LUG

Librería de Utilidades Gráficas
(Graphics Utilities Library)

UNIVERSIDAD DE OVIEDO

Users Manual

Raúl Rivero

Adolfo Otero

to the memory of my grandmother

What is LUG ?

Introduction

LUG (***G**raphics **U**tilities **L**ibrary*) is born within the Mathematics Department LabCAD, University of Oviedo, because of the need of a massive digital image processes.

At the first moments, those processes were overcome by using interactive programs but after a while we realized that we had to use batch utilities and programs.

Let's see an example: a 20 second animation is made of 500 frames -or pictures (*20x25=500*). Let's suppose we wanna add a small logo -the one almost every TV channel uses in their broadcastings-, if we try to add it using an interactive program we could spend days in the task, but if we do it at the same time that we record the frames with the VCR we will not be able to see any difference.

When is it born?

First, every utility created was independent from the others, that is, if we wanted a program to resize the frames, we had to do it from the beginning; if we wanted a chroma program, we had to do it. Really, too hard work!.

Four or five months later -realizing that we had to do more and more complex tasks-we decided to build a library wich were able to accomplish all the tasks we were in need.

Beetwen the winter and 92 spring the main LUG functions were done.We spent summer and auttum in the verification proccess and the construction of the library utils.

As the program became more popular we thought that it could be interesting to publish the code through INTERNET. This implied a new code revision, the adittion of a few more functions, and above all, to prepare it to be used, compiled and modified by every potential user of the library.

In the beggining of the 93 summer the first LUG based programs are done (binaries only), not the library itself. In July it is published the first source code, it was *tovcr*, a program wich records animations -frame secuences- on video, allowing resampling, scaling, zooming, and so on. It is distributed worldwide by the USENET group *comp.sources.misc*.

The first LUG release is made in september, 1993, and it is the one you have in your hands.

Why so much time?

The keyword is... MANUALS. Really, is there anyone who enjoys writing a manual?

Well, let's be serious. The first problem was the beggining. LUG was created for self-purpose only, things were done as we were in need of them, without any order or planning.

Once we decided to publish it we faced the problem of the reestructuration of the library, we had to do some changes in the program philosophy and a long checking proccess was need.

Of course, the manuals took me a lot of time -well, I must admit that a L...O...T- and they had to be translated to english (thanks Jorge !).

Well, but ... what is it good for ?

In LUG there are three main parts:

i. Interfaces with the most popular graphic formats (TIFF, JPEG, GIF, TGA, Alias "pix", PCX, ...)

ii. Viewers for the most used computer architectures (X11, SGI, IBM r6000, HP9000 and PC)
iii. Digital image proccessing (quantizing, reescaling, masks, ...)

And they are all based in functions which can be called one after another with a high abstraction level.

Let's see an example: if we need of a program to reduce a GIF file in half of its the size, the code would be:

```
#include <lug.h>
#include <lugfnts.h>
#define NO_BLUR                 0

main( int argc, char **argv )
{
        if ( argc < 2 ) {
                /* Ooopps!, not enough arguments */
                fprintf(stderr, "Usage: %s <bitmap>\n", argv[0]);
                exit( 1 );
        }else {
                bitmap_hdr in, out;

                /* Read the input file */
                read_gif_file( argv[1], &in );
                /* Resample the image */
                adjust_bitmap( &in, &out, in.xsize/2, in.ysize/2,
                                NO_BLUR );
                /* Write the resampled image */
                write_gif_file( argv[1], &out );
        }
        exit( 0 );
}
```

It's clear that it is a complete program (even if we bear in mind the calling syntax and comments) and than the core body (read, reescaling and writing) only has three lines of code.

To whom is it pointed ?

Of course, to everyone who deals with images.

The abstraction level of the library allows its integration in other programs effortlessly. Reading and writing commercial graphic formats is done with only one instrucction and we are able to treat images individually or as a part of a series.

Some parts of LUG will make a happier living to some programmers (not that way!).

Of course, the main goal is SPEED, so the recomended hadware is a workstation or a machine with an OS with virtual memory, at least.

LUG has been tested on SGI, IBM r6000, HP9000 s300/400/800, Intergraph, DEC stations and PCs 386/486 (with a 32 bit compiler) and it appears work fine.

External libraries

Four of the supported formats are based on external libraries. However, LUG unifies the use of them, allowing the user to use then as a [common] simple instruction.

They are all divulged libraries and their use has become an standard:

- TIFF: libtiff library, from Sam Leffler
- JPEG: libjpeg library, from Thomas G. Lane, Independent JPEG Group´s Software
- RLE: librle library (Utha Raster Toolkit), from Spencer W. Thomas, U. of Michigan
- rgb-SGI: libimage library, released with sgi machines

The two latter are the official libraries of that formats so its use is the most normal.

The two former, well, we could say that they also are official. I have not seen a program (including something as powerful as Khoros) using those formats and not using those libraries, of course LUG wasn't going to be less -or more-.

Frankly, I think that those libraries are there to be used and LUG is intended to be used too. I think that the mania of trying to control everything and try to get all the credit has to be replaced by a more global vision, consisting in a team working, which is much more productive (and practical).

This mades me sick because as an administrator I know what the people is working on and how they deny to share the information -or they share it, but in a restricted way- with other users working in the same area. It's something I -and most people- will never understand.

I hope that LUG is copied and modified -respecting the copyrights- by anyone, because that is its main goal.

And now ... what ?

Nothing is perfect, so -by inclusion- LUG isn't either.

Improvings, contributions, ampliations, comments, suggestions, complainings, ... all these and much more is expected to happen.

As a resume, this is version 1.0, and I hope there are many more.

Index

Installation	*1*
Compiling with LUG	*9*
Inside LUG	*13*
Graphic File Formats Interfaces	*16*
Internal Library Functions	*26*
Viewers	*31*
Digital Image Processing Functions	*36*
Conclusion	*46*
Appendix A	*48*
Appendix B	*51*
Appendix C	*52*
Appendix D	*61*

Installation

Introduction

For impatiences: ***Makefile*** & ***include/lugconf.h***. Let's continue...

LUG has been written in C because it is the most used computer language and any serious operative system has it by default.

It has been verified its correct compilation and functionality with the standard compilers of the following machines: Silicon Graphics, HP 9000 (s300/400/800), IBM r6000, Intergraph and DEC stations. It has been also compiled with the *gcc* compiler on: HP9000 (s300/400/800), DEC and PCs.

Besides the compiler, it is also neccesary the *make* utility.

Other libraries

LUG supports 14 graphic formats, they are:

- gif, property of Compuserve.
- heightfield, used by Rayshade (Craig Kolb), write only
- jpeg/jtif, using Thomas G. Lane's library
- pbm/pgm/ppm from the PBM Plus package
- pcx, property of Zsoft Corporation
- Alias "pix", property of Alias Research
- postcript, property of Aldus, using J. G. Antuña Roces's algorithms, output only
- raw
- rgb

- rla, property of Wavefront Technologies
- rle, using the Utah Raster Toolkit original library, by Spencer W. Thomas
- rgb from sgi, using the library included with the SGI machines
- tga, property of AT&T-Truevision
- tiff, property of Aldus, using the libtiff library, by Sam Leffler

We have already made the moral disquisition of why are used the libraries of other authors, so we will deal with the installation proccess. But if you want to achieve the full power of LUG you will have to have installed them before starting the compilation process.

The original distributin sites of the original code are:

- jpeg: ftp.uu.net: /graphics/jpeg/jpegsrc.v?.tar.Z
- rle: princeton.edu: /pub/Graphics/urt/urt*.tar.Z
- tiff: sgi.com: /sgi/src/tiff/libtiff*.tar.Z

But they can be collected too from the same site of LUG:

telva.ccu.uniovi.es (156.35.31.31):

- /pub/graphics/Image/jpeg/jpegsrc.v?.tar.gz
- /pub/graphics/Image/urt/urt*.tar.gz
- /pub/graphics/Image/tiff/libtiff*.tar.gz

Each one of the libraries has its own installation instructions. One advise only, you have to use the same compiler in all of them (this includes LUG), if you don't do it that way you could experience some problems (trust me!!!).

A wise reader (:-)) should have noticed that I haven't especified where to find the exact localization of the SGI library. It is because it is reserved for the use with their own machines **only**, it is property of the company and they should be the ones to distribute it. Besides, it is a format used only by their own machines so using it in different ones makes no sense.

Let's go

If you are reading this documentation I suppose that you already have the LUG source code and that you have decompressed it, so I will override that step.

There are two files wich configure LUG, ***Makefile*** int the root directory and ***include/lugconf.h***. The former especifies wich parts are compiled and the latter especifies some internal options.

Let's see the format of the Makefile file:

```
#
# Makefile for LUG
#
# (c) 1992, Raul Rivero
#
# Makefile for the LUG Library.
#

SHELL = /bin/sh

#
# Directories of LUG package.
#
DIRS = convert lib show utils

#
# Uncomment next line if you are using other compiler (e.g. gcc)
#
#CC = gcc

#
# Compiler options ( only C-O-M-P-I-L-E-R, not LUG options ).
#
COPTS = -O

#
# Location of LUG header files.
#
LUGINC = -I../include

#
# If you have intalled the Utah Raster Toolkit library define
# next variables, example:
#
#    URTINC = -I/remote/carreras/usr/people/nuevos/correo/urt/include
#      URTDEF = -DiRLE
#
# , or leave undefined if you don't have it.
#
URTINC = -I/usr/local/include/rle
URTDEF = -DiRLE

#
# If you have intalled the Sam Leffler's TIFF library define
# next variables, example:
#
#    TIFFINC = -I/remote/carreras/usr/people/nuevos/correo/libtiff/include
#      TIFFDEF = -DiTIFF
#
# , or leave undefined if you don't have it.
#
TIFFINC = -I/usr/local/include/tiff
TIFFDEF = -DiTIFF

#
# If you have intalled the JPEG library define
# next variables, example:
#
#    JPEGINC = -I/remote/carreras/usr/people/nuevos/correo/jpeg/src
#      JPEGDEF = -DiJPEG
```

```
#
# , or leave undefined if you don't have it.
#
JPEGINC = -I/usr/local/include/jpeg
JPEGDEF = -DiJPEG

#
# If your machine is a SGI computer define next variables, ... ok,
# ok, an example:
#
#      SGIINC = -I/usr/include/gl
#      SGIDEF = -DiSGI
#
# , else leave undefined.
#
SGIINC = -I/usr/include/gl
SGIDEF = -DiSGI

#
# Define what kind of viwer you'll support. Options are:
#
#      Machine                  Add
#      ------------             ---------
#      hp9000                   -DiHP
#      sgi/aix                  -DiGL
#      pc                       -DiPC
#      x11                      -DiX11
#      vfr ( sgi )              -DiVFR  <--  SGI's Video Framer needs
#                                            hardware
#      linux                    -DiLINUX
#
# Probably, also you'll need give a path to the headers.
#
# And example: we have a SGI with X11 and a Video FRamer, so
# we define...
#
#      VIEWDEF = -DiGL -DiVFR
#      VIEWINC = -I/usr/include/gl -I/usr/video/vfr/src/inc
#
VIEWDEF = -DiGL -DiVFR -DiX11
VIEWINC = -I/usr/include/gl -I/usr/video/vfr/src/inc -I/usr/local/X11R5/include
.....
```

As can be seen, every optional part can be configured (*italic* typeface) and the added comments are explicit enough of what has to be done. Anyway, I will comment it a bit more.

In the upper listing it has been supposed the following hardware:

- Silicon Graphics (this implies the support for the SGI rgb format and the avaliability of the GL graphics library)
- A video controller (with its headers on their default path, */usr/video/vfr/src/inc*)
- X11R5 installed in */usr/local/X11R5*
- libtiff library with its headers placed in */usr/local/include/tiff*
- urt library with its headers placed in */usr/local/include/rle*
- jpeg library with its headers placed in */usr/local/include/jpeg*

Every optional graphic format requires, for its inclusion, the use of the define ***i<format_name>*** (so you can use: iRLE, iJPEG, iTIFF and iSGI)

when compiling. As can be seen in the Makefile listing, and because of the machine on wich it is supposed, we're working on they are all defined.

The definition of each format requires its headers placed (it's not supposed they are placed in the default directory /usr/include). If we look again the example we'll see how each one has its own place such as ***/usr/local/include/<format_name>*** and this is included during the compilation process with the ***-I*** option for each library.

In our example we had:

```
-DiRLE          -I/usr/local/include/rle
-DiJPEG         -I/usr/local/include/jpeg
-DiTIFF         -I/usr/local/include/tiff
-DiSGI
```

Well, now that all the supported graphic formats are already defined, let's go with the viewers we want to include.

We can include viewers for:

- X11: all machines with the Xlib library installed (the most standard), it hasbeen tested undel releases 4 and 5.
- GL: a new rising standard. It is included in SGI and IBM machines and there are versions being developed for DEC, Sun and NT. It will be called Open GL in a nearfuture and some instructions will have to change its name.
- Starbase: graphics library of the hp9000 machines (all series). It also supports X11.
- PC: here it is used the GRX library distributed as a contribution with the GCC compiler by GNU (PC version).
- Linux: a viewer for the text mode of this OS. I used the SVGA and SVGA_GL libraries, so a lot of boards are supported.

The format is the same, define***i<architecture>*** (following the instructions included in the *Makefile*) and, if needed, the path to reach the headers.

Our hipotetic machine is a Silicon Graphics with X11 (placed in a no standard place), with GL and the appropiated hardware to control a VCR. So we define:

```
-DiGL -DiVFR -DiX11
-I/usr/include/gl -I/usr/video/vfr/src/inc -I/usr/local/X11R5/include
```

We have already configured the ***Makefile*** file. Now there is only left to do the header ***include/lugconf.h*** to be ready. Continuing with the example of our hipotetic machine, we'll have:

```
/*
 * If your machine is not bigendian then
 * you'll need define the NO_BIGENDIAN macro.
 *
```

```
 * The DEC and PC machines needs this.
 *
 * If your compiler defines the MSDOS macro ( ex: the
 * djgcc ), the definition of BIG_ENDIAN is done
 * automaticly.
 *
 */
#ifdef MSDOS
#       define NO_BIGENDIAN
#endif


/*
 * If you has an ANSI compiler ( ex: gcc ), define
 * the USE_PROTOTYPES macro.
 */
#define USE_PROTOTYPES

/*
 * Define the USE_STDLIB_H macro if your machine has
 * the stdlib.h header.
 */
#define USE_STDLIB_H


/*
 * Define the VOID_STAR macro if your malloc definition
 * needs void* instead of char*.
 *
 * ( Only used if USE_STDLIB_H hasn't been defined. )
 */
#ifndef USE_STDLIB_H
#       define VOID_STAR
#endif


/*
 * Define USE_STRINGS if you have the strings.h
 * header ( and not the standard string.h ).
 */
#undef USE_STRINGS


/*
 * If your library has bcopy, bzero, index, ...
 * then you'll need define BSD ( else LUG will use
 * memcpy, memset, ... ).
 */
#define BSD

/*
 * If you have installed the GNU's GZIP and you wanna
 * use it instead of COMPRESS then define the USE_GNU_GZIP
 * macro.
 */
#define USE_GNU_GZIP
```

The first of all is (BIGENDIAN), will configure LUG for those machines which put into memory the least significative byte before the most significative. For example: the number 12345, should be stored in a bigendian machine as 3039, while the non-bigendian should store it as

3930. Within the latter we have PC machines (INTEL processors) and DEC (VMS or ULTRIX).

With USE_PROTOTYPES we will instruct the compiler to accept ANSI especifications like:

```
main( int argc, char **argv )
{
        ...
}
```

against traditional:

```
main( argc, argv )
int argc;
char **argv;
{
        ...
}
```

Depending of the existence or absence of the headers you will have to define, or not, USE_STDLIB_H y USE_STRINGS

If the compiler supports pointers as void* you will have to define VOID_STAR.

If the C library has the *bcopy*, *bzero*, *index* and *rindex* then a BSD chain has to be defined. If not, mcpy, memset, etc... traductions will be used.

At last (but not least :-)), if the machine has the *gzip* compressor, from *GNU*, and you want to use it instead *compress*, if neccessary, you will have to define the USE_GNU_GZIP chain (caution: this program is not part of the system and has to be installed apart).

Continuing with the example: Silicon Graphics are bigendians, allow using prototipes, have the *stdlib.h* header but not the *string.h*, the *void** are supported, it is a BSD system and we want to use *gzip*.

Well, at this point we have everything we need to compile LUG.

We go to the root distribution directory and execute the make command. After a few minutes we will obtain the liblug.a ready to be used.

There is only left to install it where we want to keep it (this is completely optional). From this moment the manual will suppose that all the headers (all those that are into the *include/* directory) are installed in */usr/local/include/lug* and the library is in */usr/local/lib*. For this process [install] we can use copies:

```
$ mkdir /usr/local/include/lug
$ cp include/* /usr/local/include/lug
$ cp liblug.a /usr/local/lib
```

Of course, it would be interesting reading the complete manual before using what we have installed.

Conclusion

With all the comment included in the config files, was this chapter really neccesary?.

Compiling with LUG

Introduction

In this chapter we'll provide you a guide of how to compile the programs using LUG.

It will be assumed a low knowledge of UNIX programming so, for those who have it only remind them that they have to specify *-llug* (-L if it is not in an standard place, in our example *-L/usr/local/lib*) and where are installed the headers (*-I/usr/local/include/lug*).

What is neccessary to compile?

Everything depends upon two things:

- what uses the program we are compiling
- and what are we including in the moment of the LUG compilation

If what we are compiling is a program which does not use any external library (that is: any format which needs using an external library or viewer), then we will be able to compile it with the math library.

Let's suppose the example What is LUG?:

```
#include <lug.h>
#include <lugfnts.h>
#define NO_BLUR                 0

main( int argc, char **argv )
{
        if ( argc < 2 ) {
                /* Ooopps!, not enough arguments */
```

```
                fprintf(stderr, "Usage: %s <bitmap>\n", argv[0]);
                exit( 1 );
        }else {
                bitmap_hdr in, out;

                /* Read the input file */
                read_gif_file( argv[1], &in );
                /* Resample the image */
                adjust_bitmap( &in, &out, in.xsize/2, in.ysize/2,
                        NO_BLUR );
                /* Write the resampled image */
                write_gif_file( argv[1], &out );
        }
        exit( 0 );
}
```

We only use here the read/write and reescaling GIF routines, all of them from LUG. So, if we suppose the program is called halfgif.c, we would compile it with:

```
$ cc -I/usr/local/include/lug -L/usr/local/lib -O -o halfgif halfgif.c -llug -lm
```

Step by step:

- we have already saw that the headers were placed in */usr/local/include/lug* and the library in */usr/local/lib*. With the *-I* and *-L* options we specify both paths.
- the -O option is used for optimization (see man pages of cc)
- -o halfgif, so the compiler generates a program with that name
- and, at last, the libraries to include, that is the *lug* and *math* library

At this moment we will have an executable program which will reescale all the stored images in the GIF format to its half.

Now, let's suppose that we want to use TIFF (of course, LUG has to be compiled with that option).The program, Halftiff, could be something like:

```
#include <lug.h>
#include <lugfnts.h>
#define NO_BLUR                 0

main( int argc, char **argv )
{
        if ( argc < 2 ) {
                /* Ooopps!, not enough arguments */
                fprintf( stderr, "Usage: %s <bitmap>\n", argv[0] );
                exit( 1 );
        }else {
                bitmap_hdr in, out;

                /* Read the input file */
                read_tiff_file( argv[1], &in );
                /* Resample the image */
                adjust_bitmap( &in, &out, in.xsize/2, in.ysize/2,
                        NO_BLUR );
```

```
                    /* Write the resampled image */
                    write_tiff_file( argv[1], &out );
                }
                exit( 0 );
}
```

If we compile it in the same way, we would obtain a error series like:

```
tiff.o: Undefined symbol _TIFFOpen referenced from text segment
tiff.o: Undefined symbol _TIFFFdOpen referenced from text segment
tiff.o: Undefined symbol _TIFFGetField referenced from text segment
tiff.o: Undefined symbol _TIFFGetField referenced from text segment
tiff.o: Undefined symbol _TIFFGetField referenced from text segment
tiff.o: Undefined symbol _TIFFGetField referenced from text segment
tiff.o: Undefined symbol _TIFFGetField referenced from text segment
tiff.o: Undefined symbol _TIFFGetField referenced from text segment
tiff.o: Undefined symbol _TIFFPrintDirectory referenced from text segment
tiff.o: Undefined symbol _TIFFGetField referenced from text segment
tiff.o: Undefined symbol _TIFFScanlineSize referenced from text segment
tiff.o: Undefined symbol _TIFFReadScanline referenced from text segment
tiff.o: Undefined symbol _TIFFClose referenced from text segment
tiff.o: Undefined symbol _TIFFOpen referenced from text segment
tiff.o: Undefined symbol _TIFFFdOpen referenced from text segment
tiff.o: Undefined symbol _TIFFSetField referenced from text segment
tiff.o: Undefined symbol _TIFFSetField referenced from text segment
tiff.o: Undefined symbol _TIFFSetField referenced from text segment
tiff.o: Undefined symbol _TIFFSetField referenced from text segment
tiff.o: Undefined symbol _TIFFSetField referenced from text segment
tiff.o: Undefined symbol _TIFFSetField referenced from text segment
tiff.o: Undefined symbol _TIFFSetField referenced from text segment
tiff.o: Undefined symbol _TIFFSetField referenced from text segment
tiff.o: Undefined symbol _TIFFSetField referenced from text segment
tiff.o: More undefined symbol _TIFFSetField refs follow
tiff.o: Undefined symbol _TIFFWriteScanline referenced from text segment
tiff.o: Undefined symbol _TIFFClose referenced from text segment
tiff.o: Undefined symbol _TIFFClose referenced from text segment
```

What's the problem?.The problem is that the TIFF interface (tiff.o) is using an external library which wasn't included in the linkage process.

The correct compilation instruction would be:

```
$ cc -I/usr/local/include/lug -L/usr/local/lib -O -ohalftiff halftiff.c -llug -ltiff -lm
```

So, if any process refencing to an external library is included, those libraries should be included at compiling time. They are:

- references to the formats: TIFF (-ltiff), RLE (-lrle), rgb from SGI (-limage) and JPEG (-ljpeg).
- to any viewer: X11 (-lX11), Starbase (-ldld -lsb), GL (-lgl or the shared -lgl_s), vfr (-lvfr), PC (-lgrx) and Linux (-lvgagl -lvga).

For the last ones it is recommended to re-read the appropiate manuals to see any suitable info.

Universal Script

To make easy the compilation of any program under LUG and as only functions of the referenced libraries are included, we can build a shell script to compile everything we give to it.

Let's suppose an IBM R6000 machine, with X11 and GL installed, and with all the auxiliar libraries installed in the */usr/local/lib.*

The script (we would call it clug), is:

```
PROGRAM=$1
shift
INCLUDES=-I/usr/local/include/lug
LIBRARIES=-L/usr/local/lib -llug -ltiff -ljpeg -lrle -lgl -lX11
cc -O $INCLUDES -o $PROGRAM $PROGRAM.c $* $LIBRARIES -lm
```

We give to it execution permission:

```
chmod 755 clug
```

and now we can compile every program using LUG with *clug*. For example, the two programs we have already seen could be converted to executables with:

```
$clug halfgif
$clug halftiff
```

Inside LUG

Introduction

In this chapter we will give a description of the internal structure of the library and its working and implentation philosophy.

What is a library?

Of course, for someone with a minimal programation knowledge, this has an obvious answer, but... I had to choose an starting point.

Briefly, libraries are a group of subroutines which make easy the programmers work giving him high abstraction levels.

For example: routines like *fprintf* and *fscanf* are part of the *libc* library, don't exist that way in the machine. Probably, the computer will only understand read and write. If we think about it, what would happen without libraries?, we would have to do everything (do you remmber?).

LUG joins a series of read/write and processing images routines that simplify the work as much as possible.

Philosophy of the library

The main goal of the library is achieving the maximum speed in every routine.Only sharpen, for didactic purposes, was implemented using theorethical descriptions based on a previous HSL process over the

image in order to manage the intensity.

Why?. Ok, treating 200 or 300 images can not done with a program which takes several minutes in doing anything, when a similar result can be obtained in half a minute or less.

Then, first of all, SPEED.

Second, supporting external formats implies a common link beetwen all of them. Therefore, we need to represent the images in the easiest way we can do its manage -in C- would not cause any trouble.

We choosed an extremely simple structure in which only basic data is represented, this is:

- size of the image
- raw information of the bitmap
- depth
- and, if neccessary, a color map

in C:

```
typedef struct {
        int xsize, ysize;       /* sizes */
        int depth;              /* # of colors */
        int colors;             /* # of colors */
        byte *r, *g, *b;        /* components or bitmap (planes < 8) */
        byte *cmap;             /* cmap if planes < 8 */
        int magic;              /* used ? */
} bitmap_hdr;
```

By fields:

- *xsize* and *ysize* are used to store the size in each axis, with the begginning of both in the upper left corner.
- *depth* and *colors* will represent the number of colours. The former does it by planes and the latter by number. For example, an image could use 8 planes for its representation but only use 222 of the possible 256.
- *r*, *g* and *b* store the raw information of the image. One byte for pixel and the r buffer will be the only one active if the image has less than 24 planes.
- *cmap*: array which represents the colour map of the image (if neccessary). Its storing will be done in secuences of r g b values (that is *rgb* of first colour, *rgb* of second, and so on...).
- *magic*: it is very important making this field the same as LUGUSED when this structure stores an image, if not, none of the routines we could use will be recognized. With this I try to avoid the use of empty or incorrect structures -this is, air pointers-.

Internal estructure

Four different parts can be found in LUG:

i. graphic format interfaces (/convert directory)
ii. functions of internal libraries for LUG (/lib directory)
iii. viewers for many architectures (/show directory)
iv. digital treatment functions (/utils directory)

I tried to isolate one of each other -except, of course, the functions of the internal library-. Only viewers use color reduction processes to manage some images.

Graphic format interfaces

Introduction

In this chapter it will be explained the way of use of the read/write routines for the supported graphic formats.

Remember that it's neccesary to use external libraries so the support of some formats could be effective, we will suyppose that they are all installed.

Supported formats

LUG, with its full configuration, supports:

- **GIF**: Graphics Interface Format, property of Compuserve. It's a PC standar due to its compression method, LZW. It supports any deep to a maximum 8 planes depth (256 colours).
- **HEIGHTFIELD**: It is not what we'd call a graphic format, it's more like a 3D object description, and it is a input for *Rayshade*. It is included in LUG because of its great use in representing DEMs.
- **JPEG/JFIF**: Joint Photography Experts Group / JFIF file interchange format. It is becoming an standard to store real images (with MPEG to animated sequences). It's a must due to its extremely high compression rates.
- **PBM/PGM/PPM:** It's the easiest and most common format among UNIX systems. Its beginning can be found in the PBM

Plus package by Jef Poskanzer. It will be the chosen format for any program which wants to be connected to the rest of the world without working too much in its interfaces (and without using any library).

- **PCX**: Zsoft format, it's extended because it was the only format used in their programs. It was difficult there would be a worst designed format in the world.It's written for the existing P.C. at that time and they did not even think about the possibility of better machines -well, Microsoft also did this with MS DOS-.

- **ALIAS "pix"**: Format used by Alias Research. Very simple, just to cover their needs.

- **POSTCRIPT**: What can I say about postcript? It's from Aldus and only is supported for output.

- **raw and rgb**: We put both in the same group because of their similarity. They are basically the raster info from the image. It's very used in vision departments of several Universities and for investigation facilities.

- **RLA**: Format property of Wavefront Technologies and used by its products. It has an spectacular header but only supports 24 bitplanes

- **RLE**: Despite its generic name, we only refer to the RLE from Utah Raster Toolkit, by Spencer W. Thomas. The use of this format by Rayshade has made it very popular in laboratories where people work with computer graphics.

- **RGB from SGI**: It's only used on SGI machines, so it has no interest in other architectures.

- **TARGA**: It has been the standard for a long time to store high quality images (24 bitplanes), altough it also supports 8 and less bitplanes images.

- **TIFF**: Property of Aldus and, in few words, *the mother of all formats*. It includes everything: any compression proccess (Pack Bits, Fax, LZW, JPEG,...), any kind of info (image maker, any format resolution, colour adjustment rules...), any colour space (RGB, CMYK,...) additional fields for any user need. And all of this with a easy way of ampliation due to it's TAG implementation.

Input

If we re-read the example given in the introduction we could get an idea about how reads (and writes) formats LUG.

The complexity of the task can be put in a single line:

```
read_<file_format>_file( char *name, bitmap_hdr *bitmap )
```

where *<file_format>* is one of the next: gif, jpeg, pbm, pcx, alias, rla, sgi, tga or tiff.

For example: to read an Alias file and store it as TIFF, we'd build a program like:

```
#include <lug.h>
#include <lugfnts.h>

main( argc, argv )
int argc;
char **argv;
{
        bitmap_hdr bitmap;

        read_alias_file( argv[1], &bitmap );
        write_tiff_file( argv[2], &bitmap );
}
```

For the gif, jpeg, pbm, pcx, alias, rle, rla and tga there are, also, functions like:

```
read_<file_format>( FILE *handle, bitmap_hdr *bitmap )
```

where the reading is made by the file descriptor (not its name).

This has its extension and great utility with the pbm, alias, rle and tga formats, where reading through standard reading is allowed (this could have been implemented for any format but it'd have been neccesary to load the WHOLE file in main memory and treat it before, so the memory use would be much higher, and that is why it was ignored).

Let's suppose we have a program, for example Rayshade, which creates, on the standard output an RLE file. But what we need is a image in TIFF file to include it in our paper. We'd build a program like:

```
#include <lug.h>
#include <lugfnts.h>

main( argc, argv )
int argc;
char **argv;
{
        bitmap_hdr bitmap;
```

```
        read_rle( stdin, &bitmap );
        write_tiff_file( argv[1], &bitmap );
}
```

If we call it rle2tiff, we could execute rayshade in this way:

```
rayshade -v projector.ray | rle2tiff projector.tif
```

Anyway, if we prefer to use the *_file routines (?!), we can do it replacing the file name with NULL (of course, only supported in the former file formats and TIFF). The program would stay like:

```
#include <lug.h>
#include <lugfnts.h>

main( argc, argv )
int argc;
char **argv;
{
        bitmap_hdr bitmap;

        read_rle_file( NULL, &bitmap );
        write_tiff_file( argv[1], &bitmap );
}
```

Now, the image:

There is only left to comment the raw and rgb routines. The need of specifying the file size and, with the raw format, three files (or handles) change the format of their routines. These are:

```
read_rgb_file( char *name, bitmap_hdr *bitmap, int xsize, int ysize )
```

reads a file stored in rgb format (rgb from the first pixel of the first row, rgb from the second...) and a size of *xsize*ysize*.

```
read_rgb_i_file( char *name, bitmap_hdr *bitmap )
```

as the other, but the image size is asked directly to the user through the standard input/output,

```
read_24bitmap( int xsize, int ysize,
                FILE *rhandle, FILE *ghandle, FILE *bhandle,
                bitmap_hdr *bitmap )
```

reads an *xsize*ysize* size image, stored in three independent files,

```
read_raw_file( char *name, bitmap_hdr *bitmap )
```

reads an image in graytones. As before, the user is asked about the image size through the standard input/output,

```
read_8bitmap_file( char *name, bitmap_hdr *bitmap, int xsize, int ysize )
```

as before, but now, we have already defined an *xsize*ysize* size,

```
read8bitmap( FILE *handle, bitmap_hdr *bitmap, int xsize, int ysize )
```

the only difference is that the file is specified through its handle.

All these routines acept their use through the standard input (using NULL or stdin when needed). Let's see an example of this: if we were in need of a program to change 512x512 grayscale images to GIF with a program which implements the FFT to see them in our PC, we would write:

```
#include <lug.h>
#include <lugfnts.h>

main( argc, argv )
int argc;
char **argv;
{
        bitmap_hdr bitmap;

        read_8bitmap_file( argv[1], &bitmap, 512, 512 );
        write_gif_file( argv[2], &bitmap );
}
```

Output

Few things can be added to what we have already seen, even writing routines was used before.

The *complex* routine is now:

```
write_<file_format>_file( char *name, bitmap_hdr *bitmap )
```

being *<file_format>* valid for the gif, hf, jpeg, pbm, pcx, alias, ps, raw, rgb, rla, rle, sgi, tga and tiff -yeah, that it, ALL-.

Besides, we have its companion:

```
write_<file_format>( FILE *handle, bitmap_hdr *bitmap )
```

for gif, jpeg, pbm, pcx, alias, raw, rgb, rla, rle and tga.

They accept writing to the standard output (with NULL or stdout) for all formats excepting rla.

The postcript interface can be called, also, through the routine:

```
write_ps_file_dimensions( char *name, bitmap_hdr *bitmap,
                          double width, double height )
```

where width and height represent the width and height (sic), in centimetres, we want to give to the image in the page (default is *19.0* and *27.0*)

For the heightfield writing there are another two routines (besides *write_hf_file*) which are:

```
write_hf_file_scale( char *name, bitmap_hdr *bitmap, double scale )
write_hf_scale( FILE *handle, bitmap_hdr *bitmap, double scale )
```

we can specify the desired height for the heightfields (default is *0.5*).

As for the raw/rgb writing routines, there is only left to show:

```
write_24bitmap( FILE *rhandle, FILE *ghandle, FILE *bhandle,
                bitmap_hdr *bitmap )
```

At last, and as we have not seen any examples yet, let's go with one. Let's suppose we want to convert the images generated by *The Advanced Visualizer*, from Wavefront, to PCX format to make an slide-show for PC. We'd write:

```
#include <lug.h>
#include <lugfnts.h>
#define NO_COLORS                       256


main( argc, argv )
int argc;
char **argv;
{
        bitmap_hdr in, out;

        read_rla_file( argv[1], &in );
        quantize( &in, &out, NO_COLORS);
        write_tiff_file( argv[2], &out );
}
```

As can be seen, beetwen reading and writing a quantification process is made -a reduction to 256 colors- so we obtain an image suitable for the PCX format (the truth is that the last version of this format supports full color puctures, but as there is no info about it, so LUG only support it in for reading not writing).

Conections

The only cautions you have to take when writing an image is with the depth limitations in every format. For instance: there would be an error if we try to store an image of 24 bitplanes in GIF format (which support no more than 8).

The limitations are:

- alias: only 8 or 24 bitplanes
- gif: no more than 8 bitplanes
- heightfield: raw gray tones
- jpeg: 24 bitplanes
- pbm: no limitations
- pcx: til 8 planes (24 bitplanes not supported)
- postcript: raw gray tones
- raw and rgb: no limitations
- rla: only 24 bitplanes
- rla: only 24 bitplanes
- sgi's rgb: only 24 bitplanes
- targa : only 24 bitmplanes (not supported less depth)
- tiff: no limitations

There is a subroutine, as we'll see in the next chapter, to manage in execution time this table.

There is left to the ***Digital Image Processing*** chapter the fitting of the images to the requeriments of the target format (quantification, dither,...).

Universal Reader/Writer

If it was easy to manage any format, now it will be easier.

We pretend to read/write any format with only one instruction to choose the appropiate interface and the neccessary conversions. Its sintaxis is:

```
read_lug_file(char *name, bitmap_hdr *bitmap)
write_lug_file(char *name, bitmap_hdr *bitmap)
```

As every format has its characteristic extension -for instance, any GIF image will end in .gif- the election of the interface acording to its extension. This is not a perfect solution, but I don't consider this as a

basis of LUG -it's more a curiosity-, and it's proposed to be improved by the user.

Well, the formats will be identified with the extensions:

- gif: .gif
- heightfield: .hf
- jpeg: .jpg / .jpeg
- pbm: .pgm / .pgm / .ppm
- pcx: .pcx
- alias: .pix / the default format
- postscript: .ps
- raw: .raw
- rgb: .rgb
- rla: .rla
- rle: .rle
- rgb de sgi: .sgi
- targa: .tga
- tiff: .tif / .tiff

As we have already said, if no extension is given, the choosen interface will be Alias, -because is our base format in the LabCAD-. This, of course, can be changed in compiling time of the liblug library, in the header lug.h (define DEFAULT_FORMAT).

One example: we need a universal translator from any to any format -not more nore less-, and we'd write a program *lugcnv.c* as:

```
#include <lug.h>
#include <lugfnts.h>

main(argc, argv)
int argc;
char **argv;
{
        bitmap_hdr in;

        if ( argc < 3 ) {
                fprintf( stderr, "Usage: %s <input_file> <output_file>\n",
                        argv[0] );
                exit( 1 );
        }

        /* Read the file ... */
        read_lug_file( argv[1], &in );

        /* ... and write it. */
        write_lug_file( argv[2], &in );

        /* But, but ... bye !?. */
        exit( 0 );
}
```

We'll only have to take care with the file extensions and LUG will manage the bitplane depth conversions, reading and writing. Very easy, isn't it?

It's been already commented in a previous paragraph that there were a routine to know the possibilities of any graphic format, it is:

```
get_depth_writelug_function( char *file_name )
```

It will return, depending upon the format provided: ONLY_BINARIES_PLANES (one bit images), ONLY_RAW_PLANES (grayscale, 0..255), ONLY_8_PLANES (colour, maped 8 bitplanes), ONLY_24_PLANES and ALL_PLANES (any depth).

Other interesting routines are:

```
ifunptr get_readlug_function( char *file_name )
ifunptr get_writelug_function( char *file_name )
```

they will return to us the neccesary routine, to read or write, the given format.

For instance: if we want to build a program to translate one format to any other different format (always with the same depth) we would write:

```
#include <lug.h>
#include <lugfnts.h>

main(argc, argv)
int argc;
char **argv;
{
        bitmap_hdr bitmap;
        ifunptr read_ptr, write_ptr;

        if ( argc < 3 ) {
                fprintf( stderr, "Usage: %s <input_file> <output_file>\n",
                        argv[0] );
                exit( 1 );
        }

        /* Get the correct function and read the input file */
        read_ptr = get_readlug_function( argv[1] );
        read_ptr( argv[1], &bitmap );

        /* Get the supported depths of the target format */
        new_depth = get_depth_writelug_function( argv[2] );
        /* All ok ? */
        if ( !new_depth || bitmap.depth == new_depth ) {
                /* Yeah!. Lets go ... */
                write_ptr = get_writelug_function( argv[2] );
                write_ptr( argv[2], &bitmap );
                exit( 0 );
        }else {
                /* Ooooopppss!!! */
                fprintf( stderr, "No correct depths\n" );
                exit( 1 );
        }
}
```

Conclusion

As we have seen, it has been intended to create a series of routines to help the potential user in reading and writing graphic formats without taking care of anything. I think these is something that has been achieved.

Of course, some graphic formats may be missed, but there are the basic ones. I invite to any user to add all format he considers of interest, they will be welcome.

Internal library functions

Introduction

In this chapter several routines implemented by LUG are explained to give service to the rest of the library itself and to help the user in its work.

Why?

There is a simple reason, let's look the *fopen* routine.

It's known that its sintaxis is

```
FILE *fopen( char *file_name, char *type )
```

where NULL will be returned if the file could not be opened -of course this has to be checked in each call to avoid any fatal error-.

LUG implements:

```
FILE *Fopen( char *file_name, char *type )
```

with the same sintaxis, but many interesting peculiarities. First, it checks internally a correct opening, and second, if the file is compressed, it decompresses it in a temporal file and after this it returns a pointer to it. We mean, code lines as:

```
if ((handle = fopen("the_input_file", "r")) == NULL ) {
        fprintf( stderr, "Cannot open the input file\n");
        exit( 1 );
}
```

will be transformed into:

```
Fopen( "the_input_file", "r" );
```

making the decompression of the file if neccesary.

Subroutines

We put together some general purpose routines here:

- **representation conversion**

  ```
  short *bytetoshort( byte *in, short *out, int size )
  byte *shorttobyte( short *in, byte*out, int size )
  float *bytetofloat( byte *in, float *out, int size )
  byte *floattobyte( float *in, byte*out, int size )
  ```

  If we want the routine to create the storing buffer -this is, not providing it- NULL must be given in the out field.

  The conversion routines from/to float suppose we are in a [0..1] environment.

- **change from character to number**: they are similar to the existing ones. They are:

  ```
  int Atoi( char *str )
  double Atod( char *str )
  ```

  Also, there is the comprobation of it is a chain or a number

  ```
  int isnumber( char *str )
  ```

- **compression/decompression**: (de)compress a file. They use *gzip* or *compress* depending upon the provided parameters in compiling. They are:

  ```
  uncompress( char *input_name, char *temp_name )
  compress( char *name )
  rm_compress()
  ```

  Although the uncompress routineis thought to be used from *fopen*, it can be called by user directly. Its philosophy is to decompress *input_name* over a temporal file, *temp_name*, in that way the original information is never lost.It's neccesary to call *rm_compress()* before exiting any program which calls fopen (this erases any open temporal files that would have been created).

- **others**: we put two general ones in this group. They are:

  ```
  char *basename( char *string, char *suffix )
  ```

  which works as the UNIX command with the same name. This is, the command:

  ```
  filename = basename( "pfeiffer.gif", ".gif" );
  ```

  would assignto filename the chain *pfeiffer*.

  ```
  int no_bits( int colors )
  ```

  returns the number of bits (less one) necessary to represent the number of colours desired. I mean, if we give *234* as argument, it would return *7* (*7+1=8*, number of planes we need to represent the especified number of colours).

Input / Ouptut

It will be explained here some of the routines implemented to aid the user in the I/O tasks:

- **opening/closing**:

  ```
  FILE *Fopen( char *filename, char *type )
  ```

  At the beginning of this chapter we saw the advantages of decompresion and opening made by it -so I'll not mention it again-.

  To make it simpler and not checking the presence of stdin/stdout/ stderr in the handle values, I implemented:

  ```
  int Fclose( FILE *handle )
  ```

- **existence of a file**:

  ```
  exist_file( char *name )
  ```

  It'll return:

  → FILE_NO_EXIST: oppsss!.
  → FILE_EXIST: great!, the file exists.
  → FILE_IS_GZIP: the file name exists but it's compressed with *gzip*.

→ FILE_IS_COMPRESS: the file name exists but it's compressed with *compress*.
→ FILE_WITH_GZIP: the file name doesn't exist, but there is a *<name>.gz* file.
→ FILE_WITH_COMPRESS: the file name doesn't exist, but there is a *<name>.Z* file.

- **reading of a file as one block**

```
char *read_file( FILE *handle, int *no_bytes )
```

since the current file position it reads it until its end and returns *no_bytes* (it is an output and is not an argument) in a internal buffer. The management of the standard input (pipes) is supported.

Although is not for reading, it is interesting the following routine:

```
long filelen( FILE *handle )
```

which returns the number of bytes left for reading in the file. That is, if we use it inmediatly after opening the file it will return the file size.

- **read/write by bytes**: used to read/write numbers from the Most Significative Byte First to the least one (and the viceversa Less Signaficative Byte First)

```
putshortMSBF( int value, FILE *handle )
putshortLSBF( int value, FILE *handle )
int getshortMSBF( FILE *handle )
int getshortLSBF( FILE *handle )
```

Memory

Some functions to manage images and memory.

- **memory management**:

```
char *Malloc( int size )
```

the same as malloc but it checks internally the memory asignment, and if affirmative, it returns it initialized to zeroes.

```
Free( void *ptr )
```

it takes in order the possibility of assigning a NULL pointer to free -all of us had this problem some time-.

- **image management**: this are already related to LUG and image processing. They are:

```
allocatebitmap( bitmap_hdr *image, int xsize, int ysize,
                int depth, int colors )
```

returns an image ready to use, with all the buffers allocated and its header with the appropiate values.

```
freebitmap( bitmap_hdr *image )
```

frees all the buffer and stores the image header.

```
copy_bitmap( bitmap_hdr *source, bitmap_hdr *target )
```

copies the first image in the second structure. If the second is already an image, all its contents will be freed before the duplication.

Conclusion

As you may already noticed, most of thes functions have nothing to do with images, but it's use simplifies the programation -at least with LUG they did-.

Viewers

Introduction

The image processing usually has associated the intention of viewing them. So, LUG implements six viewers for as many architectures.

I think there is only left a specific one for Sun machines, but I haven't had access to one of them and programming for a machine I will not be able to check if it works properly is, let's say, too much *conceptual*. So, til someone makes it -please!-, *Sun Microsystem* users will have to use the X11 viewer.

HP9000

Viewer for any *Hewlett Packard 9000* machines, it is based on the *Starbase* library and it has been checked with the *HP-UX* releases 8.0 and 9.0.

Its sintax is:

```
show_bitmap( char *name, bitmap_hdr *bitmap, int ditherflag )
```

where

- *name*: is the name of the image (same as the others for compatibility purposes).
- *bitmap*: the image
- *ditherflag*: if it is *true*, and neccesary, it will be used quantification instead dither to reduce the number of colours.

It recognizes any hardware graphic interface automatically and the user only has to define the environment variable SB_OUTDEV if the graphic display depicted by device is other than /dev/crt.

PC

It's sintax is:

```
show_bitmap( char *name, bitmap_hdr *bitmap, int ditherflag )
```

it uses the GRX library functions provided as contribution with the DJGPP (GCC) compiler for PC.

I think neccesary to remind the importance of defining the GO32 variable so the graphic card driver can be loaded (this is of course, explainend in the compiler docs). For instance: I have a *Trident* graphic card with its driver located in *d:/c/djgpp/drivers* and I want LUG working in *1024X768X256* colours, so in my *autoexec.bat* I will put the following line:

```
set GO32=driver d:/c/djgpp/drivers/trid8900.grn gw 1024 gh 768 nc 256
```

The last version of *go32.exe* extender (v. 1.11) includes a internal VESA driver so you can define:

```
set GO32=driver ::p gw 1024 gh 768 nc 256
```

SGI / IBM r6000

Viewer using the *GL* library.

Again, it's sintax is:

```
show_bitmap( char *name, bitmap_hdr *bitmap, int ditherflag )
```

where *dither_flag* is kept for compatibility purposes, because it always works in 24 bitplanes (if the hardware has not 24 bit planes it will simulate them)

This routine will be soon implemented for *OpenGL*.

VFR

This is an special viewer because it's specifically designed for video output, commonly known as Video Framers, found in *Silicon Graphics* computers.

It uses the *libvfr* library, it needs dedicated hardware and a certain awareness of the user when choosing a correct video format.

Its sintax is:

```
write_vfr( bitmap_hdr *bitmap, int resetVFR, char *VFRbuffername )
```

where:

- *bitmap*: is the image
- *resetVFR*: if true will reset the video framer before its use
- *VFRbuffername*:it is the name of the video system that will be used to show the image (*rgb*, *r_y*, *D1*, *D2*, *pal*, ...). In compiling time it is possible to change the default one if this field is NULL (in the original distribution *r_y_625*). To choose the appropiate value of this field it is neccesary to take a look at the **VFR Programmers Manual**.

It's a very simple routine (it only shows images) and if the user wants a complete management of the video it would be interesting to upload ***tovcr***, a LUG derived program and it will offer the required control.

X11

Viewer programmed with the libX library and, therefore, can be used by any computer with X11 -is there anyone without it installed yet?!-.

It has been checked under the X11 releases 4 and 5 and with eight bitplane screens (no 24 planes).

Its sintax is:

```
show_bitmap_x11( char *name, bitmap_hdr *bitmap )
```

The function adds _x11 because it can co-exist with any of the other viewers (for example: in a HP9000 you can have the Starbase viewer and this one). What's more, we have override the use of dither_flag because

of this method [dither] will be used to obtain a more friendliness with the other windows.

We have not had an example for a long time so here is one: let's suppose we don't want to do a dither, we want a colour quantification for the 24 bitplanes, we would built the following program:

```
#include <lug.h>
#include <lugfnts.h>

main(argc, argv)
int argc;
char **argv;
{
        bitmap_hdr in;
        ifunptr read_file;

        /* Get the input file */
        read_file = get_readlug_function( argv[i] );
        read_file( argv[i], &in );
        if ( in.depth > 8 ) {
                /*
                 * Ooooopppss!. We need quantize the image,
                 * we don't wanna a dithering on the screen
                 */
                bitmap_hdr aux;

                /* Quantize the in bitmap */
                quantize( &in, &aux, 256 );
                /* And show the new bitmap */
                show_bitmap_x11( argv[i], &aux );
        }else {
                /*
                 * All is ok.  Release the child and show
                 * the image.
                 */
                show_bitmap_x11( argv[i], &in );
        }

  exit( 0 );
}
```

Linux

With Linux you can work under two enviroments: X11 and text mode, so LUG implements two viewers :-).

Here we explain the text mode viewer. The sintaxis is:

```
show_bitmap( char *name, bitmap_hdr *bitmap, int ditherflag )
```

incredible, isn't it ?.

The subroutine was programmed using the VGA and VGA_GL libaries, so you'll need define the enviroment variable GSVGAMODE. An example:

```
export GSVGAMODE=G1024x768x256
```

I couldn't access to a true color board so I limited to 256 colors this viewer. It's too easy, so if you can do it, please modify the code to support these boards (and thanx!).

Digital Image Processing functions

Introduction

In this chapter we'll revise all digital image processing functions implemented by LUG. They will be revised one after another.

It's here where we notice that speed we explained a lot of pages before this one As we wanted speed we choose the spacial dominion management instead the frequential one because it is one of the ways to achieve that speed.

Functions

BLUR

Its syntax is:

```
blur_bitmap( bitmap_hdr *in, bitmap_hdr *out )
```

where *out* is the new image generated by the difumination of the first one, *in*. There is only one restriction and it's that it only can be aplied to true-color or gray-color images.

The blur matrix aplied is:

$$\begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix}$$

if we want to aply it to a data buffer, we can use:

```
byte *blur( byte *buffer, int zsize, int ysize )
```

It is the same, but with no restrictions. This has to be considered, because along this chapter it will be repeated continuously: a general routine and other for any buffer.

COLOR CHANGES

It allows changing one colour for another giving a percent of similarity:

```
change_color( bitmap_hdr *in, bitmap_hdr *out,
              int r, int g, int b,
              int new_r, int new_g, int new_b,
              int distance )
```

will replace *r, g, b* with *new_r, new_g* and *new_b* with an error of *distance*.

CHROMA

```
chroma_bitmap( bitmap_hdr *base, bitmap_hdr *super, bitmap_hdr *out )
```

It creates a new image which will combine *base* and *super*. This will be done as follows: everything in *super* greater than an specified umbral will be superposed to base. The CHROMABORDER define (lug.h) fixes this umbral. If we are not interested in a new image, but modifying the base itself, we'd use:

```
chroma( bitmap_hdr *base, bitmap_hdr *super )
```

In both cases, *base* and *super* will have to be the same size, if not we will have to use one of the following:

```
paste( bitmap_hdr *base, bitmap_hdr *super, int xpos, int ypos )
center_image( bitmap_hdr *base, bitmap_hdr *super )
```

where ypos, xpos are, in the first one, the upper left corner from which the image will be put. I think it is quite obvious that the second centers super over base -you know, sometimes, the name of the functions are useful-.

CONVOLUTION

Everyone involved in digital image processing will reckon that many of the tasks can be done through the use of a matrix, or mask, which can be aplied to all pixels of the image. This is what we call convolution.

Again, we have two routines:

```
convolve_bitmap( bitmap_hdr *in, bitmap_hdr *out, double matrix[9] )
byte *convolve( bitmap_hdr *in, bitmap_hdr *out,
                int xsize, int ysize, double matrix[9] )
```

in and *out* are the input and output images respectively. But what is really important here is the convolution routine. When blur was explained we saw how it defined a matrix and how it called these routines. As can be seen, the matrix is a *3x3 double* type one.

Three pages without an example!!!, let's end that: Suppose we want to extract the contours of a TIFF image, we'd do:

```
#include <lug.h>
#include <lugfnts.h>

main( argc, argv )
int argc;
char **argv;
{
        bitmap_hdr in, out;
        static double matrix[9]= {      -1.,    -1.,    -1.,
                                        -1.,     8.,    -1.,
                                        -1.,    -1.,    -1.  };

        read_tiff_file( argv[1], &in );
        convolve_bitmap( &in, &out, matrix );
        write_tiff_file( argv[2], &out );
}
```

CUT

```
cut_bitmap( bitmap_hdr *in, bitmap_hdr *out,
        int xini, int yini, int xsize, int ysize )
```

Cuts the image from *(xini, yini)* with a size *xsizexysize*. So, if we want to cut five pixels around a 512x512 image we'd write:

```
cut_bitmap( &in, &out, 5, 5, 502, 502 )
```

COLOR REDUCTION

There are four methods to reduce the number of colours in an image:

- quantize: It has been seen in previous examples

  ```
  quantize( bitmap_hdr *in, bitmap_hdr *out, int no_colors )
  ```

  It uses an statistical algorithm by Xiaolin Wu to select the *no_colors* more representative and to adjust the others to them.

- dither: The technic used here is based in the deceive that suffers the human eye with similar colours, that's why each colour is aproximated by an array of some other next to each other.

  ```
  dither_image( bitmap_hdr *in, bitmap_hdr *out,
          int levels, double gamma )
  ```

  Both values, besides the typical in and out, they are the maximum number of colours we can use and the gamma correction we will

aply to the colour table. The first one is obtained raising the number of levels, to the cube (if we have 8 spare bitplanes, we will give 6 to it, *6x6x6=218* levels), there is aroutine:

```
int compute_levels( int colors )
```

that will return the number of planes apropiated for the number of colours we give to it.

- grey tone: converts the image to grey tones using the NTSC conversion method

```
tobw( bitmap_hdr *in, bitmap_hdr *out )
```

- halftone: converts the image to one bit (black or white) through the halftone technic.

```
tohalftone( bitmap_hdr *in, bitmap_hdr *out )
```

GAMMA CORRECTION

It makes the typical gamma correction over the image we provide to the function. Its sintax is:

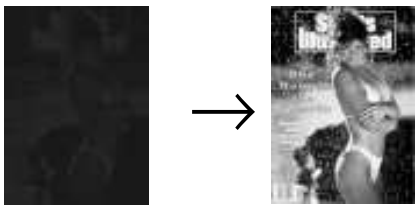
```
gamma_correct( bitmap_hdr *in, bitmap_hdr *out, double gamma )
```

where *gamma* is the new value of that correction in the image we provide the routine.

FILTERS

We'll put together to reconstruction methods: histogram equalization and median filter.

The first one is one of the most typical processes in digital reconstruction, it expands the histogram of an image spreading in a uniform way. So, in few words, if we have a dark (but not to much!) image with this technique we would see something.



The routine we have to execute is:

```
histogram_equalization( bitmap_hdr *in, bitmap_hdr *out )
```

The second method, the median filter, is used to eliminate the impulsional noise from an image:

```
median_filter( bitmap_hdr *in, bitmap_hdr *out )
byte *filternoise( byte *buffer, int xsize, int ysize )
```

The image must be true-color or gray-color (although the second one will aply it to any buffer which is passed to it). What it does is to move a 3x3 matrix over the image and substitutes each pixel for the medium value which corresponds.

FLIPS

We will group here both, vertical and horizontal flips, they are:

```
mirror_image( bitmap_image *in, bitmap_hdr *out )
byte *mirror( byte *buffer, int xsize, int ysize )
```

this makes an horizontal flip, the same effect we obtain from a mirror, and:

```
flip_image( bitmap_hdr *in, bitmap_hdr *out )
byte *flip( byte *buffer, int xsize, int ysize )
```

which makes a vertical flip, this makes the image to be upside down.

RGB/HSL CONVERSIONS

Some digital treatment algortihms, if not all, require the management of colours in the HSL norm. There are two functions to convert from/to HSL:

```
RGB_to_HSL( int r, int g, int b,
            double *h, double *s, double *l )
HSL_to_RGB( double h, double s, double l,
            byte *r, byte *g, byte *b )
```

For convenience, two other routines were implemented to pass complete buffers:

```
rgb_to_hsl_buffer( byte *r, byte *g, byte *b,
                   double *h, double *s, double *l,
                   int size )
hsl_to_rgb_buffer( double *h, double *s, double *l,
                   byte *r, byte *g, byte *b,
                   int size )
```

ROTATIONS

With these procedures we'll be able to rotate an image. The first one will rotate it any angle:

```
rotate_bitmap( bitmap_hdr *in, bitmap_hdr *out, double degrees )
```

It's something wich has not been very used in the LabCAD and, therefore, is one of the weakest routines included in LUG -if not the only one :-)-. It will be the first one in experimenting a reimplementation.

What is more usual, is rotating an image 90 degrees, for example, when an image is scanned horizontally when we intended to do it vertically.

```
rotate90_image( bitmap_hdr *in, bitmap_hdr *out )
byte *rotate90( byte *buffer, int xsize, int ysize )
```

SHARPEN

This is the unique routine which was not implemented to achieve the maximum possible speed. This was because didactical purposes, to show the use of the routines to/from HSL, because it is neccessary to use it in the improving algorithm. It is:

```
sharpen_bitmap( bitmap_hdr *in, bitmap_hdr *out )
```

In the last given examples -three pages before this one- it was defined a contour detection routine and a convolution matrix was given. If we change the *8.0* valour, in the central point, for a *9.0*, we'd have a sharpen routine for convolution over the r-g-b planes, directly.

SOLID IMAGES

If we want to create a true color image of only one color, we can use:

```
create_solid_image( bitmap_hdr *out, int xsize, int ysize,
                    int r, int g, int b )
```

easy, isn't it?

GRAY TONE MANAGEMENT

Although a method to convert from color to gray color images has already been seen:

```
tobw( bitmap_hdr *in, bitmap_hdr *out )
```

there are more functions, they are.

The first one is another routine that assures creating an intensity buffer image:

```
to_raw_bw( bitmap_hdr *in, bitmap_hdr *out )
```

the difference is that *tobw* only converts a color map if it is a mapped image, besides this will reconvert all the raw information. Anyway, to pass true-color it is the same:

```
byte *create_bw_pallete()
```

creates a 256 gray palette ready to be assigned to the cmap fields into the bitmap_hdr structure,

```
int RGB_to_BW( int r, int g, int b )
rgb_to_bw( byte *r, byte *g, byte *b, byte *outbuffer, int size )
```

converts a color map of the image (a gray scale one), only the map, and it returns it in a new buffer;

```
int RGB_to_BW( int r, int g, int b )
rgb_to_bw( byte *r, byte *g, byte *b, byte *outbuffer, int size )
```

both convert a(some) r-g-b valour(s) to gray tones, but while the first one returns the valour, the scond one puts it in a buffer (*outbuffer*). The last subroutine is:

```
int isagrayscaled( bitmap_hdr &bitmap )
```

will return TRUE if the image is a gray tone one and the 0 valour is for blackand 255 for white -is neccesary to talk about the mid valours?-

TRUE COLOR PASS

Because it is neccesary to transform an image to true colour in order to apply some processes -or gray tones-, there is a routine which does it:

```
to24( bitmap_hdr *in, bitmap_hdr *out )
```

...need more info?

NEGATIVE

Some times it is neccessary to make the negative of an image to look at the image details in a better way:

```
toinverse( bitmap_hdr *in, bitmap_hdr *out )
```

REESCALING

There are two routines to make anamorphic reescalings over images. The first one is:

```
adjust_bitmap( bitmap_hdr *in, bitmap_hdr *out,
               int newx, int newy, int noblur )
byte *zoom( int oldx, int oldy, int newx, int newy,
            byte *buffer, int noblur)
```

both make an image reescaling to the new given size. This technique does not use any interpolation process (remember, fast!), so, it uses a blur flag to improve this. It will be used only in case of the image being bigger and this flag, *noblur*, inactive this.

The second method is more accurate. It uses an interpolation method in a bell shape which surounds the current pixel, of course it is slower:

```
slow_adjust_bitmap( bitmap_hdr *in, bitmap_hdr *out,
                    int newx, int newy )
```

here is not a routine that manages the buffer because it needs some filters wich are precalculated before.

Conclussion

After this bunch of routines -more or lee ordered- probably the best thing we can do is to show an example that uses some functions:

```
/*
 * mural.c - builds a mural image from many others.
 *
 * Author:  Raul Rivero
 *          Mathematics Dept.
 *          University of Oviedo
 * Date:    Sat Jan 18 1992
 * Copyright (c) 1992, Raul Rivero
 *
 */

#include <lug.h>
#include <lugfnts.h>

/*
 * 730x576 * 1.75
 */
#define XSIZE                   1277
#define YSIZE                   1008

extern int LUGverbose;
extern char *MY_NAME;

main(argc, argv)
int argc;
char **argv;
{
  int i;
  bitmap_hdr base;
  bitmap_hdr inbitmap;
  bitmap_hdr outbitmap;
  int noblur = 0;
  int xsize, ysize;
  int xpos, ypos;
  int no_frames;

  MY_NAME = argv[0];

  if ( argc > 1 ) {             /* else core on SGI */
```

```
 while ( argv[1][0] == '-' ) {
   for ( i = 1; argv[1][i]; i++ ) {
            switch ( argv[1][i] ) {
              case 'v':
                        verbose++;
                        break;
              case 'n':
                        noblur++;
                        break;
              default :
                        error( 0 );
                        break;
            }
   }
   argv++;
   argc—;
  }
}

no_frames = argc - 1;
if ( no_frames < 1 )
  error( 0 );

/*
 * Number of times.
 */
i = 1;
while ( i*i < no_frames )
  i++;

/*
 * Size of each image.
 */
xsize = XSIZE / i;
ysize = YSIZE / i;

/*
 * Create the base bitmap.
 */
create_solid_image( &base, XSIZE, YSIZE, 0, 0, 0 );

/*
  * Begining of the first image.
 */
xpos = 0;
ypos = 0;

/*
 * Ok, lets go!
 */
for ( i = 1; i < argc; i++ ) {
  fprintf( stderr, " %d of %d\r", i, no_frames ), fflush( stderr );
   /*
    * Read the image.
    */
 read_alias_file( argv[i], &inbitmap );
   /*
     * We need an image of xsize x ysize pixels, so ...
    */
 adjust_bitmap( &inbitmap, &outbitmap, xsize, ysize, noblur );
   /*
     * Paste that resampled image from (xpos,ypos).
    */
```

```
    paste( &base, &outbitmap, xpos, ypos );
    /*
     * We don't need more these images.
     */
    freebitmap( &inbitmap );
    freebitmap( &outbitmap );
    /*
     * Get the orign of the next image.
     */
    xpos += xsize;
    if ( xpos + xsize > XSIZE ) {
      /* Next row */
      xpos = 0;
      ypos += ysize;
    }
  }

  /*
   * Write the file.
   */
  write_alias( stdout, &base );

  fprintf( stderr, "Output image has been created\n" );

  return(0);
}
```

It is intended to create an image with as many images in its input as desired.

If we use 4 images they will be grouped as a *2x2* image, if 9 then *3x3*, if 17 then *5x5*, etc. When the number of rows and columns (the same each other) is known we will know the image of each size, so the only thing left is to read the image, resampling to the proper size and putting them in the proper place, and voila!, it's done!!!.

Conclusion

Global variables

We're at the end, but first we must talk about a pair of things the reader should have detected in the last example.

Every LUG routine works in silence if there are no errors -if there are, they scream-. You can work in a verbose mode activating the proper int flag, ***LUGverbose***. If it is active each routine will show each step of its work through the error output.

If you wanna know the name of the routine it is currently working you must assign that name to the char variable ***MYNAME***. For example, and going back to the last example, if we try to read a non-Alias file, we'd obtain the message:

```
mural:Unknow file type
```

At last

Well, this is the end.-like the song :-)-

In the appendix sections a list of all the existing functions and a few more examples of the use of the library are given.

I hope that this manual will help the user to understand how it [LUG] works and it will clear every doubt a user may have. Anyway, the best thing a user can do is to use it and face the real problems involved in programming.

I wrote this manual in spanish (yea!, if you can read spanish, please, I'll allways maint a spanish version, so USED IT!) and it was tranlated by my friend Jorge Soto. I made the last review so insult me :-) if get some mistake.

I may have forgotten something in the explanations -it's very difficult to remember things that are supposed to be known-, due to their obviouness (I wrote them!), so if you have any doubt, suggestion, bug, complain... you can contact with me by E-mail at rivero@pinon.ccu.uniovi.es or by paper at:

Raúl Rivero
Vicerrectorado de Estudiantes
Universidad de Oviedo
Argüelles, 39, 2º
33003 - Oviedo
Spain

If you use LUG, please E-mail me, I would like track all the users (new releases, bugs, ...).

Of course, all new subroutines will be welcome.

Cheers.

Appendix A

In this appendix a complete list of the functions included in LUG will be given.

Graphic format interfaces:

```
read_gif_file( char *file_name, bitmap_hdr *bitmap )
read_gif( FILE *handle, bitmap_hdr *bitmap )
write_gif_file( char *file_name, bitmap_hdr *bitmap )
write_gif( FILE *handle, bitmap_hdr *bitmap )
write_hf_file( char *file_name, bitmap_hdr *bitmap )
write_hf( FILE *handle, bitmap_hdr *bitmap )
write_jpeg_file( char *name, bitmap_hdr *bitmap )
write_jpeg( FILE *handle, bitmap_hdr *bitmap )
read_jpeg_file( char *name, bitmap_hdr *bitmap )
read_jpeg( FILE *handle, bitmap_hdr *bitmap )
read_pbm_file( char *file_name, bitmap_hdr *bitmap )
read_pbm( FILE *handle, bitmap_hdr *bitmap )
write_pbm_file( char *file_name, bitmap_hdr *bitmap )
write_pbm( FILE *handle, bitmap_hdr *bitmap )
read_pcx_file( char *file_name, bitmap_hdr *bitmap )
read_pcx( FILE *handle, bitmap_hdr *bitmap )
write_pcx_file( char *file_name, bitmap_hdr *bitmap )
write_pcx( FILE *handle, bitmap_hdr *bitmap )
read_alias_file( char *file_name, bitmap_hdr *bitmap )
read_alias( FILE *handle, bitmap_hdr *bitmap )
write_alias_file( char *file_name, bitmap_hdr *bitmap )
write_alias( FILE *handle, bitmap_hdr *bitmap )
write_ps_file( char *name, bitmap_hdr *image )
write_ps_file_dimensions( char *name, bitmap_hdr *image,
                          double width_cms, double length_cms )
read24bitmap( char *file_name, bitmap_hdr *bitmap )
read8bitmap( FILE *handle, bitmap_hdr *bitmap )
write24bitmap( char *file_name, bitmap_hdr *bitmap )
write8bitmap( FILE *handle, bitmap_hdr *bitmap )
read_rgb_file( char *name, bitmap_hdr *bitmap, int xsize, int ysize )
read_rgb( FILE *handle, bitmap_hdr *bitmap, int xsize, int ysize )
write_rgb_file( char *name, bitmap_hdr *image )
write_rgb( FILE *handle, bitmap_hdr *bitmap )
read_rla_file( char *file_name, bitmap_hdr *bitmap )
read_rla( FILE *handle, bitmap_hdr *bitmap )
write_rla_file( char *file_name, bitmap_hdr *bitmap )
write_rla( FILE *handle, bitmap_hdr *bitmap )
```

```
read_rle_file( char *file_name, bitmap_hdr *bitmap )
read_rle( FILE *handle, bitmap_hdr *bitmap )
write_rle_file( char *file_name, bitmap_hdr *bitmap )
write_rle( FILE *handle, bitmap_hdr *bitmap )
read_sgi_file( char *file_name, bitmap_hdr *bitmap )
write_sgi_file( char *file_name, bitmap_hdr *bitmap )
read_tga_file( char *file_name, bitmap_hdr *bitmap )
read_tga( FILE *handle, bitmap_hdr *bitmap )
write_tga_file( char *file_name, bitmap_hdr *bitmap )
write_tga( FILE *handle, bitmap_hdr *bitmap )
write_tga_rle( FILE *handle, bitmap_hdr *bitmap )
read_tiff_file( char *file_name, bitmap_hdr *bitmap )
write_tiff_file( char *file_name, bitmap_hdr *bitmap )
```

Internal functions:

```
int Atoi( char *string )
double Atod( char *string )
allocatebitmap( bitmap_hdr *image, int xsize, int ysize,
                int depth, int colors )
char *basename( char *string )
float *bytetofloat( byte *source, float *target, int size )
short *bytetoshort( byte *source, short *target, int size )
Compress( char *name )
compute_levels( int colors )
copy_bitmap( bitmap_hdr *source, bitmap_hdr *target )
exist_file( char *name )
Fclose( FILE *handle )
long filelen( FILE *handle )
byte *floattobyte( float *source, byte *target, int size )
FILE *Fopen( char *name, char *mode )
Free( void *ptr )
freebitmap( bitmap_hdr *image )
isnumber( char *string )
char *Malloc( int size )
int no_bits( int colors )
char *read_file( FILE *handle, int *lenght )
rm_compress()
byte *shorttobyte( short *source, byte *target, int size )
Uncompress( char *name, char *tmp_file_name )
```

Viewers:

```
show_bitmap( char *name, bitmap_hdr *bitmap, int ditherflag )
write_vfr( bitmap_hdr *bitmap, resetVFR, char *VFRbuffername )
show_bitmap_x11( char *name, bitmap_hdr *bitmap )
```

Utilities:

```
adjust_bitmap( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap,
               int newx, int newy, int noblur )
blur_bitmap( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap )
byte *blur( byte *buffer, int xsize, int ysize )
bw_to_halftone( int x, int y, int position )
center_image( bitmap_hdr *base, bitmap_hdr *super )
changecolor( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap,
             int cR, int cG, int cB, int nR, int nG, int nB, int distance )
```

```
chroma_bitmaps( bitmap_hdr *basebitmap, bitmap_hdr *superbitmap,
                bitmap_hdr *outbitmap )
chroma( bitmap_hdr *base, bitmap_hdr *super )
chroma_mask( bitmap_hdr *base, bitmap_hdr *mask,
             bitmap_hdr *super, bitmap_hdr *outbitmap )
chroma_shadow_mask( bitmap_hdr *base, bitmap_hdr *mask,
                    bitmap_hdr *super, bitmap_hdr *outbitmap )
byte *cmap_to_bw( bitmap_hdr *image )
byte *cmap_to_inverse( bitmap_hdr *image )
convolve_bitmap( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap,
                 double matrix[9] )
byte *convolve( byte *buffer, int xsize, int ysize, double matrix[9] )
byte *create_bw_pallete()
byte *create_bw_pallete()
byte *cmap_to_bw( bitmap_hdr *image )
create_solid_image( bitmap_hdr *image, int xsize, int ysize,
                    int r, int g, int b )
cut_bitmap( bitmap_hdr *outbitmap, bitmap_hdr *inbitmap,
            int xini, int yini, int xsize, int ysize )
dither_image( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap,
              int levels, int gamma )
fade_mask( bitmap_hdr *base, bitmap_hdr *super,
           bitmap_hdr *mask, bitmap_hdr *out )
flip_image( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap )
byte *flip( byte *buffer, int xsize, int ysize )
gamma_correction( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap,
                  double gamma )
histogram_equalization( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap )
HSL_to_RGB( double h, double s, double l, byte *r, byte *g, byte *b )
hsl_to_rgb_buffer( double *h, double *s, double *l,
                   byte *r, byte *g, byte *b, int size )
mask_change_color( bitmap_hdr *base, bitmap_hdr *mask,
                   bitmap_hdr *outbitmap, newr, newg, newb )
mask_change_to_bw( bitmap_hdr *base, bitmap_hdr *mask,
                   bitmap_hdr *outbitmap )
mask_darken_color( bitmap_hdr *base, bitmap_hdr *mask,
                   bitmap_hdr *outbitmap )
medianfilter( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap )
mirror_image( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap )
byte *mirror( byte *buffer, int xsize, int ysize )
paste( bitmap_hdr *base, bitmap_hdr *super, int xpos, int ypos )
quantize( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap, int no_colors )
byte *rotate90( byte *buffer, int xsize, int ysize )
rotate_bitmap( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap,
               double degrees )
rotate90_image( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap )
byte *rgbtobw( byte *r, byte *g, byte *b, byte *position, int size )
rgb_to_hsl_buffer( byte *r, byte *g, byte *b,
                   double *h, double *s, double *l, size )
RGB_to_HSL( byte r, byte g, byte b, double *h, double *s, double *l )
sharpen_bitmap( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap )
to24( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap )
tobw( bitmap_hdr *inbitmap, bitmap_hdr *outbitmap )
```

Appendix B

or, how to add new subroutines?

The rule is... NO RULES!!! :-)

Everything will be welcome: new formats, more functions for digital treatment, correction and improving of the existing, comments,...

I think that it's only neccesary to keep the philosophy and the way the routines are used. This will help the users and will avoid the *hard* use of the manuals.

For example, suppose we want to create an ew interface with the Khoros VIFF format. We should build the following routines:

```
read_viff_file( char *name, bitmap_hdr *inbitmap )
read_viff( FILE *stream, bitmap_hdr *inbitmap )
write_viff_file( char *name, bitmap_hdr *outbitmap )
write_viff( FILE *stream, bitmap_hdr *outbitmap )
```

The two first ones would fill the *inbitmap* structure with the correspondent information (including the magic field equal to **LUGUSED**) and the two last would create the proper files.

In the viewers, it is recommended to use the name ***show_bitmap*** for the main function (to make possible its co-exist with the X11 subroutine) and the same parameters as the others, even if they are not used.

At last, the only recommendation I can think about for the new digital traetment functions is to keep as the first parameter the incoming image and the output image as the second. The remainder ... well, you are free!.

It may not be neccesary to remind it, but the name of the functions should recall us its purpose. If we are going to implement a function to create an emboss effect, we should call it ***emboss_bitmap***, simply ***emboss*** or ****emboss**** but not something like ***my_new_function***. You get it?

Appendix C

Introduction

In this appendix we intend to give a few more LUG examples.

It will be based in utilities created in the LabCAD. So, the input /output will be made, by default, in ALIAS format.

Meltings

The problem is that we need of a tool to genereate a secuence of frames to fade from one frame to other -something very usual in transitions, beetween sequences, in movies and tv-.

The algorithm is very simple; we know the number of frames to be generated and two images, the first and the last of the sequence. So the process will be a linear interpolation of this two images.

The code would be something like:

```
#include <lug.h>
#include <lugfnts.h>

extern char *MY_NAME;
extern int LUGverbose;

void fade();

main(argc, argv)
int argc;
char **argv;
{
  int i;
```

```
bitmap_hdr source, target;

MY_NAME = argv[0];

/*
 * Get options ( some day I'll build a procedure ).
 */
if ( argc > 1 ) {           /* else core on SGI */
  while ( argv[1][0] == '-' ) {
    for ( i = 1; argv[1][i]; i++ ) {
      switch ( argv[1][i] ) {
            case 'v':
                  LUGverbose++;
                  break;
            case '!':
                  print_copyright();
                  break;
            default :
                  usage();
                  break;
      }
    }
    argv++;
    argc—;
  }
}

/*
 * We need two imagenes, one name and a number.
 */
if ( argc < 5 ) {
  usage();
}

/*
 * A real number ?.
 */
if ( !isnumber(argv[4]) )
  usage();

/*
 * Read the source and the target image.
 */
read_alias_file( argv[1], &source );
read_alias_file( argv[2], &target );

/*
 * Ok. Lets go!.
 */
fade( &source, &target, argv[3], atoi(argv[4]) );

return 0;
}

void
fade( source, target, out_base_name, no_frames )
bitmap_hdr *source, *target;
char *out_base_name;
int no_frames;
{
 int totalsize = source->xsize * source->ysize;
 float *rinc, *rptr;
 float *ginc, *gptr;
```

```
float *binc, *bptr;
byte *srptr, *sgptr, *sbptr;
byte *trptr, *tgptr, *tbptr;
bitmap_hdr image;
char out_name[132];
int i, j;

/*
 * We need image with the same sizes...
 */
if ( source->xsize != target->xsize || source->ysize != target->ysize )
  error( 12 );

/*
 * ... and true color.
 */
if ( source->depth != 24 || target->depth != 24 )
  error( 7 );

/*
 * Get memory.
 */
rptr = rinc = (float *) Malloc( totalsize * sizeof(float) );
gptr = ginc = (float *) Malloc( totalsize * sizeof(float) );
bptr = binc = (float *) Malloc( totalsize * sizeof(float) );

/*
 * Set the pointers.
 */
srptr = source->r, sgptr = source->g, sbptr = source->b;
trptr = target->r, tgptr = target->g, tbptr = target->b;

/*
 * Calculate the diffs. buffers.
 */
fprintf( stderr, "Precalculating differences"), fflush( stderr );
j = no_frames - 2;
for ( i = 0; i < totalsize; i++ ) {
  *rptr++ = ((float)(*trptr++ - *srptr++)) / (float)j;
  *gptr++ = ((float)(*tgptr++ - *sgptr++)) / (float)j;
  *bptr++ = ((float)(*tbptr++ - *sbptr++)) / (float)j;
}

/*
 * Write the new first image.
 */
sprintf( out_name, "%s.%d", out_base_name, 1 );
fprintf( stderr, "\nWritting: %s", out_name ), fflush( stderr );
write_alias_file( out_name, source );

/*
 * The new images will be modifications from the original. So
 * we need it.
 */
copy_bitmap( source, &image );

/*
 * The algorithm: we know the differences per frame ( r/g/binc ),
 *                so each new frame is calculated using the source
 *                image and this buffer.
 * You get it ?.
 */
for ( i = 1; i < no_frames-1 ; i++ ) {
```

```
    fprintf( stderr, "\nCalculating" ), fflush( stderr );
    /* reset the pointers */
    rptr = rinc, gptr = ginc, bptr = binc;
    srptr = source->r, sgptr = source->g, sbptr = source->b;
    trptr = image.r, tgptr = image.g, tbptr = image.b;
    /* calculate the NEW image */
    for ( j = 0; j < totalsize; j++ ) {
      *trptr++ = (int) ( *rptr++ * (float)i + (float)*srptr++);
      *tgptr++ = (int) ( *gptr++ * (float)i + (float)*sgptr++);
      *tbptr++ = (int) ( *bptr++ * (float)i + (float)*sbptr++);
    }
    /* get the file name and write the image */
    sprintf( out_name, "%s.%d", out_base_name, i+1 );
    fprintf( stderr, ". Writting: %s", out_name ), fflush( stderr );
    write_alias_file( out_name, &image );
  }

  /*
   * Write the last image.
   */
  sprintf( out_name, "%s.%d", out_base_name, no_frames );
  fprintf( stderr, "\nWritting: %s", out_name ), fflush( stderr );
  write_alias_file( out_name, target );

  /*
   * Free all buffers and images.
   */
  Free( rinc ); Free( ginc ); Free( binc );
  freebitmap( &image );

  fprintf( stderr, "\nDone\n");
}

usage()
{
  char *msg = "\n\
%s: Usage: %s [-v!] <source_file> <target_file> <out_base_name>
<no_frames>\n\n\
Flags:\n\
\t-v: verbose\n\
\t-!: hey!, what about this program ?!\n\n\
The Alias 'pix' format is the format used.\n";

  fprintf( stderr, msg, MY_NAME, MY_NAME );
  exit( 1 );
}

print_copyright()
{
  char *msg = "\
fade ( v.1.0 ) - creates several images fading from the source to the target\n\n\
This program - (c) 1992, Raul Rivero\n\
LUG  library - (c) 1992, Raul Rivero && Math Dept. ( U. of Oviedo )\n\n\
This software is free and you can get a full copy of original LUG library\n\
via E-mail to rivero@pinon.ccu.uniovi.es or via anonymous ftp to \n\
telva.ccu.uniovi.es ( 156.35.31.31, /uniovi/mathdept/src ).\n\n\
The LUG library includes support for several file formats, viewers on\n\
different architectures and digital image processing.\n\n\
Supported input formats:\n\n\
\t* Pix ( Alias )  *** default ***\n\
\t* TIFF ( needs Sam Leffler's TIFF library )\n\
\t* RLE ( needs Utah Raster Toolkit library )\n\
\t* RLA ( Wavefront )\n\
```

```
\t* SGI ( internal Silicon Graphics file format )\n\
\t* Targa ( Truevision )\n\
\t* GIF ( Compuserve )\n\
\t* PCX ( ZSoft )\n\
\t* PBM/PGM/PPM\n\
\t* Postscript\n\
\t* JPEG ( needs Thomas G. Lane's JPEG library )\n";

  fputs( msg, stderr );
  exit( 1 );
}
```

The precalculations of the needed increment in each step and for any pixel (buffers *rinc*, *ginc* and *binc*), which leaves us with a minimum central loop where we only add the increment to the beggining image.

There is a problem with the use of sums to obtain the increment of each step, and it is the rounding made over the values, so we will use multiplications (for example: if the increment is 1.5 and we add it two times as an integer we'd obtain 2 instead of three, and this is what we obtain with the use of multiplications).

There is left as an exercise the program to fade series of images, this is, given 25 frames from an A sequence and 25 from a B one, we want another 25 sequence to pass from A to B -in three words: a moving fade-

Heightfield to GIF

We already have seen that the only avaliable conection with the **Height Field** format, from *Rayshade*, is for output. But, what happens if we need it in the other way?.

Well, we have to implement that.

```
/*
 * hf2gif.c
 *
 * (c) 1993, Raul Rivero
 *
 */

#include <lug.h>
#include <lugfnts.h>

main( argc, argv )
int argc;
char **argv;
{
 bitmap_hdr bitmap;
 FILE *handle;
 float *input_buffer;
```

```
byte *ptr;
int file_size;

if ( argc < 3 ) {
 fprintf( stderr, "Usage: %s <float_file> <out_file_format>\n");
  exit( 1 );
}

/*
 * Open, read and close the input file.
 */
handle = Fopen( argv[1], "rb" );
input_buffer = (float *) read_file( handle, &file_size );
Fclose( handle );

/*
 * It's a hf file so we supose a [0..1] scale,
 * skip the first four bytes ( the dimension ) and
 * transalete the rest.
 */
bcopy( input_buffer, &(bitmap.xsize), sizeof(int) );
bitmap.ysize = bitmap.xsize;
fprintf( stderr,"converting a %dx%d pixels heightfield\n",
       bitmap.xsize, bitmap.ysize );
ptr = (byte *) input_buffer;
bitmap.r = floattobyte( (float*)(ptr+sizeof(int)), NULL,
                  bitmap.xsize*bitmap.ysize );

/*
 * Fill the other bitmap's fields.
 */
bitmap.depth = 8;
bitmap.colors = ( 1 << bitmap.depth );
bitmap.cmap = create_bw_pallete();
bitmap.magic = LUGUSED;

/*
 * Ok, write it!.
 */
write_gif_file( argv[2], &bitmap );

/*
 * Bye!.
 */
exit( 0 );
}
```

I think it is so easy that it won't need any comment.

Another raw format

Let's suppose a computer vision signature, where with the purpose of freeing the students from the read/write tasks, allowing them to apply their afforts in the digital treatment task, we define the following format:

bytes	offset	description
2	0	xsize
2	2	ysize
xsize*ysize	4	raster data

where the raster information is an intensity description.

```
/*
 * tovision.c
 *
 * I don't wanna write the header so only ...
 *
 * (c) 1992, Raul Rivero
 */

#include <lug.h>
#include <functions.h>

main(argc, argv)
int argc;
char **argv;
{
      bitmap_hdr in, out;
      FILE *handle;

      /* Read the Alias file */
      read_lug_file( argv[1], &in );

      /* Convert it to b&w */
      to_raw_bw( &in, &out );

       /*
        * Ok, now write it ( to file name if given else
         * to stdout.
         */
       if ( argc > 2 )
            handle = Fopen( argv[2], "w" );
      else handle = stdout;
      /* Write the sizes ( MSByte first ) ... */
      putc( MSB(out.xsize), handle );
      putc( LSB(out.xsize), handle );
      putc( MSB(out.ysize), handle );
      putc( LSB(out.ysize), handle );
      /* ... and the image */
      write8bitmap( handle, &out );
       Fclose( handle );

      /* Bye!!! */
       exit( 0 );
}
```

Reescaling

To finish, let's suppose we want to create a program to reescale any kind of image:

```
/*
 * zoomlug.c - perform a zoom ( using the better algorithm ).
 *
 * Author: Raul Rivero
 *         Mathematics Dept.
 *         University of Oviedo
 * Date:   Mon 5 Jul 1993
 * Copyright (c) 1993, Raul Rivero
 *
 */

#include <lug.h>
#include <lugfnts.h>

extern int LUGverbose;
extern char *MY_NAME;

main( argc, argv )
int argc;
char **argv;
{
  int i;
  bitmap_hdr in, out;
  int xsize, ysize;
  char *inputfile, *outputfile;

  MY_NAME = argv[0];

   /*
   * Get options ( some day I'll build a procedure ).
   */
  if ( argc > 1 ) {          /* else core on SGI */
    while ( argv[1][0] == '-' ) {
      for ( i = 1; argv[1][i]; i++ ) {
        switch ( argv[1][i] ) {
              case 'v':
                    LUGverbose++;
                    break;
              case '!':
                    print_copyright();
                    break;
              default :
                    usage();
                    break;
        }
      }
     argv++;
     argc—;
    }
  }

  /*
   * Two arguments ( new_xsize and new_ysize ) ?.
   */
  if ( argc < 2 )
    usage();
```

```
  /*
   * Yeah, almost two arguments, but ... are numbers ?.
   */
  if ( !isnumber(argv[1]) || !isnumber(argv[2]) )
    usage();

  /*
   * Hmmm!, fine.
   */
  xsize = atoi( argv[1] );
  ysize = atoi( argv[2] );

  /*
   * The input/output file.
   */
  inputfile = argv[3];
  outputfile = argv[4];

  read_lug_file( inputfile, &in );
  slow_adjust_bitmap( &in, &out, xsize, ysize );
  write_lug_file( outputfile, &out );

  return 0;
}
```

Conclusion

It's possible that some of the given examples could be too simple or obvious, but it wasn't my intention to give a complete LUG based program, as tovcr, which is a 7000+ lines of code program, but some suggestions and a few more examples.

All these programs, and some more, can be found into the Examples/ directory.

Appendix D

LUG is distributed under the **GNU LIBRARY GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION.**

These are:

- 0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called «this License»). Each licensee is addressed as «you».

  A «library» means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

  The «Library», below, refers to any such software library or work which has been distributed under these terms. A «work based on the Library» means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term «modification».)

  «Source code» for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

  Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a

work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

- 1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

  You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

- 2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

  a) The modified work must itself be a software library.

  b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

  c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

  d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

  (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

  These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate

works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

- 3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

  Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

  This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

- 4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

  If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

- 5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a «work that uses the

Library». Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a «work that uses the Library» with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a «work that uses the library». The executable is therefore covered by this License.Section 6 states terms for distribution of such executables.

When a «work that uses the Library» uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked ithout the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

- 6. As an exception to the Sections above, you may also compile or link a «work that uses the Library» with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

  You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

  a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable «work that uses the Library», as object code and/or source code, so that the user can modify the

Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the «work that uses theLibrary» must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

- 7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

  a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

  b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

- 8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

- 9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

- 10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

- 11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

  If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

  It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software

distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

- 12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

- 13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

  Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and «any later version», you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

- 14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

- 15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY «AS IS» WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS

FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

- 16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Acknowledgements

Well, I express what I fell better in spanish than english. So, sorry, but I'll maint this section in spanish...

El primer agradecimiento debe ser para mi director de proyecto, Adolfo -Adolfillo- Otero. Gracias a él fue mi primer contacto con toda esa comunidad de locos computerizados que hay por todo el mundo (léase INTERNET) y que fue mi fuente de información principal. Además, creo que es fundamental tener libertad a la hora de encarar un proyecto y yo la tuve. Gracias.

¿Y del LabCAD?, ... ¿qué NO puedo decir?. Ya expliqué como nació LUG y su extrecha -extrechísima- relación con el LabCAD. Sin el potencial de hardware allí existente, LUG no hubiese tenido sentido. Pero, por supuesto, tanto o más importante es el potencial humano y el ambiente de trabajo, con nombres propios: Antonio Lana, por permitirme dedicar parte de mi trabajo allí al desarrollo de LUG; al Dr. Mihai -Mija- P. Datcu, mi salvación en algunos temas; y a Quique, el primer usuario de la librería y que al grito de cualquier obscenidad echó partes del código abajo. Gracias, thanks, gracies.

Por supesto, también a toda esa gente que no tiene ningún complejo moral para hacer públicos los fuentes de su trabajo. Existen programas buenísimos que no tienen competidor comercial (hmmm... ¿porqué me estaré ahora acordando del Khoros?) y que son totalmente libres. De muchos de ellos tuve que ir sacando información que aquí, en España, no está disponible (uno de mis mayores problemas fue la imposibilidad de acceder a muchas, muchísimas, de las referencias a libros y *proceedings* que se hacen). En concreto a los autores de las librerías externas que usa LUG: Sam Leffler (tiff), Thomas G. Lane (jpeg/jfif) y Spencer W. Thomas (urt); y a los de algún procedimiento interno de LUG: Chano (de la etsi4g, que implementó la escritura de PostScript); Xiaolin Wu (cuantificación); Spencer W. Thomas, Jim McKie, Steve Davies, Ken Turkowski, James A. Woods, Joe Orost (codificación LZW) y Alan W. Paeth (calculo del punto medio), entre otros. Thanks guys!.

Y, por ultimo, a Jorge. You could read this manual in english cos he translated it from the spanish version. His english is better than mine and although he wasn't able to bear clear -he repeated-, he did a good work. Thanks.

A todos ellos, muchas, muchas gracias :-).

Raúl Rivero