The UTIPS Image Processing System

Vincent S. S. Hwang

AI TR87-51 April 1987

The support of the MCR grant from the Department of Computer Sciences at the University of Texas at Austin is gratefully acknowledged.
The UTIPS Image Processing System

Vincent S. S. Hwang
Artificial Intelligence Laboratory
University of Texas at Austin
April, 1987

Abstract

This document describes the UTIPS image processing software for the SYMBOLICS 3600 series Lisp machine with optional 8-bit color system running Genera 7.0 operating system. UTIPS contains more than 40 image manipulation and image processing functions and is implemented using Common-Lisp.

Copyright 1987, Vincent Hwang

The support of the MCR Grant from the Department of Computer Sciences at The University of Texas at Austin is gratefully acknowledged.
# Table of Contents

1 Introduction ........................................................................................................... 2  
2 UTIPS Functions .................................................................................................... 2  
   2.1 UTIPS Image Format ................................................................. 2  
   2.2 Low Level Image Operators ..................................................... 2  
   2.3 Image Information ................................................................. 4  
   2.4 Displaying Images ............................................................................. 4  
      2.4.1 Display Configuration .................................................... 5  
      2.4.2 Image Displaying Functions ......................................... 6  
   2.5 Operations on Display Window .................................................. 6  
   2.6 Image File Operations ..................................................................... 7  
      2.6.1 Image File Format .......................................................... 7  
      2.6.2 Image Files Operations ................................................ 8  
   2.7 Image Operations ............................................................................. 9  
   2.8 Other Displaying Functions ....................................................... 12  
3 Interactive Menu: UTIPS-MENU ......................................................................... 14  
   3.1 Overview ....................................................................................... 14  
   3.2 Define Working Window ............................................................ 16  
   3.3 Select and Show Working Image .............................................. 17  
   3.4 Save Window ................................................................................. 18  
   3.5 Histogram ..................................................................................... 19  
   3.6 Threshold ....................................................................................... 20  
   3.7 Halftoning Image ........................................................................... 21  
4 Appendix ............................................................................................................ 22  
   4.1 Loading UTIPS ............................................................................... 22  
   4.2 A Sample Session ............................................................................ 23  
   4.3 Source Code for Some UTIPS Functions .................................... 24  
      4.3.1 Threshold ............................................................................... 24  
      4.3.2 Copy-image ............................................................................ 24  
      4.3.3 Sobel ....................................................................................... 24
1. Introduction

UTIPS is an image manipulation system for the Symbolics 3600 Lisp machine with optional 8-bit color system. This system is implemented using Common Lisp. A knowledge of the Symbolics Lisp Machine, the Symbolics Window system, and the Common-Lisp is required to fully understand this document.

There are more than 40 image manipulation and processing functions in the UTIPS image processing system. UTIPS supports three modes for the display of images on the color display: 256 levels of gray; 64 levels of gray with three overlay colors; and the standard 8-bit pseudo color. To display gray image on the systems without color system, the UTIPS provides the capability of halftoning the image and display the resulting image on the B/W console.

All the functions in the UTIPS system can be called in the Common-Lisp User package (cl-user). There are two ways in which one can use these functions. One way is to call the function in user program. The other way is to use the function, utips-menu, to interactively use the UTIPS system. Section 2 describes the functions available in the UTIPS system while section 3 describes the interactive menu utips-menu. Section 4 presents the loading of the UTIPS system, a sample session, and the source codes for some functions in the UTIPS system.

2. UTIPS Functions

This section describes basic UTIPS functions.

2.1. UTIPS Image Format

The primary data object of UTIPS is the image. It is implemented as a two-dimensional array. The origin of an image is at the left corner. X-dimension refers to the number of columns of the image while Y-dimension refers to the number of rows of the image. Pixel indices start at 0 and is stored in row major.

The type of an image is determined by the pixel value. UTIPS currently supports the following image types:

1. Binary: The pixel value can be either 0 or 1.
2. Overlay: The pixel value can be 0, 1, 2, or 3.
3. Integer: The pixel value can be any integer between 0 and 255 (inclusive).
4. Float: The pixel value can be any floating number.
5. Fixnum: The pixel value can be any integer.
6. Lispobj: The pixel value can be any lisp object.

Each image can be attached with a comment. The comment is a string which can be used to describe the content of the image as well as the operations performed on the image.

2.2. Low Level Image Operators

make-2d-image $x$-dimension $y$-dimension &rest keyword-arguments

Make-2d-image allocates an image with dimension specified by $x$-dimension
and \textit{y-dimension} which must be two positive integers. \texttt{Make-2d-image} also accepts the following keyword arguments:

\texttt{:image-type} This specifies the pixel type of the image. It can be any one of the allowable image type. The default value is INTGERR.

\texttt{:comment} This specifies the initial comment of the image. The default value is NIL.

\texttt{:initial-element} This is identical to the \texttt{:initial-element} for \texttt{make-array} of Common-Lisp. The default value is 0 for non LISPOBJ image, and NIL otherwise.

\texttt{:image-name} This is the image name maintained by the UTIPS image database manager. If this parameter is not given, an unique name will be generated. Note that the image-name is also the print-name of the image.

Example: To create an 256 x 256 8-bit image filled with 100 and with comment "New Image" and with image name \texttt{myimage}, use

\begin{verbatim}
(make-2d-image 256 256
 :image-type 'integer :initial-element 100
 :comment "New Image":image-name 'myimage))
\end{verbatim}

\texttt{make-similar-image master-image &rest keyword-arguments}

\texttt{Make-similar-image} allocates an image with similar attributes as \texttt{master-image}. The \texttt{keyword-arguments} are the same as that of \texttt{make-2d-image}, and, if specified, will override the attributes of \texttt{master-image}.

Example: To create an image called \texttt{floatimage} which is similar to \texttt{myimage} but with floating point pixel value, use:

\begin{verbatim}
(make-similar-image myimage
 :image-type 'float :image-name float-image))
\end{verbatim}

\texttt{copy-image master-image copying-image &key image-type image-name}

\texttt{Copy-image} duplicates \texttt{master-image} into \texttt{copying-image}. If \texttt{copying-image} is NIL, a new image similar to \texttt{master-image} is created. \texttt{Image-type}, if specified, becomes the image type of \texttt{copying-image}. \texttt{Image-name}, if specified, becomes the image name of \texttt{copying-image}.

\texttt{unmake-image image}
Unmake-image removes image from the UTIPS image database and frees up the storage.

print-all-utips-image-names

Prints the print-name of all the images in the UTIPS image database.

2.3. Image Information
describe-image image

Returns the attributes of image.

add-image-comment image comment

Add-image-comment append comment to the end of existing comment for image. Comment must be a string.

image-x-dim image

Returns the x dimension of image.

image-y-dim image

Returns the y dimension of image.

iset value image ind-x ind-y

Store value into the pixel at location ind-x, ind-y. Both ind-x and ind-y must be integers within the range (the x, and y dimension of image). Value must be a value of the same type as the image-type of image. No type conversion is made.

iref image ind-x ind-y

Returns the pixel value at location (ind-x, ind-y) of image. Both ind-x and ind-y must be integer.

2.4. Displaying Images

This section describes functions for displaying images on the console or on the color display. We first discuss various screen configurations used by UTIPS. This is
followed by the description of image display functions.

2.4.1. Display Configuration

UTIPS defines the orientation of the color screen and the console as follows:

- Origin: Top-left corner of the screen.
- X dimension: Increments toward the right.
- Y dimension: Increments toward the bottom.

Two screen configurations are available for the color display. The first configuration, \textit{SCREEN}−1, is the full color screen; while the second configuration, \textit{SCREEN}−2, decompose the screen into four equal quadrants. Initially, configuration \textit{SCREEN}−2 is used. To change the screen configuration from configuration \textit{screen}_a into configuration \textit{screen}_b, use function

\texttt{(change-color-screen-configuration screen}_b\texttt{)}.

When one changes the color screen display configuration from \textit{screen}_a into configuration \textit{screen}_b, all the images displayed in the windows in configuration \textit{screen}_a will be hidden (by the windows of configuration \textit{screen}_b). Instead, all the images that was in the windows in configuration \textit{screen}_b will reappear (exposed).

The windows created using these configurations are as follows:

1. \textit{QUAD}−0: This is the window created in configuration \textit{SCREEN}−1. It is the full color screen. The top-left corner locates at (0,0) and the lower-right corner locates at (1279,1023). This window is initially deexposed.

2. \textit{QUAD}−1: This is a window created in configuration \textit{SCREEN}−2. It is the upper-left quadrant of the color display. The top-left corner locates at (0,0) and the lower-right corner locates at (639,511). This window is initially exposed.

3. \textit{QUAD}−2: This is a window created in configuration \textit{SCREEN}−2. It is the upper-right quadrant of the color display. The top-left corner locates at (640,0) and the lower-right corner locates at (1279,511). This window is initially exposed.

4. \textit{QUAD}−3: This is a window created in configuration \textit{SCREEN}−2. It is the lower-left quadrant of the color display. The top-left corner locates at (0,512) and lower-right corner locates at (639,1023). This window is initially exposed.

5. \textit{QUAD}−4: This is a window created in configuration \textit{SCREEN}−2. It is the lower-right quadrant of the color display. The top-left corner locates at (512,512) and lower-right corner locates at (1279,1023). This window is initially exposed.

UTIPS supports three color maps on the color display system.

1: Standard Color. This is the same as the standard color for the Symbolics color system. Check the Symbolics manual for details. To set the color display in this mode, use:
(set-color-display-mode 'standard-color)

2. Full Gray. In this mode, the color screen can display 256 different levels of gray. Pixel value 0 is displayed as black while pixel value 255 is displayed as white. Value between 0 and 255 is displayed as gray accordingly. This is the most commonly used mode. To set the color display in this mode, use:

   (set-color-display-mode 'full-gray)

3. Overlay and Gray. In this mode, the color screen display 64 levels of gray and 3 colors (full red, full green and full blue). The pixel is displayed according to the following formula:

<table>
<thead>
<tr>
<th>Pixel Value</th>
<th>Displaying Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel-value MOD 4 = 0</td>
<td>Gray (0 as Black, 252 as White)</td>
</tr>
<tr>
<td>pixel-value MOD 4 = 1</td>
<td>Full Red</td>
</tr>
<tr>
<td>pixel-value MOD 4 = 2</td>
<td>Full Green</td>
</tr>
<tr>
<td>pixel-value MOD 4 = 3</td>
<td>Full Blue</td>
</tr>
</tbody>
</table>

To set the color system in this mode, use:

   (set-color-display-mode 'overlay-and-gray)

   In addition to the color display, one can also display binary image on the Dynamic Lisp Listener (with print name *standard-output*).

2.4.2. Image Displaying Functions

display-image window image &key from-x from-y to-x to-y erase

   Display image in window. From-x and from-y indicate the top-left corner of the rectangular portion of image to be displayed. To-x and to-y indicate the top-left corner of window where image is to be displayed. The values of these parameters must be integer and their default values are 0. Erase (possible values: t or nil) indicates whether the window should be erased before displaying the image. The default value for erase is t (i.e., erase the display region before displaying image).

2.5. Operations on Display Window

erase-window window &key from-x from-y to-x to-y erase

   Erase the content of window inside the region specified by from-x, from-y, to-x, and to-y. Erase (possible values: default, image, overlay or binary) indicates the bit-planes are to be reset. Default specifies the entire pixel value is to be set to 0, image specifies the entire pixel value except the lower two bits is to be set to 0, overlay specifies that only the lower two bits is to be set to 0, and
binary specifies that only the lowest bit is to be set to 0.

```
save-window window image &key from-x from-y to-x to-y
```

Save-window copies the image displayed in the rectangular portion specified by from-x from-y (the upper left corner of window) and to-x to-y (lower right corner of window) and save it in memory as image. If image is NIL, a new image of proper size is created. The default values for from-x and from-y are both 0. The default values for to-x and to-y are the X and Y dimension of the window, respectively.

```
get-point-position window
```

Get-point-position allows user to obtain current mouse position and the pixel value pointed to by the mouse. Window specifies the window in which the mouse is to be tracked. The left or the right button defines the point. The middle button exits. The coordinate of the point and the pixel value at the point are returned.

```
get-rectangle-position window
```

Get-rectangle-position allows user to define a rectangular subwindow of window using mouse. The left button defines the upper left corner of the subwindow while the right button defines the lower right corner. The middle button exits. Get-rectangle-position returned five values. The first value indicates the button pressed (see Symbolics Manual), the next two numbers indicate the upper left corner, and the last two numbers indicate the lower right corner.

```
mouse-pan-and-zoom
```

Perform pan and zoom on the color display. The mouse centers at the color screen. The left button zooms in and the right button zooms out. Moving mouse around achieves the panning effect. Pressing the middle button exits. A pop menu will be displayed upon exiting. To restore the screen to its original status, click on YES. The default value is NO.

2.6. Image File Operations

This section describes the functions on UTIPS image file. We first discuss the UTIPS image file format. This is followed by the description of image display functions.

2.6.1. Image File Format

An UTIPS image file is described by two files:

1. A header file. This file indicates the properties of an image. It includes the x-dimension, y-dimension, image-type, element-type, comment, and name of
the image.

2. A data file: this file contains the pixel values stored in row major.

Suppose that an UTIPS image file whose name is picture. The name of the image is DOG. The x-dimension and the y-dimension of DOG are 256, the image-type is INTEGER, and there is no comment. Then, the pathname (in the Symbolics file system) of the header file is picture.image, while the pathname (in the Symbolics file system) of the data file is picture.data. The content of file picture.image is:

;;;*- Mode:lisp; Syntax:Common-Lisp; Package:common-lisp-user -*-

(display-image-file :image-type 'INTEGER
: name "DOG"
: size-x 256
: size-y 256
: element-type '(MOD 256)
: comment "")

Function display-image-file loads the image file and display the image parameters on the console. If there is a color system, the image will be displayed in the upper left window of the color display.

2.6.2. Image File Operations

UTIPS maintains a default pathname (current working image directory) to be used in order to generate a complete pathname for incomplete file name specification. All the file names in UTIPS, if not completely specified, will be merged with this default pathname to generate a complete pathname.

load-image-file image-file image &key image-name

Load-image-file loads the image-file from the Symbolics file system into memory as image. If image is NIL, an image of appropriate size is created. Image-name, if specified, is used as the print-name of the image. Otherwise, a new image name is generated.

load-non-utips-image-file image-file image xsize ysize
&key headersize image-name image-type byte-size

Load-non-utips-image-file loads a header encoded image-file from the Symbolics file system into memory as image. Image-name specifies the pathname of the file contains the image. Xsize and ysize specify the dimensions of the image to be loaded. If image is NIL, an image of appropriate size is created. In this case, image-name and image-type are used to create the image. Headersize indicates the length (in bytes) of the header, and the byte-size indicates the number of bits per pixel of the image to be loaded.
save-image image image-file

Save image (in memory) as file image-file (an UTIPS image file name).

describe-image-file image-file

Returns the parameters of image-file.

print-image-directory

Returns the current working directory. This directory is appended to the image file name to generate the full pathname.

set-image-directory image-directory

Set the current working image directory to be image-directory.

list-image-file-in-directory &optional directory

Returns the UTIPS image file name in directory. If directory is not specified, the default image directory (current working image directory) is used.

2.7. Image Operations

dirnms mag-image dir-image result &key angular-tolerance image-name

Dirnms computes the location of edge point using directional non-maximum suppression. Mag-image and dir-image are the magnitude and direction of edge response computed by sobel. Angular-tolerance indicates the allowable errors during the suppression operation. The default value for angular-tolerance is 60 degree. The suppressed edge is stored in result. If result is NIL, an image similar to mag-image is created. Image-name, if specified, becomes the image name of new image.

halftone image result &key x-scale y-scale invert image-name

Halftone computes the halftoned image of image and store the result in result. The x and y dimensions of result is x-scale and y-scale times the x and y dimension of image, respectively. Both x-scale and y-scale must be larger than or equal to 1. Invert (possible values: 0 or 1) indicates the value to be used for white pixel. The default values for x-scale and y-scale are both 1.0, and the default value for invert is 1. Result must be a binary image. If result is NIL, an
image similar to image is created. Image—name, if specified, becomes the image name of new image.

`histogram image window &key scale display—size from—x from—y`

**Histogram** computes the histogram of image and display the result in window. Scale (possible values: default or any numerical values) indicates the number of pixel per displayed point. If scale is not specified, histogram computes an optimal value to fit the entire histogram in window. The size of the displayed histogram can be either 256 x 256 (when display-size is 1), or 512 x 512 (when display-size is 2). The histogram is displayed with upper left corner at (from—x, from—y). The default values for from—x and from—y are both 0.

`intensity-stretching image result &key min—in max—in min—out max—out image—name`

**Intensity-stretching** stretched the distribution of the intensity of image by linearly map pixel (of image) values between min—in and max—in (inclusive) into values between min—out and max—out (inclusive). The result is stored in result. The default values for min—in and min—out are the minimal value and the maximum value of the pixel values of image, and the default values for min—out and max—out are 0 and 255, respectively. If result is NIL, an image similar to image is created. Image—name, if specified, becomes the image name of new image.

`mirror-image axis image result &key image—name`

**Mirror** computes the mirror image of image with respect to axis axis. Axis must be either X or Y. If result is NIL, an image similar to image is created. Image—name, if specified, becomes the image name of new image.

`point-operation image result operation—specification &key image—name image—type`

**Point-operation** performs a point operation on the image (i.e., the operation depends only on the pixel value at the pixel). The result is stored in result. Operation—specification is a lisp expression to be evaluated at every pixel. The pixel value is represented by px. If result is NIL, an image similar to image is created. Image—name and image—type, if specified, are used to create result. For example, to add 3 to each pixel, use:

```lisp
(point-operation image result '(+ px 3))
```
**rotate-image** degree image result &key image-name

**Rotate-image** computes result of rotating image in clockwise direction degree degree. Degree must be one of 90, 180, or 270. If result is NIL, an image similar to image is created. Image-name, if specified, becomes the image name of new image.

**sobel** image &key mag dir

**Sobel** performs the sobel edge operation on image. The magnitude of the edge operation is stored in mag, and the direction of the edge is stored in dir. Image-type for image is integer, and is float for dir. Both mag and dir must exist before calling sobel.

**selective-smooth-ku** image result &key image-name

**Selective-smooth-ku** performs the selective smoothing operation on image as described in the article "Edge Preserving Smoothing" by M. Nagao and T. Matsuyama, 1980. If result is NIL, an image similar to image is created. Image-name, if specified, becomes the image name of new image.

**selective-smooth-um** image result &key image-name

**Selective-smooth-um** performs the selective smoothing operation on image as described in the article "Two New Image Smoothing Operators" by H. Hakalahti and I. Virtanen, UM-CAR-TR-55, 1984. If result is NIL, an image similar to image is created. Image-name, if specified, becomes the image name of new image.

**size-scaling** image result &key x-size y-size

**Size-scale** returns an image which is an interpolation (bilinear interpolation) of image. The x and y dimension of the interpolated image are x-size and y-size, respectively. The result is stored in result. If result is NIL, an image similar to image is created. Image-name, if specified, becomes the image name of new image.

**thin** image result &key connectivity on-value image-name

**Thin** performs thinning operation on image. The image-type of image must be integer. Pixel value 0 indicates background and 1 indicates object (to be thinned). The thinned result is stored in result which can has arbitrary image type.
Connectivity indicates the type of connection to be used. It can be either 4 or 8 (default is 4). The resulting skeleton pixel is assigned with on-value (default is 1). If result is NIL, an image similar to image is created. Image-name, if specified, becomes the image name of new image.

threshold image result
&key lth uth on-value off-value image-name image-type

Threshold thresholds image and stored the result in result. All pixels with intensity between lth and uth (inclusive) are assigned with new pixel value on-value. Other pixels are assigned with value off-value. The default values for lth and off-value are both 0, and the default values for uth and on-value are both 255. Result can be any image-type. If result is NIL, an image similar to image is created. Image-name and image-type, if specified, are used to create result.

zero-cross image result diameter &key image-name

Zero-crossing computes the zero crossing of image using gaussian mask whose diameter is diameter. The zero-crossings are indicated by 1's in image result. See article "Implementation of a Theory of Edge Detection" by E. Hildreth, S.M. Thesis, MIT, 1980 for details. If result is NIL, an image similar to image is created. Image-name, if specified, are used to create result. The default image type of result is binary.

2.8. Other Displaying Functions

display-array an-array window &key from-x from-y erase fill

Display-array displays the content of an-array in window with upper left corner at (from-x,from-y). The X-dimension represents the index of the array while the Y-dimension represents the value stored at that array cell. erase (possible values: t or nil) indicates if the region used to display the array is to be erased. Fill (possible values: t or nil) indicates if the displayed graph is to be filled (i.e., the space between the X-axis and the graph). The default value for erase is t and the default value for fill is nil.

draw-line from-x from-y to-x to-y &key color

Draw-line draws a line from (from-x,from-y) to (to-x,to-y) in window. The color of the line is specified by color (possible values red, green, blue, or white).

movie window image-sequence inter-frame-sleep
&key to-x to-y delay
Movie displays a sequence of images, image-sequence, in window at (to-x,to-y) in motion. Inter-frame-sleep indicates that amount of delay between consecutive frame (in 100th of a second). Delay, if specified, indicates the sequence is to be played back infinitely many times. The amount of time to be delayed is described by delay (in 100th of a second).
3. Interactive Menu: UTIPS-MENU

3.1. Overview

UTIPS-MENU provides an interactive user interface to manipulate images. The function uses the last four print lines on the console screen to display messages (Figure 1). Upon exit from the function, these region will be cleared. Therefore, it is best to avoid using this region.

UTIPS-MENU is designed for easy usage. The following for sets of parameters are used as defaults by the UTIPS-MENU to generate appropriate parameters for a selected function.

1. The screen format parameters. These parameters describe the current configuration of the color system. These parameters must be defined, use SET-UTIPS-ENV before one display images onto the color display.

2. The working window parameters. These parameters describe the window that is to be used by any operation on a screen. These parameters must be defined, use DEFINE-WORKING-WINDOW before performing any window operation.

3. The current image directory. This parameter describes the image directory that is to be used by to generate the complete pathname for partially specified image file name. To set the current image directory, use SET-IMAGE-DIRECTORY.

4. The current working image. This parameter describes the image that is to be used by any operation on image(s). To set the current working image, use SELECT-WORKING-IMAGE.

To use this interactive menu, type

(utips-menu)

To exit from the interactive menu, click on EXIT.

In the following sections, we illustrate some examples of using UTIPS-MENU to manipulate images.
Figure 1: UTIPS-MENU Screen Layout
3.2. Define Working Window

Step 1: Select a working window.
Step 2: Select means to define a subwindow.
Step 3: Define the subwindow.
3.3. Select and Show Working Image

```
SELECT UTIPS PROGRAM
SET-UTIPS-ENV  SET-IMAGE-DIRECTORY  HISTOGRAM
SHOW-UTIPS-ENV  LOAD-IMAGE-FILE  THRESHOLD
PAN-AND-ZOOM  DELETE-UTIPS-DB-IMAGE  IMAGE-SIZE-SCALING
DEFINE-WORKING-WINDOW  SELECT-WORKING-IMAGE  STRETCH-INTENSITY
SAVE-WINDOW  DISPLAY-WORKING-IMAGE  SOBEL-EDGE
PRINT-WINDOW  ERASE-DISPLAY-WINDOW  HALFTONING-IMAGE
MAKE-TITLE  COPYING-WORKING-IMAGE  POINT-OPERATION
LOAD-NON-UTIPS-IMAGE  SAVE-IMAGE-TO-FILE  ROTATE-IMAGE
EXIT  SHOW-IMAGE-INFO  MIRROR-IMAGE

SELECT WORKING IMAGE
#UTIPS-WINDOW-IMAGE#
ARCH-L  ARCH-R
LETTER  TOWER  TOWER-HALFTONE
```

Image name: TOWER-HALFTONE; Column Size: 256; Row Size: 256
Image type: BINARY; Element type: (MOD 2)
3.4. Save Window

Save the content of the rectangle defined by current window into image with print-name *utips-window-image*.

Step 1: Select the bits to be saved.
  ALL : all bit planes.
  IMAGE : all bit planes except the lowest two bits.
  OVERLAY : the lowest two bit planes.
  BINARY : the lowest bit plane.
3.5. Histogram

Step 1: Select a working image.
Step 2: Select HISTOGRAM program.
Step 3: Select the display parameters.
Step 4: Define the displaying coordinates.
3.6. Threshold

Step 1: Select a working image.
Step 2: Select THRESHOLD program.
Step 3: Select the operation parameters.

```
SELECT UTIPS PROGRAM
SET-UTIPS-ENV    SET-IMAGE-DIRECTORY    HISTOGRAM
SHOW-UTIPS-ENV   SET-IMAGE-DIRECTORY	X-THRESHOLD
PAN-AND-ZOOM     LOAD-IMAGE-FILE       IMAGE-SIZE-SCALING
DEFINE-WORKING-WINDOW SELECT-WORKING-IMAGE STRETCH-INTENSITY
SAVE-WINDOW      DISPLAY-WORKING-IMAGE SOBEL-EDGE
PRINT-WINDOW     ERASE-DISPLAY-WINDOW HALFTONING-IMAGE
MAKE-TITLE       COPYING-WORKING-IMAGE POINT-OPERATION
LOAD-NON-UTIPS-IMAGE SAVE-IMAGE-TO-FILE ROTATE-IMAGE
EXIT             SHOW-IMAGE-INFO      MIRROR-IMAGE
```

THRESHOLD IMAGE: TOWER
Output Image : LOWER-THRESHOLD
Output Image Type : BINARY OVERLAY
Lower Threshold Value : 0
Upper Threshold Value : 175
On Pixel Value : 1
Do it ? : Y N Y L

```
result of threshold
```

20
3.7. Halftoning Image

Step 1: Select a working image.
Step 2: Select HALFTONING-IMAGE program.
Step 3: Select the operation parameters.

![Halftoning Image Diagram]

**Output Image**: TOWER-HALFTONE
**Output Image Type**: BINARY OVERLAY
Invert Output**: YES NO
Size Scaling (in Column): 1.0
Size Scaling (in Row): 1.0
Do It? : Y/N

Result of halftoning

**Current Working Image**: TOWER
Displaying Window: Dynamic Lisp Listener 1
4. Appendix

4.1. Loading UTIPS

To load the UTIPS system, type:

LOAD SYSTEM UTIPS

The Symbolics will prompt:

UTIPS : CHECK SYSTEM CONFIGURATION .....  
UTIPS : Is There A Color System ? (Y or N)

If there is a color system attached to your Symbolics, answer Y. Then, the Symbolics will load the color system and print:

UTIPS : System Configuration :  
COLOR SYSTEM EXISTS  
USE BOTH CONSOLE AND THE COLOR SCREEN AS IMAGE DISPLAY DEVICE  
UTIPS : CREATE COLOR WINDOW  
UTIPS : DONE

If there is no color system attached to your Symbolics, answer N. Then, the Symbolics will print:

UTIPS : System Configuration :  
NO COLOR SYSTEM  
USE CONSOLE AS IMAGE DISPLAY DEVICE  
UTIPS : DONE

In general, if any errors occur during the loading of the UTIPS system, RESUME key will get you through. Also, at some point, if you want to reset the entire UTIPS environment, use function (set-utips-init-parameters).
4.2. A Sample Session

Here is a sample session of using UTIPS to load image into memory, display image, and write a simple function to perform threshold.

Step 1: Set current image file directory to be "Alamo:\u201euTips\images\misc":
-> (set-image-directory "Alamo:\u201euTips\images\misc")

Step 2: Print the name of all the image files under current image directory:
-> (get-image-filename-under-dir)

Step 3: Load the image file tower into memory as an image object with print name tower-image:
-> (load-image-file "tower" nil :image-name 'tower-image)

Step 4: Display the image on the first quadrant of the color screen:
-> (display-image quad-1 tower-image)

Step 5: Create a binary image that is similar to tower-image with name tower-binary:
-> (make-similar-image tower-image :image-type 'binary :image-name 'tower-binary)

-> (halftone tower-image tower-binary)

Step 7: Display tower-binary on the console:
-> (display-image *standard-output* tower-binary)

Step 8: Print the names of all images in the UTIPS image data base:
-> (print-all-utips-image-names)

Step 9: Perform threshold on tower-image and store the result into tower.binary using threshold value th-val:
-> (loop for x from 0 below (image-x-dim tower-image) do
  (loop for y from 0 below (image-y-dim tower-image) do
    (if (> (iref tower-image x y) th-val)
      (iset 1 tower-binary x y)
      (iset 0 tower-binary x y))))
4.3. Source Code for Some UTIPS Functions

4.3.1. Threshold
(defun threshold (image result &key (lth 0) (uth 255) (image-type 'binary) image-name
  (on-value 1) (off-value 0))
  (if (null result)
      (setq result (make-similar-image image :image-name image-name
                                         :image-type image-type)) nil)

  (let ((tv val)
        (loop for j from 0 below (image-y-dim image) do
          (loop for i from 0 below (image-x-dim image) do
            (let ((tv (if (and (>= val lth) (<= val uth)) on-value off-value)))
              (setq result i j)))))

4.3.2. Copy-image
(defun copy-image (image result &key image-type image-name)
  (let ((temp1)
        (loop for j from 0 below (image-y-dim image) do
          (loop for i from 0 below (image-x-dim image) do
            (setq temp1 (iref image i j))
            (setq result i j)))))

4.3.3. Sobel
(defun sobel (image &key mag dir)
  (let ((cols (image-x-dim image)) (rows (image-y-dim image)) sx sy tv)
    (loop for j from 1 below (1- rows) do
      (loop for i from 1 below (1- cols) do
        (setq sx (+ (iref image (1+ i) (1- j))
                   (iref image (1+ i) (1+ j))
                   (ash (iref image (1+ i) j) 1))
        (setq sy (+ (iref image (1- i) (1+ j))
                    (iref image (1- i) (1+ j))
                    (ash (iref image (1- i) j) 1)))))
    (setq tv (+ (abs sx) (abs sy)))
    (setq mag i j)
    (setq tv (if (and (= sx 0) (= sy 0)) 0 (atan sy sx)))
    (if dir (iset tv dir i j))))