise and inverts each byte.

4.3.2. Erasure of Solitudes

A function

```
void GraphSolitudes(w)
Widget      w;
```

that deletes all lonely points is exported. This function scans the entire image and deletes points which have all neighbours unset.

4.3.3. Thinning algorithm

The problem of non zero line width can be solved by filtering the image with a thinning algorithm. The object of the thinning algorithm is to reduce all curves in an image so that the filtered image will contain the medial paths of all the curves in the original image. It is however very expensive to actually compute the medial paths of all curves. For this reason an iterative method that deletes edge points on the curves will be used. The method used was developed by Zhang and Suen in 1984 and is described in "Digital Image Processing". The algorithm is implemented in the exported function

```
void GraphThin(w)
Widget      w;
```

Each iteration of the algorithm consists of two steps, where in the first step a point is flagged for deletion if the following conditions are satisfied (see Fig. 4.1).

(a) $2 \le N(p_1) \le 6$,

(b) $S(p_1) = 1$,

(c) $p_2 \cdot p_4 \cdot p_6 = 0$,

(d) $p_4 \cdot p_6 \cdot p_8 = 0$,

where $N(p_1)$ is the number of nonzero neighbors of p_1; that is,

$$N(p_1) = p_2 + p_3 + \ldots + p_8 + p_9$$

and $S(p_1)$ is the number of 0-1 transitions in the ordered sequence of $p_2, p_3, \ldots, p_8, p_9, p_2$. When all of the bitmap has been scanned according to the conditions above, the flagged points are deleted. In the second step, conditions (a) and (b) remain the same, but conditions (c) and (d) are changed to

(c') $p_2 \cdot p_4 \cdot p_6 = 0$,

(d') $p_4 \cdot p_6 \cdot p_8 = 0$,

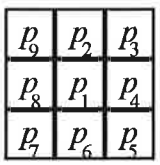


Fig. 4.1: Neighborhood arrangement used by the thinning algorithm.

Points meeting the conditions (a), (b), (c'), (d') are flagged and deleted in the same manner as in step one. This procedure is iterated until there are no points left to delete.

The left part of condition (a) is to prevent endpoints to be erased, and the right part is to prevent erosion into corners. Condition (b) prevents cutting off one pixel thick lines. Conditions (c), (d), (c'), (d') assures that regions are thinned symmetrically. Step one deletes to the upper left and step two to the lower right.

The thinning algorithm has been tested on various images and performs well. One problem however, arises in the vicinity of multiple line intersections, where severe deformation can take place.

4.3.4. Cross- and endfinding algorithm

An algorithm that finds all endpoints and points where curves intersect, and puts them in a list is implemented in the function

```
void GraphCrossings(w)
Widget      w;
```

This function is intended to be used after the thinning algorithm has been applied to the bitmap. A pixel is put in the list if it is set and its surrounding according to Fig. 4.1 meets either of the conditions

(a) $S(p_1) \geq 3$, or

(b) $p_2 = p_3 = p_4 = 0$, or

$p_4 = p_5 = p_6 = 0$, or

$p_6 = p_7 = p_8 = 0$, or

$p_8 = p_9 = p_2 = 0$

(c) $S(p_1) = 1$, $N(p_1) \leq 2$

Condition (a) handles normal 3- and 4-way crossings, and condition (b) handles crossings of the type in Fig. 4.2.

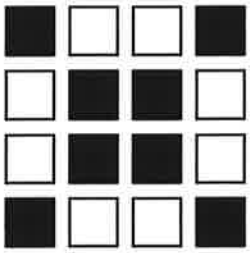


Fig. 4.2: Crossing handled by condition (b).

All four points of the cluster in the middle are stored as crossings. Condition (c) matches all endpoints.

The structure of the elements in the list is

```
typedef struct node_struct {
    short               x, y;
    struct node_struct  *connections[8];
    char                *path[8];
    char                rev[8];
    XPoint              midpoints[8];
    XPoint              *vect[8];
    short               *nvects[8];
    int                 type;
}   nodetype;
```

where XPoint is a data type defined by Xlib according to

```
typedef {
```

```
    short    x, y;
} XPoint;
```

The member type can have one of the values

```
NORMAL_CROSS
COO_CROSS
END_POINT
```

The function GraphCrossings assignes the value NORMAL_CROSS to intersections, and END_POINT to endpoints. The function also initiates all the other members.

When testing the algorithm, the conditions (a) - (c) above have proven to constitute a correct model of intersections and endpoints.

4.3.5. Algorithm for identifying coordinate intersections

The exported function

```
void GraphCoos(w)
Widget    w;
```

checks all crossings found with the cross finding function and tries to match an orthogonal cross of user specified size in the original bitmap. If there is a match, the node is marked as COO_CROSS.

4.3.6. Algorithm for adjusting the coordinate system

If the user of the program has specified a coordinate system manually, the exported function

```
void GraphCreateCooSystem(w)
Widget    w;
```

can adjust the coordinate system according to the coordinate intersections identified with the function GraphCoos. The function scans the intersections in the entered system and checks if there is an identified intersection close enough and, if so, uses the identified intersection instead.

4.3.7. Algorithm for finding connections

An algorithm that traverses all paths between all endpoints and crossings found with the cross finding function, and establishes connections is implemented in the exported function

```
void GraphFindConnections(w)
Widget    w;
```

What the function does is, leaving out technical details

```
for (all elements in the list of crossings and endpoints) {
    for (all paths emerging from the node that are not already traversed)
        follow, and store the path until a crossing or endpoint is reached;
        establish a bidirectional connection between the nodes;
    }
}
```

The arrays path, connections, rev of nodetype above all refer to connections between the node and other nodes. The pointer array, **connections**, contains pointers to the connected nodes. The array of character strings, **path**, holds the information about the paths. The array of

characters, **rev**, tells for each connection the array index of the same connection in the other direction.

When following a path, coded values of the direction of travel are stored in a character string, which when the path is complete is stored in the array path. This saves both time and memory space compared to storing the coordinates in a list. The directions are coded with the binary numbers below.

RIGHT	00000001
DOWNRIGHT	00000010
DOWN	00000100
DOWNLEFT	00001000
LEFT	00010000
UPLEFT	00100000
UP	01000000
UPRIGHT	10000000

When following paths it is important to assure that there is no more than one way to do so. It is also essential to avoid any possibility to get caught in a loop. If the algorithm avoids searching in directions that are partly or entirely backwards (se Fig. 4.3), the possibility of getting caught disappears.

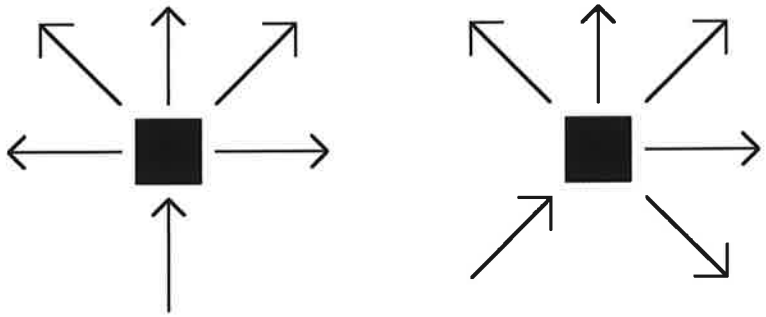


Fig. 4.3: Permissible directions of search.

One problem with the result produced by the thinning algorithm is illustrated in fig. 4.4. The lower pixel has two neighbours in the forward direction. The way around this problem is to let horizontal and vertical directions have higher priority than diagonal directions.

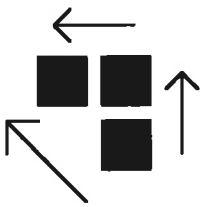


Fig. 4.4: Path following algorithm having to make a choice between two directions.

From fig. 4.5 it follows that there can be no more than eight paths coming from a single crossing. Actually there can't be more than four but the simplicity of indexing arrays led to a data structure with an array of eight possible connections from each crossing or endpoint. The numbers shown in Fig. 4.5 are used to index the arrays concerning the connections. Consequently the index of a connection will in general have different indices in different directions. This is why the array rev is needed.

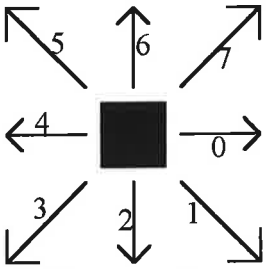


Fig. 4.5: Possible path directions from crossing or endpoint.

For future use the coordinates of the midpoints on every path are stored in the array **midpoints**.

The path finding algorithm has proven to perform as expected when applying to thinned images.

4.3.8. Algorithm for vectorizing the image

An algorithm that converts all connections between nodes to vector sequences is implemented in the exported function

```
void GraphVectorize(w)
Widget      w;
```

The idea of vectorizing is illustrated in Fig. 4.6.

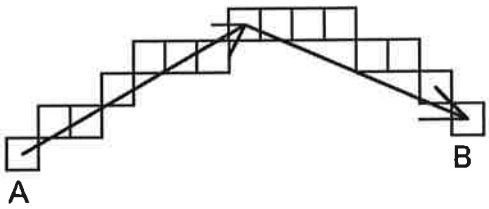


Fig. 4.6: Vectorization of a path.

The path from node A to node B is vectorized with a vector length of 7.This is done for all connections found with GraphFindConnections, and the result is stored in the array vect in the node structure. Each element of vect can hold an array of vectors. The elements of the array nvects tells how many vectors there are in each connection.

A rough description of the algorithm is, with vector length = length

```
for (all nodes) {
    for (all paths from the node) {
        while (not end of path) {
            Create vector from length steps of the path;
            Put vector in array;
        }
        if (more than one vector, and last vector is shorter than length) {
            Add this vector to the second last vector;
        }
        Put vector array in node structure;
    }
}
```

The vectorization smothes the connecting paths, which is useful when searching through the network trying to identify an entire curve.

4.3.9. Erasure of Coordinate Lines

The exported function

```
void GraphEraseCooLines(w)
Widget      w;
```

goes through all connecting paths and checks if they keep within a user specified distance from a coordinate line. If so, the algorithm deletes the path and breaks the connection. The purpose of doing this is mainly to help the auto searching routine in making the right decisions.

4.3.10. Erasure of Dead Ends

The exported function

```
void GraphEraseDeadEnds(w)
Widget      w;
```

checks the lengths of paths to or from endpoints and deletes paths with a length shorter than a user specified value.

4.3.11. Following Curves

The function

```
void LineSearch(gw)
GraphWidget  gw;
```

tries to digitize a whole curve starting with a selected node and connection. The function starts at the selected connection and compares the last vector with the first vectors in the neighboour connections. The function chooses the connection with the least difference in direction if the difference is small enough, and continues from there (see Fig. 4.7). This is done in both directions from the selected connection. All vector starting points are stored in the current data line.

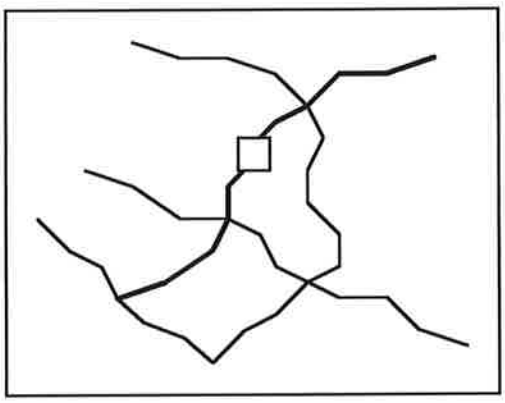


Fig. 4.7: Selected connection and the connections chosen by the algorithm.

A problem arises when an intersection of many lines has been separated by the thinning algorithm into two or more nodes. This is handled by the algorithm by checking directions of vectors from neighbour nodes as well if they are close enough (see Fig. 4.8).

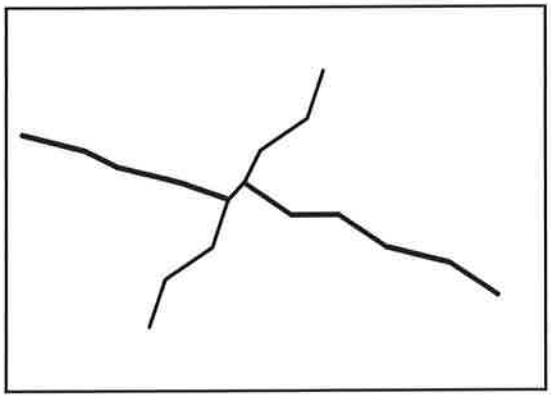


Fig. 4.8: Separated intersection where the algorithm "looks one step ahead".

To assure that the algorithm doesn't get caught in an endless loop every connection passed is marked, and if this connection is reached again it cannot be chosen.

References

More on the thinning algorithm can be found in section 8.1.5. of

Gonzales, R. C., and Wintz, P., 1987. "Digital Image Processing". Addison Wesley. ISBN: 0-201-11026-1

5. Coordinate Handling

5.1. Introduction

When an image is scanned there will always be deformations in parts of the image. These deformations can take place for example if the paper has been folded. For this reason the program must be able to handle slightly non-linear coordinate systems.

5.2. Method

The method used in the application is to place all coordinate intersections in a matrix, and have a local coordinate system in each cell in the matrix (see Fig. 5.1).

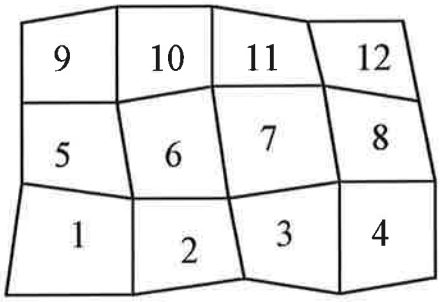


Fig. 5.1: Matrix of coordinate intersections, where each numbered cell has its own local coordinate system.

The real coordinates of an arbitrary point on screen can now be calculated from the coordinates of the downleftmost corner of the cell containing the point, and the local coordinates in the cell. The question is however what method to use for calculation of the local coordinates.

5.3. Theory

Fig. 5.2 shows one cell in the matrix of coordinate intersections. Let the surface be parameterized by s, t, and let these parameters be the "local coordinates" of the cell. Let $(x_1(s), y_1(s))$ denote the screen coordinates of the points on line 1, and $(x_3(s), y_3(s))$ the screen coordinates of the points on line 3.

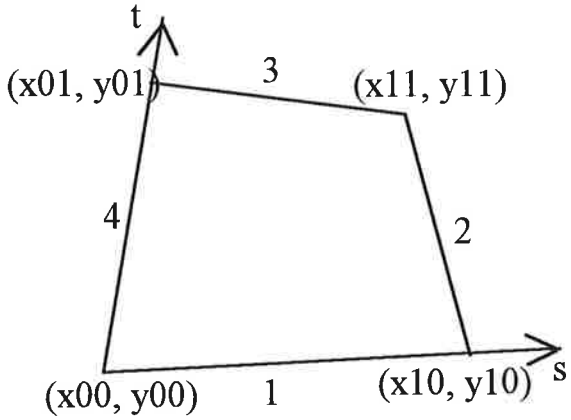


Fig. 5.2: A cell in the coordinate matrix. Marked coordinates are the screen coordinates of the corners.

It follows from the image that

$$x_1(s) = x_{00} + s*(x_{10} - x_{00}); \qquad y_1(s) = y_{00} + s*(y_{10} - y_{00})$$
$$x_3(s) = x_{01} + s*(x_{11} - x_{01}); \qquad y_3(s) = y_{01} + s*(y_{11} - y_{01})$$

For an arbitrary point in the cell with screen coordinates (x(s, t), y(s, t)) it follows that

$$x(s, t) = x_1(s) + t*(x_3(s) - x_1(s)) \quad ; \quad y(s, t) = y_1(s) + t*(y_3(s) - y_1(s))$$

Simple calculations give

$$x(s, t) = x_{00} + s*(x_{10} - x_{00}) + t*(x_{01} - x_{00}) + s*t*(x_{11} + x_{00} - x_{01} - x_{10})$$
$$y(s, t) = y_{00} + s*(y_{10} - y_{00}) + t*(y_{01} - y_{00}) + s*t*(y_{11} + y_{00} - y_{01} - y_{10})$$

This can be written

$$x(s, t) = a + b*s + c*t + d*s*t \; ; \qquad y(s, t) = e + f*s + g*t + h*s*t \quad (5.1)$$

with

$$a = x_{00} \; ; \quad e = y_{00}$$
$$b = x_{10} - x_{00} \; ; \quad f = y_{10} - y_{00}$$
$$c = x_{01} - x_{00} \; ; \quad g = y_{01} - y_{00}$$
$$d = x_{11} + x_{00} - x_{01} - x_{10} \; ; \quad h = y_{11} + y_{00} - y_{01} - y_{10}$$

We would like to invert (5.1) in order to get a formula to calculate (s, t) from (x, y), but the last terms are non-linear and makes it difficult. If however the coordinate system is close to linear we can neglect these terms and calculate an approximation (s', t') of (s, t).

$$\begin{bmatrix} s' \\ t' \end{bmatrix} = \begin{bmatrix} g & -c \\ -f & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

The error made in the approximation can now be calculated by applying (5.1) on (s', t') and get (x', y'). if the difference between (x', y') and (x, y) is too big a correction can be made.

$$s \approx s' - \frac{x' - x}{b} \qquad t \approx t' - \frac{y' - y}{g}$$

In the program this correction is iterated until $x' - x < 0.5$ and $y' - y < 0.5$.

6. Graph Widget

6.1. Introduction

The Graph widget is the heart of the application. Its purpose is to provide a tool for displaying and digitizing a scanned image read from a file. The widget can operate in one of three modes:

- Digitization of mathematical functions from x to y.
- Digitization of mathematical functions from y to x.
- Digitization of general continuous curves in two dimensions.

The Graph widget supports manual digitization of coordinate intersections and data lines, but also exports some filter functions for interactive semi automatic digitization of coordinate intersections as well as data lines.

6.2. Widget Class

The class name for the Graph widget is **GraphWidgetClass**, and to use an instance of the Graph widget in an application the following include statement must be at the top of the application.

```
#include "Graph.h"
```

6.3. Resources

Following resources are defined by the Graph widget.

Name	Class	Type	Default
XfpNcallback	XfpCCallback	XmRCallback	NULL
XfpNforeground	XfpCForeground	XmRPixel	Black
XfpNbackground	XfpCBackground	XmRPixel	White
XfpNpointColour	XfpCPointColour	XmRPixel	Blue
XfpNcurrentColour	XfpCCurrentColour	XmRPixel	Red
XfpNcooColour	XfpCCooColour	XmRPixel	Green
XfpNdataFile	XfpCDataFile	XmRString	NULL
XfpNoutFile	XfpCOutFile	XmRString	NULL
XfpNimage	XfpCImage	XmRPointer	NULL
XfpNclickMode	XfpCClickMode	XmRInt	0
XfpNxCoo	XfpCXCoo	XmRPointer	NULL
XfpNyCoo	XfpCYCoo	XmRPointer	NULL
XfpNxoutCoo	XfpCXoutCoo	XmRPointer	NULL
XfpNyoutCoo	XfpCYoutCoo	XmRPointer	NULL

XfpNcurrentF	XfpCCurrentF	XmRPointer	NULL
XfpNshowConnections	XfpCShowConnections	XmRBoolean	False
XfpNdisplayMode	XfpCDisplayMode	XmRInt	0
XfpNcrossSize	XfpCCrossSize	XmRInt	10
XfpNvectorLength	XfpCVectorLength	XmRInt	10
XfpNminPath	XfpCMinPath	XmRInt	30
XfpNxSpacing	XfpCXSpacing	XmRPointer	NULL
XfpNySpacing	XfpCYSpacing	XmRPointer	NULL
XfpNmode	XfpCMode	XmRInt	X_Y
XfpNsearchMemory	XfpCSearchMemory	XmRInt	50
XfpNregionSize	XfpCRegionSize	XmRInt	10
XfpNendLength	XfpCEndLength	XmRInt	10
XfpNmidpointLimit	XfpCMidpointLimit	XmRInt	3

XfpNcallback:
Specifies the callbacks to be called when the widget performs XtCallCallback.

XfpNforeground:
The foreground colour of the bitmap.

XfpNbackground:
The background colour of the bitmap.

XfpNpointColour:
The colour of digitized data points not belonging to the current data line.

XfpNcurrentColour:
The colour of digitized data points belonging to the current data line.

XfpNcooColour:
The colour of digitized coordinate line intersections.

XfpNdataFile:
The name string of the bitmap source file.

XfpNoutFile:
The name string of the outfile containing the coordinates of the digitized data points.

XfpNimage:
A pointer to the XImage structure containing the bitmap currently displayed on screen.

XfpNclickMode:
Specifies the mouse input mode.

COO_MODE =	Manual input of coordinate intersections
POINT_MODE =	Manual input of data points. When changing to this mode, a new data line will automatically be created.
SEMI_MODE =	Automatic digitization of selected connection. When changing to this mode, a new data line will automatically be created.

AUTO_MODE = Automatic digitization of entire data line. When changing to this mode, a new data line will automatically be created.

XfpNxCoo and **XfpNyCoo:**
Used to get user input of coordinates from application program.

XfpNxoutCoo and **XfpNyoutCoo:**
Used to transfer current coordinates of intersection to application program.

XfpNcurrentF:
Holds the f-value of the current data line.

XfpNshowConnections:
Specifies whether to show connections between data points or not.

XfpNdisplayMode:
Specifies what to show on screen.

NORMAL_DISP = Show original bitmap
FILTERED_MODE = Show filtered bitmap
VECTORS_DISP = Show vectorized image

XfpNcrossSize:
Specifies the cross size to use when identifying coordinate intersections with GraphCoos.

XfpNvectorLength:
Specifies the vector length used when vectorizing the image with GraphVectorize.

XfpNminPath:
Minimum length of a path to be erased by GraphEraseCooLines.

XfpNxSpacing and **XpfNySpacing:**
Used to transfer spacing of the coordinate system between widget and application.

XfpNmode:
Sorting mode.

X_Y = Sort data points as function from x to y
Y_X = Sort data points as function from y to x
CONTINUOUS = Do not sort data points

XfpNsearchMemory:
Specifies how much to weigh in old directions when identifying data lines.

XfpNregionSize:
Size of click sensitive regions.

XfpNendLength:
Maximum length of dead ends to be erased by GraphEraseDeadEnds.

XfpNminpointLimit:
Minimum length of connections to be click sensitive.

6.4. Functionality

6.4.1. Widget Methods

The following methods are implemented in the Graph widget:

Initialize:
The Initialize method loads graphics contexts used when drawing on screen, checks resource values set by the application, sets private state variables and loads initial bitmap if one is specified.

Redisplay:
Checks if the widget is realized and if so redraws exposed part of the bitmap. The method also redraws data points, coordinate intersections, coordinate lines and vectorized image, all according to current display mode.

SetValues:
Handles resource settings from the application.

Destroy:
Deallocates all memory space dynamically allocated by the widget.

6.4.2. Widget Actions

There are four action functions defined in the Graph widget. What action the action functions will perform depends on the resource XfpNclickMode.

StartAdd:
StartAdd is called when mouse button 1 is pressed.

If the value of XfpNclickMode is

- COO_MODE: If the number of coordinate intersections is less than 3, a new element will be placed in the list of coordinate intersections. If the number equals 3, the nearest node in the coordinate matrix will be moved to the position of the mouse pointer.

- POINT_MODE: A new data point will be added to the current data line.

- SEMI_MODE or AUTO_MODE: If the image has been vectorized, the nearest connection will be selected.

StartMove:
StartMove is called when mouse button 2 is pressed.

If the value of XfpNclickMode is:

- COO_MODE: The nearest element in the list of coordinate intersections will be selected for moving.

- POINT_MODE: The nearest element in the structure containing data points will be selected for moving. The data line containing this element will be the current data line.

- SEMI_MODE or AUTO_MODE: Same action as for button 1.

DoMove:
DoMove is called when the mouse is moved with button 1 or 2 pressed.

If the value of XfpNclickMode is:

- COO_MODE: The selected element in the list of coordinate intersections will be moved according to pointer motion.

- POINT_MODE: The selected element in the structure of data points will be moved according to pointer motion.

- SEMI_MODE or AUTO_MODE: No action is performed.

FinishMove:

FinishMove is called when mouse button 1 or 2 is released.

If the value of XfpNclickMode is:

- COO_MODE: The selected element in the list of coordinate intersections will be fixed in the current pointer position.

- POINT_MODE: The selected element in the structure of data points will be fixed in the current pointer position.

- SEMI_MODE: The vectors belonging to the selected connection will be transformed into data points in the current data line.

- AUTO_MODE: The function LineSearch will be called and all the vectors of an entire curve will be transformed into data points in the current data line.

DeletePoint:
DeletePoint is called when mouse button 3 is clicked (pressed, released).

If the value of XfpNclickMode is:

- COO_MODE: The nearest element in the list of coordinate intersections will be deleted.

- POINT_MODE: The nearest element in the structure of data points will be deleted.

- SEMI_MODE or AUTO_MODE: If the image has been vectorized the nearest connection will be deleted.

6.4.3. Callbacks

The Graph widget defines one callback with the name XfpNcallback. An element of the structure

```
typedef struct _GraphCallbackStruct {
        int             reason;
        XButtonEvent *event;
}       GraphCallbackStruct;
```

is supplied with each callback. The field reason can take the values

- ADD_DATA:	Called when a data point has been placed.
- MOVE_DATA:	Called when a data point has been moved.
- ADD_COO:	Called when a coordinate point has been placed, telling the application to specify coordinates.
- MOVE_COO:	Called when a coordinate point has been moved, telling the application to specify coordinates.
- PLACING:	Called whenever the mouse is being moved with button 1 or 2 pressed.
- CANCEL:	Called when buttons 1 or 2 has been clicked where there is no point.
- IMAGE_CHANGED:	Called when a new bitmap has been loaded.
- DEFINE_SPACING:	Called when a coordinate matrix is to be created, telling the application to specify the spacing between coordinate lines.
- NEW_LINE:	Called when a new data line has been created from within the widget, telling the application to specify a new f-value.

When the callback is caused by a button event, the event field contains the event.

6.5. Exported Functions

void **GraphCoos**(w)
Widget w;

Image filtering function described in sect. 4.

void **GraphCreateNewCurve**(w, f)
Widget w;
float f;

If a no data line with f-number f exists, the function creates one. Otherwise the data line with f-number f is made current data line.

void **GraphCrossings**(w)
Widget w;

Image filtering function described in sect. 4.

void **GraphEraseCooLines**(w)
Widget w;

Image filtering function described in sect. 4.

void **GraphEraseCurve**(w)
Widget w;

Erases current data line.

void **GraphEraseDeadEnds**(w)
Widget w;

Image filtering function described in sect. 4.

void **GraphFindConnections**(w)
Widget w;

Image filtering function described in sect. 4.

void **GraphInvert**(w)
Widget w;

Image filtering function described in sect. 4.

void **GraphLoadBitmap**(w, data_file, filt)
Widget w;
char *data_file;
Boolean filt;

Loads the bitmap stored in msp-file data_file. If filt is TRUE the bitmap is placed as filtered image, otherwise it is placed as original bitmap.

void **GraphSaveBitmap**(w, out_bitmap)
Widget w;
char *out_bitmap;

Saves the filtered image in the msp-file out_bitmap.

void **GraphSaveData**(w, out_file)
Widget w;
char *out_file;

Stores the digitized information in the text file out_file.

void **GraphThin**(w)
Widget w;

Image filtering function described in sect. 4.

6.6. Created Output

When the function GraphSaveData is called, a text file with the following syntax is created.

```
*       <name of text file without search path>
*       <name of text file with search path>
*       <name of the bitmap file with search path>

%<f-value>
<pairs of x- and y-coordinates of the points in the data line, sorted according to sort mode>
        .
        .
        .
%<f-value>
<pairs of x- and y-coordinates of the points in the data line, sorted according to sort mode>
*
```

The data lines are sorting in rising f-values.

Example:

```
*       graf.dta
*       /home/per/graf.dta
*       /home/per/graf.msp

%-1000.00
3001.29  2.83  3288.64  3.75  3544.46  4.50  3804.57  5.25
%0.00
2999.53  2.58  3298.16  3.50  3466.89  4.00  3646.39  4.50
%2000.00
3002.07  2.04  3315.53  3.01  3483.79  3.50  3662.19  4.00
*
```

6.7. Summary

The graph widget implements most of what is needed to digitize functional graphs as well as for example maps with altitude curves. The automatic path finding algorithm however lacks the ability to find closed curves because they have no endpoints and no intersections. The ability to find closed curves would be convenient when digitizing altitude curves.

References

Nye, A., and O'Reilly, T., 1992. "X Toolkit Intrinsics, Programming Manual". O'Reilly & Associates, Inc. ISBN: 0-937175-62-5

Open Soft Foundation, 1991. "OSF/Motif Programmer's Guide". Prentice Hall. ISBN: 0-13-640673-4

Nye, A., 1991. "Xlib Programming Manual". O'Reilly & Associates, Inc. ISBN: 0-937175-11-0

7. Magnify Widget

7.1. Introduction

The purpose of the Magnify widget is to provide a magnification of a bitmap picture by some integer factor. In the application this will be used as an aid in placing data points and coordinate intersections. In particular, when the mouse button is pressed to place a point the Magnify widget will display a surrounding of the actual point in the bitmap. This surrounding will continuously be updated until the mouse button is released. The window has a coloured frame surrounding the current point in the bitmap. This frame is allowed to move in a square, and the magnify window is scrolled first when the border of this square is reached.

7.2. Widget Class

The class name for the Magnify widget is **MagnifyWidgetClass**, and to use an instance of the Magnify widget in an application the following include statement must be at the top of the application.

```
#include "Magnify.h"
```

7.3. Resources

Following resources are defined by the Magnify widget.

Name	Class	Type	Default
XfpNforeground	XfpCForeground	XmRPixel	Black
XfpNbackground	XfpCBackground	XmRPixel	White
XfpNgridColour	XfpCGridColour	XmRPixel	Green
XfpNcenterFrameColour	XfpCCenterFrameColour	XmRPixel	Red
XfpNimage	XfpCImage	XmRPointer	NULL
XfpNcurX	XfpCCurX	XmRInt	300
XfpNcurY	XfpCCurY	XmRInt	300
XfpNwidthInPoints	XfpCWidthInPoints	XmRInt	20
XfpNheightInPoints	XfpCHeightInPoints	XmRInt	20
XfpNspaceBetweenPoints	XfpCSpaceBetweenPoints	XmRInt	2
XfpNwidthInPixels	XfpCWidthInPixels	XmRInt	8
XfpNheightInPixels	XfpCHeightInPixels	XmRInt	8

XfpNforeground: Colour of points that are set.

XfpNbackground: Colour of points that are reset.

XfpNgridColour: Colour of the grid between points.

XfpNcenterFrameColour: Colour of the frame surrounding the center point.

XfpNimage: Pointer to the bitmap image that is to be magnified.

XfpNcurX:	Current x-coordinate in the bitmap.
XfpNcurY:	Current y-coordinate in the bitmap.
XfpNwidthInPoints:	Width of the magnify window in magnified points.
XfpNheightInPoints:	Height of the magnify window in magnified points.
XfpNspaceBetweenPoints:	The space in pixels between the magnified points.
XfpNwidthInPixels:	Width in pixels of magnified points.
XfpNheightInPixels:	Height in pixels of magnified points.

7.4. Functionality

7.4.1. Widget Methods

The following methods are implemented in the Graph widget:

Initialize:
The Initialize method loads graphics contexts used when drawing on screen, checks resource values set by the application and sets private state variables.

Redisplay:
Checks if the widget is realized and if so redraws the magnified points.

SetValues:
Handles resource settings from the application, and if needed redraws the widget.

Destroy:
Deallocates all memory space dynamically allocated by the widget.

7.4.2. Widget Actions

No actions are defined in the Magnify widget.

7.4.3. Callbacks

No callbacks are defined for the Magnify widget.

7.5. Exported Functions

The Magnify widget exports one function.

```
void MagnifyResetOffset(w)
Widget    w;
```

7.6. Summary

The magnify widget is a very simple tool, and it is designed that way on purpose. I didn't want to make a tool that was tailor made for this application, but could be used in any application where magnification of a bitmap is needed. I could have made a magnify widget with possibility to display for example the digitized data points or with possibilities to digitize by clicking in the magnify window. Such functions would however be of very limited use in a different application.

References

Nye, A., and O'Reilly, T., 1992. "X Toolkit Intrinsics, Programming Manual". O'Reilly & Associates, Inc. ISBN: 0-937175-62-5

Open Soft Foundation, 1991. "OSF/Motif Programmer's Guide". Prentice Hall. ISBN: 0-13-640673-4

Nye, A., 1991. "Xlib Programming Manual". O'Reilly & Associates, Inc. ISBN: 0-937175-11-0

8. Application

8.1. Introduction

The application code is the part of the program that ties everything together. It creates instances of the GraphWidgetClass and MagnifyWidgetClass and creates menus with which the user can interact with the program. Because the widgets may not and cannot communicate directly with each other, the application must handle all such communication through callback functions. The application code is like "the spider in the net". Fig. 8.1 shows what the application looks like on screen.

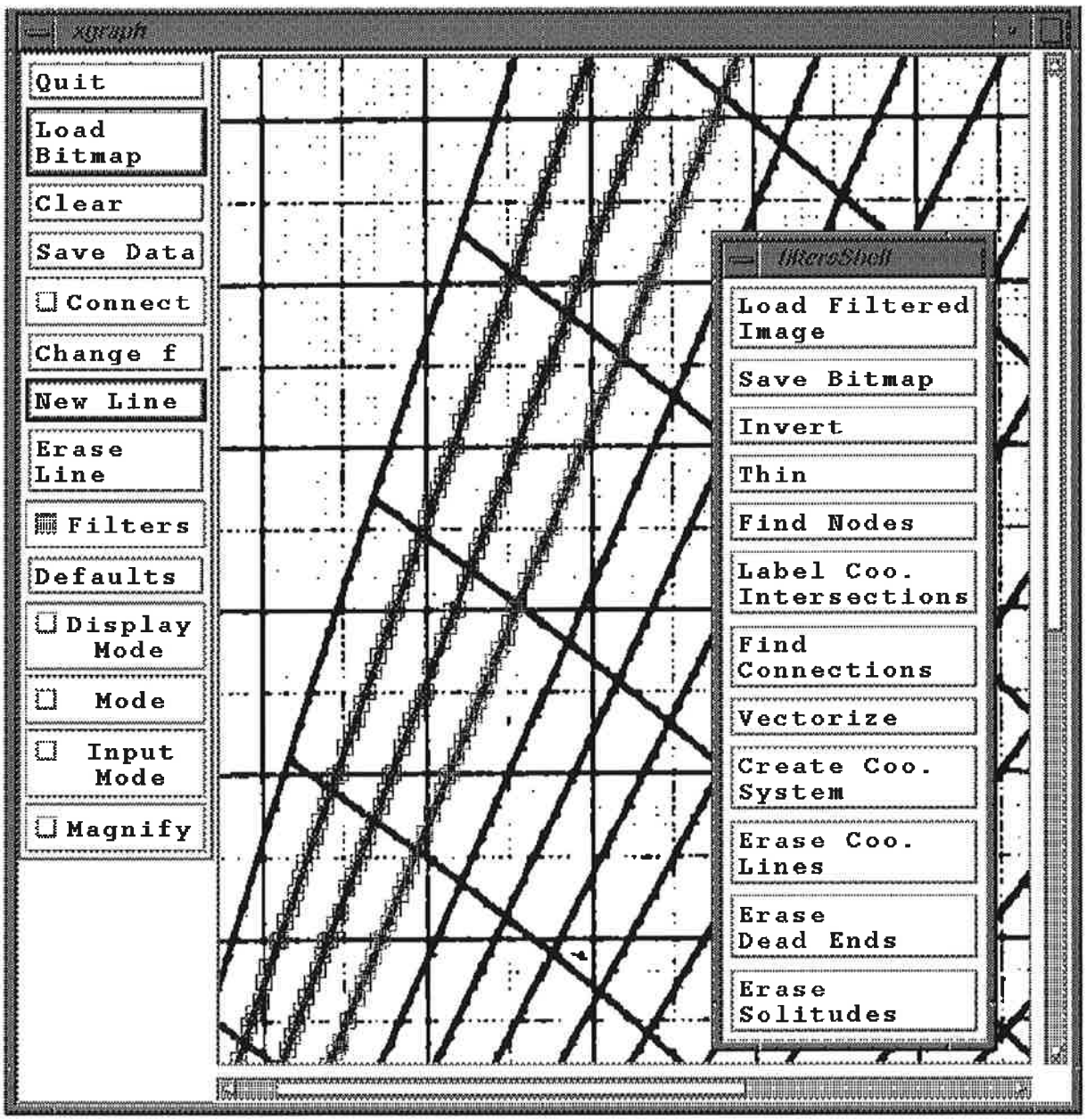


Fig. 8.1: Appearance on screen.

8.2. Widget Hierarchy

The widget hierarchy is illustrated in Fig. 8.1. Some widgets of minor importance for the overall picture have been left out. For example, all the menu boxes has a shell widget as parent. The shell widget supports its children with a window manager frame so that the menus can be resized and moved by the user.

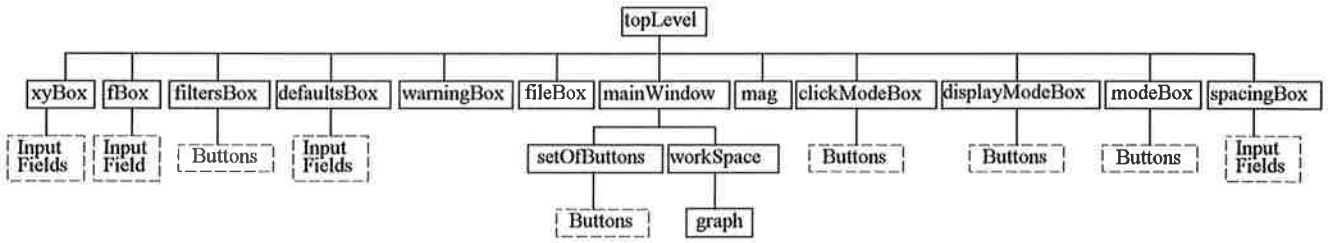


Fig. 8.2: Widget hierarchy.

8.3. Menus

All menus except for the main menu are menus that can be "popped up" by pressing the appropriate toggle button in the main menu. By pressing the same button again the menu will be "popped down".

8.3.1. Main Menu

The main menu is always visible and contains toggle buttons to "pop up" and "pop down" the sub menus. It also contains

- a push button for exiting the application.
- a push button for loading a bitmap.
- a push button for erasing all digitized information.
- a push button for saving digitized information in a text file.
- a push button for creating a new data line.
- a push button for erasing the current data line.
- a push button for changing the f-value of the current data line.
- a push button for "popping up" a input window for changing some default settings.
- a toggle button that controls whether connections are drawn between data points.
- a toggle button that controls whether the magnify window is visible or not.

8.3.2. Filters Menu

The filters menu is a box containing one push button for each of the filter functions described in sect. 4. There are also two push buttons for loading a prefiltered image and for saving a filtered image.

Each of the buttons has a callback function associated with it which, when called, will call the appropriate exported function in the graph widget.

8.3.3. Click Mode Menu

The click mode menu is a box containing four toggle buttons, one for each click mode, of which only one at a time can be active. The four click modes are

Coo. Mode	Mouse clicks are interpreted as coordinate intersections
Point Mode	Mouse clicks are interpreted as data points
Semi Mode	Mouse clicks will digitize the vectors of one connection
Auto Mode	Mouse clicks will digitize an entire data line

Each toggle button has a callback function associated with it that, when called, will set the click mode resource in the graph widget.

8.3.4. Display Mode Menu

The display mode menu is a box containing three toggle buttons, one for each display mode, of which only one at a time can be active. The three display modes are

Normal Bitmap	Show original bitmap
Filtered Bitmap	Show filtered bitmap
Vectorized Image	Show vectorized image

Each toggle button has a callback function associated with it that, when called, will set the display mode resource in the graph widget.

8.3.5. Mode Menu

The mode menu is a box containing three toggle buttons, one for each sorting mode, of which only one at a time can be active. The three sorting modes are

xy-Mode	Data points are sorted as functions from x to y
yx-Mode	Data points are sorted as functions from y to x
Continuous Mode	Data points are not sorted

Each toggle button has a callback function associated with it that, when called, will set the mode resource in the graph widget.

8.4. Other Widgets

8.4.1. Text Field

For user input of integer and float values a widget of the class xmTextWidgetClass is used. This is the case in the input boxes for default settings, x- and y-coordinates, coordinate line spacing, and f-values.

8.4.2. File Selection Box

When loading/saving files on disk the user specifies which file to load/save in a file selection box, which is a widget of the class xmFileSelectionBox.

References

Nye, A., and O'Reilly, T., 1992. "X Toolkit Intrinsics, Programming Manual". O'Reilly & Associates, Inc. ISBN: 0-937175-62-5

Open Soft Foundation, 1991. "OSF/Motif Programmer's Guide". Prentice Hall. ISBN: 0-13-640673-4

Nye, A., 1991. "Xlib Programming Manual". O'Reilly & Associates, Inc. ISBN: 0-937175-11-0

9. Conclusion

All the functions stated in sect. 3 have been implemented in the application. The user interface is somewhat primitive, and could with some effort be made much more sophisticated. With the limited time available for a master thesis I have used most of the effort on the actual performance of the application.

Flygprestanda AB
Malmö

X-Graph Digitizing tool

User's Guide

by Per Tunestål

Malmö, January 1993

1. Introduction

This is a short user's manual for the digitization tool XGraph. The application is a tool for digitizing scanned images of for example diagrams or maps. The program supports manual digitization by mouse clicking, but also supplies some filter functions for interactive automatic digitization. The program works with linear coordinate systems only, but can compensate for deformations, present as a consequense of for example photo copying.

The appearance of the application is presented in Fig. 1.1. It has a work space which holds the image, a main menu, and sub menus which can be popped up by clicking buttons in the main menu. Here the filters sub menu has been popped up.

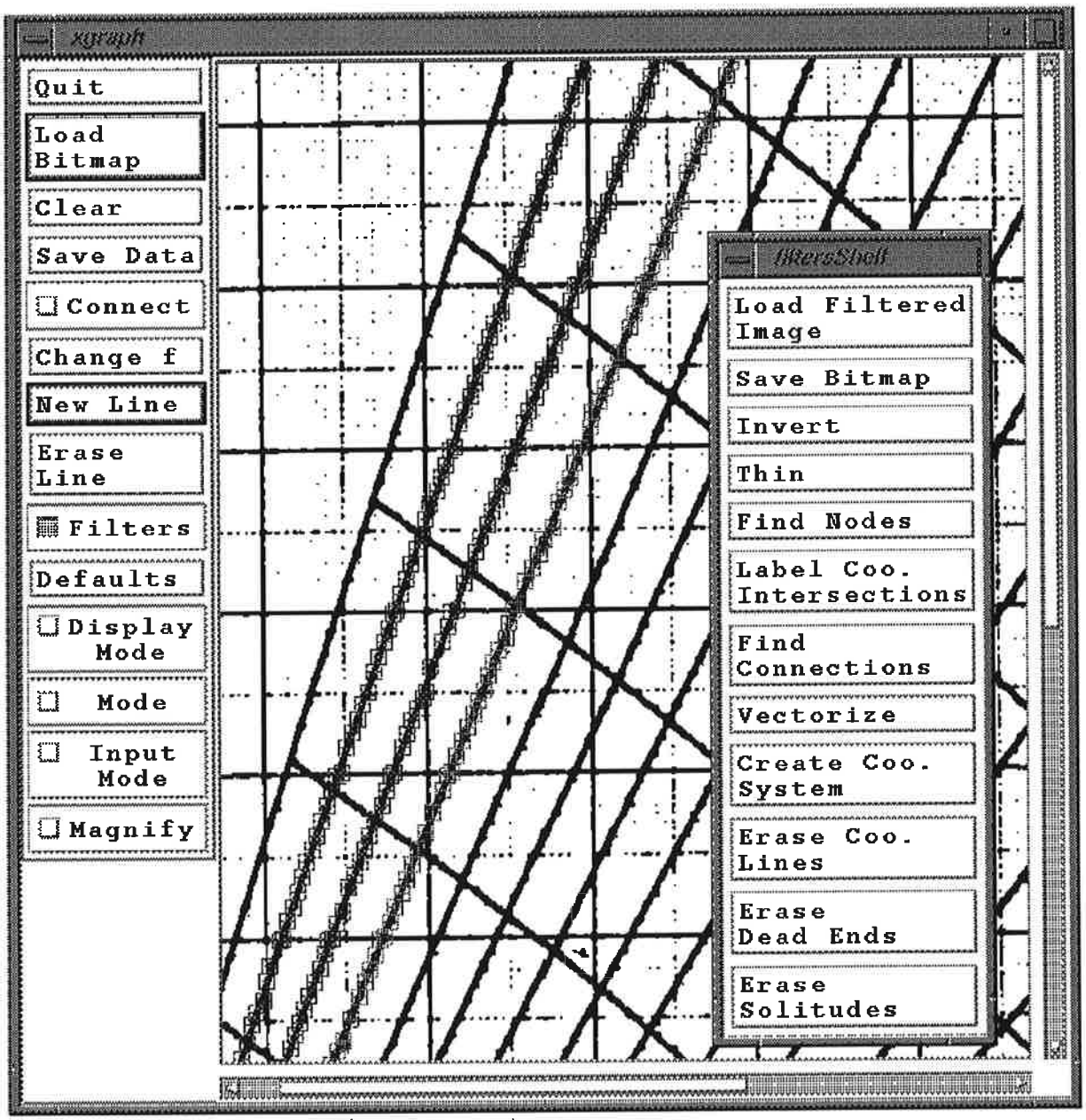


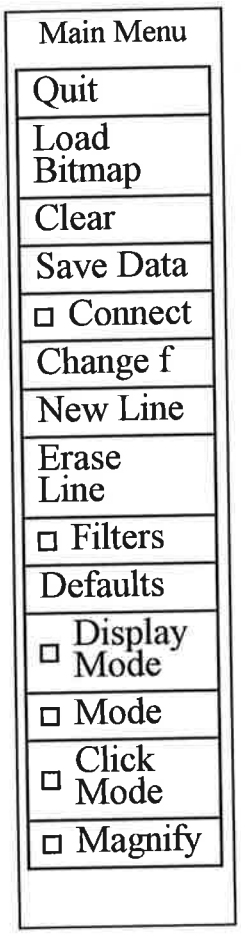
Fig. 1.1: Appearance on screen.

2. Menus

The program has a main menu and four sub menus.

The sub menus are popped up by clicking the appropriate button in the main menu, and are poped down by clicking the same button again. The sub menus can be moved freely on screen while they are popped up.

2.1. Main Menu



The main menu contains buttons for popping up sub menus and buttons for performing frequently used actions. The buttons that pop up sub menus are:

- Filters

Pops up the menu with filter functions for automatic processing of the image.

- Display Mode

Pops up a menu where the user can decide which image to display on screen.

- Mode

Pops up the menu where the user can specify in which order to sort data points on the same line.

- Click Mode

Pops up the menu where the user can decide how the program will react to mouse input.The buttons that perform actions are:

- Quit

Quits the application.

- Load Bitmap

Loads a bitmap from a msp-file on disk. The filename is specified by the user in a file selection box.

- Clear

Deletes all digitized information. A warning is displayed, where the user can cancel the operation.

- Save Data

Saves digitized information in a text file of standard Flygprestanda format. The filename is specified by the user in a file selection box.

- Connect

The program can show connections between points belonging to the same data line. This button toggles between showing, and not showing these connections.

- Change f

Lets the user type in a new f-value of the current data line.

- New Line

Creates a new data line. The user is prompted to type the f-value of the new data line. If this value matches the f-value of an existing data line, the data line with this f-value will be the current data line.

- Erase Line

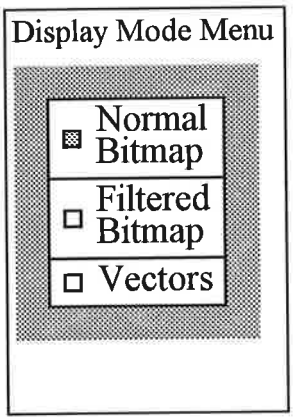
Deletes the current data line.

- Defaults

Pops up an input form where the user can change some default values.

Coo. Cross Dimension:	Size of matching cross when searching coordinate line intersections with the filter function Coo. Intersections.
Vector Length:	Minimal vector length in pixels when performing the filter function Vectorize.
Search Memory:	Number between 0 and 100 specifying the amount of memory when auto searching for data lines.
Region Size:	Size in regions of click sensitive regions.
Length of Dead Ends:	Longest paths to be deleted with the filter function Erase Dead Ends.
MIN. Path Length:	Specifies the minimal length of paths that are made click sensitive.
Magnify:	Pops up the magnifying window.

2.2. Display Mode Menu



The display mode menu has one button for each display mode, of which only one can be active.

- Normal Bitmap

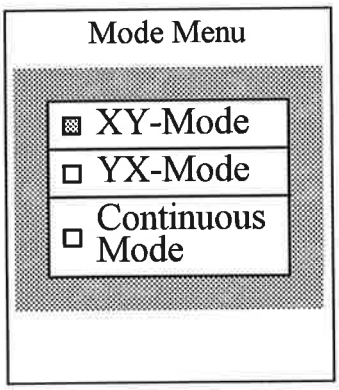
The original bitmap is displayed in the work space.- Filtered Bitmap

The filtered bitmap is displayed in the work space.

- Vectors

A vectorized representation of the filtered image is displayed in the work space.

2.3. Mode Menu



The Mode menu has one button for each sorting mode.

- XY-Mode

Points on the same data line are sorted as function from x to y in rising x-values.

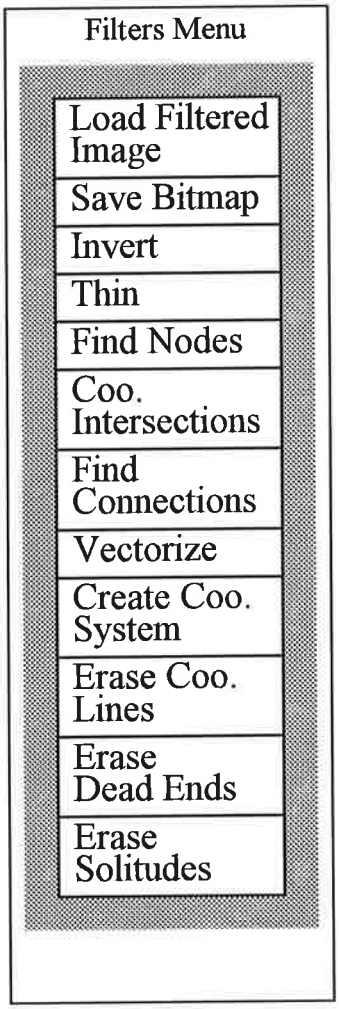
- YX-Mode

Points on the same data line are sorted as function from y to x in rising y-values.

- Continuous Mode

Points are not sorted.

2.3. Filters Menu



The Filters menu contains buttons for loading and saving a filtered image and for performing various filter functions on the image.

- Load Filtered Image

Loads a prefiltered image from a msp-file on disk. The user specifies the filename in a file selection box.

-Save Bitmap

Saves the filtered image in a msp-file on disk. The user specifies the filename in a file selection box.

- Invert

Inverts the image. The filter functions can only handle images where 1 = Set, 0 = Reset.

- Thin

Performs thinning on the image. Must be executed before executing Find Nodes.

- Find Nodes

Finds endpoints and intersections in a thinned image. Must be executed before executing Coo. Intersections or Find Connections.

- Coo. Intersections

For all the nodes found with Find Nodes this function tries to match a cross in the original image, with width and height according to the default value Coo. Cross Dimension. Must be executed before executing Create Coo. System.

- Find Connections

Finds connections between the nodes found with Find Nodes, by following paths in the thinned image. Builds a nework out of the nodes and connections. Must be executed before executing Vectorize.

- Vectorize

Transforms the paths found with Find Connections into sequences of vectors with a minimal length according to the default value Vector Length. Must be executed before doing automatic or semi automatic digitizing of data lines.

- Create Coo. System

Adjusts coordinate system according to the coordinate intersections found with Coo. Intersections.

- Erase Coo. Lines

Erases paths found with Find Connections that are close to the connecting lines between coordinate intersections.

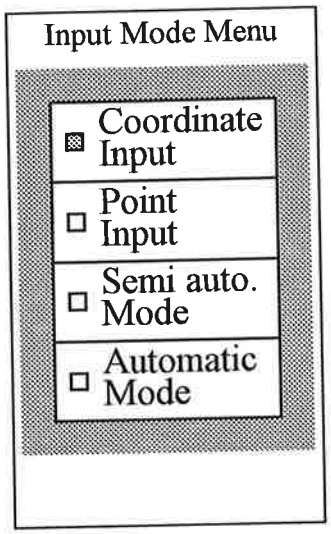
- Erase Dead Ends

Erases paths where at least one end is an endpoint. Paths must be longer than the default value Length of Dead Ends in order to be erased.

- Erase Solitudes

Deletes all lonely points.

2.5. Input Mode Menu



The input mode menu has one button for each mouse input mode.

- Coordinate Input

In this mode coordinate intersections can be placed and moved. See Coordinate handling.

- Point Input

In this mode data points can be placed and moved. When the button is pressed the New Line function is automatically executed.

- Semi auto. Mode

In this mode the vectors of a selcted connection are turned into data points of the current data line. When the button is pressed the New Line function is automatically performed.

- Automatic Mode

In this mode the program tries to find a whole data line by searching in the network created by Find Connections in the Filters Menu. When the button is pressed the New Line function is automatically performed.

3. Mouse Input

3.1. Entering coordinate systems

3.1.1. Creating a Coordinate System

In order to create a coordinate system, the click mode must be **Coordinate Input**. A starting coordinate system is created by specifying the coordinates for three points in the bitmap. This is done as follows.

1. Click the left mouse button on a point with known coordinates. Enter the coordinates for the point.
2. Click the left mouse button on a point with the same y-coordinate as the first point and higher x-coordinate than the first point. Enter the coordinates for the second point.
3. Click the left mouse button on a point with the same x-coordinate as the first point and higher y-coordinate than the first point. Enter the coordinates for the third point.

If the coordinates have been correctly entered, an input box will appear where the spacing between coordinate lines in x- and y-directions should be entered. When this is done the coordinate lines will appear on screen.

A coordinate point can be moved by pointing at it and moving the mouse with the center button pressed.

A coordinate point can be deleted by pointing at it and clicking the right button.

3.1.2. Changing the Coordinate System

When the coordinate system has been created according to 3.1.1. the coordinate line intersections can be moved by clicking with the left mouse button in the new position for the intersection. The Input Mode must be **Coordinate Input**.

If there are coordinate lines in the original bitmap, the coordinate system can be automatically corrected with the function **Create Coo. System**. in the filters menu.

3.2. Digitization of Lines

3.2.1. Manual Digitization

In order to digitize data lines manually, the input mode must be **Point Input**. When the Point Input mode is entered, the New Line function is automatically executed. The user is prompted for the f-value of the new line. If an existing f-value is entered, no new line is created. This is instead one way of changing the current data line.

Data points are **entered** to the current data line by pointing with the mouse and clicking the left mouse button.

Data points are **moved** by pointing and moving with the center button pressed.

Data points are **deleted** by pointing and clicking the right mouse button.

Current data line is changed by clicking the center button on a point on the desired data line. Another way to do the same thing is to execute the New Line function and specify the f-value of the desired data line.

The current data line is deleted by clicking the button Erase Line.

3.2.2. Semi Automatic Digitization

In order to perform semi automatic digitization the input mode must be **Semi auto. Mode**. When this mode is entered, there will appear boxes on every connecting path that is long enough.

In this mode one connecting path can be digitized by clicking the left button on the appropriate box.

A connecting path can be deleted by clicking on the box with the right mouse button.

3.2.3. Automatic Digitization

In order to perform automatic digitization the input mode must be **Automatic Mode**. When this mode is entered, there will appear boxes on every connecting path that is long enough.

In this mode a whole data line is digitized by clicking the left mouse button on a connecting path belonging to the data line.

A connecting path can be deleted by clicking on the box with the right mouse button.

4. Sample Session

The normal procedure for digitizing a diagram of functional curves is as follows:

1.

Load the original bitmap with **Load Bitmap**.

2.

Specify an initial coordinate system according to sec. 3.1.1.

3.

Pop up the **Filters** menu.

4.

If needed, invert the bitmap with the **Invert** command.

5.

Thin the image with the **Thin** command.

6.

Find endpoints and intersections with **Find Nodes**.

7.

If there is a visible coordinate system, execute **Label Coo. Intersections** followed by **Create Coo. System** to correct the coordinate system.

8.

Execute the **Find Connections** command to find the connections between endpoints and intersections.

9.

Vectorize the image with **Vectorize**.

10.

Pop up the **Input Mode** menu and change to **Automatic Mode**.

11.

Enter the f-value for the data line to be digitized.

12.

Click on the sensitive box of a part of the data line. The line will now be digitized. If part of the line was not digitized, click on a sensitive box on that part.

13.

Press **New Line** to digitize a new line.

14.

Repeat steps 11-13 for every line in the diagram.