picFX

Maxime Gamboni

COLLABORATORS			
	TITLE : picFX		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Maxime Gamboni	August 24, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

picFX

1.1 picFX documentation

Welcome to the picFX documentation !

picFX version 0.84 alpha

Latest modification: 22nd October 1998

1.2 some boring stuff

DISCLAIMER

I do not take any responsibility if picFX gives you wrong results, if you lose all your money, if your amiga explodes or anything else.

(Don't blame me if you lose your friends because they saw their heads "transformed" by my program :-)

My program is likely to contain some unknown bugs. Please contact me if you found any.

Note: PicFX is still in alpha state. This means that there are known bugs and missing features. Trying to use a missing feature will either do nothing or crash. So it is better to read the Alpha list :-)

Please read about the source-code about its state, if you feel like using it.

STATUS

Yes, picFX is postcardware :-), I would be happy to recieve a postcard of the place you live in if you use my program.

(There are no limitation if you do not register.)

MUI

This application uses

MUI - MagicUserInterface

(c) Copyright 1993-96 by Stefan Stuntz

MUI is a system to generate and maintain graphical user interfaces. With

the aid of a preferences program, the user of an application has the

ability to customize the outfit according to his personal taste.

MUI is distributed as shareware. To obtain a complete package containing

lots of examples and more information about registration please look for

a file called "muiXXusr.lha" (XX means the latest version number) on

your local bulletin boards or on public domain disks.

If you want to register directly, feel free to send

DM 30.- or US$ 20.-

to

Stefan Stuntz

Eduard-Spranger-Straße 7

80935 München

GERMANY

Support and online registration is available at

http://www.sasg.com/

1.3 What you need to discover the beauty of picture manipulation

If you want to use picFX, you will need:

An Amiga computer, it would probably not work if ran with a mac or on windoze.

A graphic card with either CyberGraphics or Picasso96 is required.

Mail me if you would like to have AGA support [1 request for now, Aug98]. ECS is of course out of question.

Version 3.0 of AmigaOS, for the datatypes (a newer is of course welcome!) [It may work under 2.1, but you would be stuck to ilbm pics]

Magic User Interface version 3.6 or better.

The faster your processor is the happier picFX will be :-)

I personnaly use a 040 @25 Mhz (About same speed than 030 @50 MHz) I think that using picFX on slower processors would be awfully slooow.

I have got some benchmarks on a 060 @50MHz. Wow, quite faster there! :-)

Mui Custom Classes used:

- Lamp.mcc by Maik Schreiber

- SettingsWindow.mcc by Ingo Weinhold

Get these classes from Aminet:Dev/mui/

Some mathematical skills would be nice, if you only use the formulas I gave you as example, using picFX would make you lose your time copying huge formulas instead of clicking on a simple button in a "standard" image maniuplator program.

However, if you manage formula creation well enough, picFX will allow you to get really nice effects and get beyond most image manipulation programs.

1.4 How to make the thing work

INTRODUCTION

picFX is a program that allows you to plot a two dimentional graph, with each pixel having the colour defined by your formulas.

You give three formulas - red green blue - depending of the x and y coordinates, and the program will calulate them for each pixels.

X is the horizonal distance to the left border, in pixels, and Y is from the top.

The interesting thing is that you can read pixels from other projects with the r() g() and b() commands, the syntax is:

r(<project number>,<xcoor to calc>,<ycoor to calc>). For example, r(2,37,44) will return the red component of the pixel at coordinates (37,44) of project number 2.

You can then e.g. mix pictures, like this:

r(2,x,y)= (r(0,x,y)+r(1,x,y))/2

g(2,x,y)= (g(0,x,y)+g(1,x,y))/2

b(2,x,y)= (b(0,x,y)+b(1,x,y))/2

The second and third arguments can be function of x and y, only the first must be a constant number:

r(0,y/2,x/2) will make the picture twice bigger and reversed.

An interesting feature of picFX is that it is multithreaded, each time you start a calculation a new task is created and you can work on other projects without waiting.

[Note: The tutorial has now been moved to a separate archive, not yet available. Ask me for the current, incomplete one, if you want]

1.5 User interface

The main window, entitled 'Settings window' is where you will open, manipulate and save your pictures.

It is divided in two parts: The left part is the list of opened projects and the right part is where you edit the current project.

Each line of the opened projects list is formatted like this:

ID: name

When the project is active (either rendering or loading a picture) the ID is followed by an exclamation mark (!).

The ID is a number that allows you to renference this project (e.g. r(3,x,y) references project that has ID #3). Each ID is unique to one project. This ID is not saved.

The name is up to you. The program does absolutely not take care of this name. You are allowed to give several projects the same name but it is better to avoid it so that you can e.g. clearly see in the project list which ID correspond to which project.

Use the 'NEW' button to create a new project. This will open a project window.

The 'Prefs' button opens the Preference Window. See at the bottom of this page.

The right part

The topmost gadget contains the name of the project. This name is put in the title of the project's window and in the project list.

Under this gadget is the actual definition of the object.

You can choose between image and function types. Each one has its own settings.

IMAGE

Path: this is the path of the picture. PicFX uses datatypes to load the pictures, so you can load anyfile, provided you have the datatype.

Note that to load the picture, you have to render the project.

FUNCTION

This projecttype does the following:

For each pixel of the project, it gets the X and Y coordinates.

These coordinates are the number of pixels from left (x) or top (y).

So in the topleft corner, x=0 and y=0

for the topright corner, x=width and y=0 (so if width is the default one, x=255)

Having these coordinates, it evaluates your r, g and b function.

With these red, green and blue, it creates a colour and draws a pixel of this colour in the window.

Currently available functions are:

a+b outputs the sum of a and b

a-b outputs the difference between a and b

a*b outputs the product of a and b

a/b outputs a divided by b. You are allowed to perform a/0, it will just give some undefinded result.

Division may give some very inaccurate results in integer, in some cases.

a^b outputs a raised to the power of b. When integer, this does just a*a*a*a..... So avoid having b too big when working in integer mode!

With float values, you are allowed to put non-integer y. a^0.5 is the square root of b.

sin(a) (Float only) outputs the sine of a. A is a radian value. You can use the "pi" constant to do a conversion if necessary.

cos(a) (Float only) (see sin)

tan(a) (Float only) (see sin). Evaluating tan(pi) only returns an undefined value.

asin(z), acos(z), atan(z) (Float only) These are the inverse functions of sin,cos,tan. The output is a radian value.

r(c,x',y') outputs the red component of the pixel at coordinates (x';y') of project n°c.

Note well that the pixel is directely read and NOT calculated.

c must be a constant. Things like r(x/20,x^2/500,y^2/500) are absolutely forbidden

Look in the project list to get the number of a project.

The project n°c will be frozen during calculation. This means that any attempt to change its bitmap will fail.

However, several projects may reference one single project at the same time without problem.

(See Multi-threading for more info about such topics)

When a coordinate is out of the range, its modulo is calculated.

When a coordinate is non-integer, it is rounded (Currently :-). Maybe in a future version interpolation will be used.

g(c,x',y') outputs the green component of the pixel at coordinates (x';y') of project n°c.

b(c,x',y') outputs the blue component of the pixel at coordinates (x';y') of project n°c.

mod(a,b) outputs a modulo b.

min(a,b) returns the smallest value between a and b. You have to give exactely two arguments.

max(a,b) returns the biggest value between a and b.

You can of course access the x and y variable where you want.

Also available are the e (2.71828...) and pi (3.14159...) constants.

They are quite senseless in integer mode, where e=pi=3 :-)

Write the function that will output the red component of pixel (x;y) at the right of "r(#,x,y)=" (# being replaced by the current project number)

"g(#,x,y)=" is the green component and "b(#,x,y)=" is the blue component.

The lights at the left of the function fields get red (or any other user-defined colour, look the mui preferences program) when the function parser routine discovered a mistake. Do not rely on it, it is still very experimental.

OUTPUT:

32Bits Integer:

This is the fastest and less accurate. It only handles integer numbers, at each step of the calculation. So for example if you have x/5*5 with x=7, it will do the following:

x=7

7/5=2

2*5= 10

However if you write your expression in a smart way, you can often use Integer.

Several functions, like the trigonometric ones, require to have at least 32Bits floating point.

32Bits Float:

This is about half slower than the former but has the advantage to be a lot more accurate.

It saves nine significant digits and features an exponent.

So for instance, 1024 gets $1.024*10^3$, 0.034567 is $3.4567*10^{-2}$

This mode is the most used; all functions work in this mode.

64Bits Float:

This is quite seldom used. It is the same than 32Bits float, but with twice more data and therefore awfully slow.

If 32Bits float gives inaccurate results then switch to this.

The problem with this is that 64Bits require working with two 32Bits words, so it is about twice slower than 32Bits float.

Also note that the modulo (mod()) function is not yet available in 64bits precision.

As a general rule, use the lower precision that give correct results; try to change your functions so that they require less precision.

Following settings Won't calculate the given function, but just take something already calculated

Old Red:

This copies the red component that the pixel had before starting calculation.

Old Green:

This copies the previous green component.

Old Blue:

This copies the previous blue component.

Note that you may mix this components (e.g. set green component to read the previous red), but take care when you're re-rendering the project, you may have to adjust these choices then..

Also note that reading the previous values of the current value may be slower than recalculating them, for very simple functions, like constants (e.g. 255) or single variables (e.g. x).

Copy Red:

Only available for green and blue, this copies the newly calculated red component. It is about like g(0,x,y)=r(0,x,y) (or r...), but doesn't recalculate the function.

(note that auto-referencing is not allowed in any other way. You can't access a pixel of the current project in that project with the inter-referencing functions r(),g() and b())

Copy Green:

Only available for blue, this copies the newly calculated green component.

To work on black 'n' white projects, use the red component and set green and blue to "copy red" and "copy red" (or blue).

SIZE:

The two string gadgets "Width" and "Height" allow you to set the dimension of the current project, in pixels.

ACTION:

Render: switches to 'pause' when clicked on.

This begins the calculation of the current project (or loads a picture, depending on the type of the current project).

Pause:

This pauses the calculation. The project will still be locked and used projects will remain frozen.

Click on it again to resume calculation (Restart).

Save Func:

This saves the current project. It will save all informations, including precision, size and so on.

The resulting file is an ascii one. You can easily modify a function file.

The format of each line is:

"variable=value"

Load Func:

This loads a function. It will overwrite the current settings.

Save Pic:

This saves the current picture to disk. [Sorry, not yet implemented, you may use a screen or window grabber to do this job]

Shown/Hidden:

This switch lets you either hide or show the window containing the picture of the project. Functions like "Save Func" are still working when the window is hidden.

Close:

This closes the project, unless it has been frozen. Check before removing a project that you really want to do it (Do you want to save it before?).

Abort:

If the project is either loading, rendering or paused, this aborts the current rendering.

It does nothing otherwise.

Clear:

This clears the current bitmap with black.

STATUS AREA:

This is divided in two parts, status field and progress gauge.

The status field can contain the following texts:

Idle: The project does not do anything and is not being referenced.

Drawing: The project is either rendering a picture or calculating a graph.

The gauge shows at which state it is.

Paused: The project was active and had been paused. Click on 'Restart' to resume the work.

Loading: The project is loading an image (Either loading from the disk or remapping the picture)

This can't be aborted.

Frozen: The project is not active but another one is reading the bitmap. You are allowed to change the attributes but not render anything.

MESSAGES WINDOW

This window contains the fifteen latest messages (The window title shows how many messages has been sent, but you can only read the last fifteen ones).

Everytime a project outputs a message, it is written there.

Between brackets is the ID of the project that sent the message. (-1 is the main program)

Below the list is a field, which contains "c(p,x,y)=[r,g,b]". Click anywhere in a project window and the coordinates you clicked on will replace x and y; the rgb colour definition will be written at the right of it.

Note well that it does display the value that it was when you clicked on it. So if you draw the project again, or even removed it, these value will remain unchanged.

PREFERENCES WINDOW

This window allows you to customize a bit the bahaviour of PicFX.

Main task priority:

Set here the priority of the main task. Usually better to keep it to zero. If you change it, avoid moving it out of [-2,+2]

Current task priority:

Set here the priority of the render task owned by the current project. Default to -1. Moving it to the same value than the main task risks slowing down the latter.

Render task priorities:

Set here the default priorities of the render tasks. Default is -2. If the current project is working, its task will have the priority set above.

(This means that the selected project will be a bit favorised by Exec, unless you set current=render)

Update Delay:

Set here the number of lines that will be drawn without refreshing.

If you set it to e.g. 5, the display in the project window (and the gauge) will be updated each five lines.

If you set it to one, the display will be updated as often as possible.

If you set it to 'None', the display won't be updated at all, only when finished.

Default is ten. The smaller it is, the slower the rendering will be, although the difference is quite small.

Close:

Set here how PicFX must behave when you close a project window.

You can choose between close, which will remove the project (and loose the picture!)

and iconify, which will simply iconify the window, without even stopping the rendering, if any. Open the window again setting the 'Hidden/Shown' switch.

MUI prefs:

This opens the standard MUI preferences window.

The three buttons below are the standard ones provided by SettingsWindow (see Thanks). They will unfortunately remain in English, regardless of the current language..

Save:

This saves the settings to ENV: and ENVARC:

Use:

Like save, but the settings wil be lost at next boot.

Cancel:

This closes the preferences window without saving.

1.6 Multi-threading

Like said in the introduction, PicFX is multi-threaded. This implies several things:

When you click on the render button, a new task is created. The project is said to be locked.

If this project uses another project (inter-referencing, with r(), g(), b()), the latter is said to be frozen.

To "unlock" a project, simply click on "Abort"; to "unfreeze" a project you have to abort the projects that use it. There is currently no way to directely know what project is using a given project.

When a project is locked, it can't be frozen.

When a project is frozen, it can't be locked, but it can be frozen again. (I.e. several project can reference one single one).

When a project is frozen, each attempt to modify the image (E.g. clearing the bitmap) fails.

You can close locked projects but not frozen projects.

Priorities of the tasks can be set in the preferences program.

1.7 Some interesting formulas...

Colour effects

De/Increase contrast: (Floating point could help :-)

r(1,x,y):=(r(0,x,y)-o)*k+o

o is the origin.

So when the contrast is put very low, all colours will be near o.

When k is bigger than one, contrast is raised (Take care not to get over 255 or below 0!)

When k is between zero and one, there is fewer contrast.

Negative output:

r(1,x,y)=255-r(0,x,y)

Black 'n' White:

r(1,x,y)=(r(0,x,y)+g(0,x,y)+b(0,x,y))/3

Hue extraction:

Divide the colour (each component) by its intensity. Then multiply by a constant, e.g 128.

Maybe better to first create the black'n'white version and then do the division, like this:

Project zero is the picture.

r(1,x,y)=(r(0,x,y)+g(0,x,y)+b(0,x,y))/3

r(2,x,y)=r(0,x,y)/r(1,x,y)*128

g(2,x,y)=g(0,x,y)/r(1,x,y)*128

b(2,x,y)=b(0,x,y)/r(1,x,y)*128

The result is that each pixel has the same hue (i.e. colour) than at the beginning, but the intensity is constant.

Pythagore:

to get the distance from point at coordinates (a;b), do the following:

((x-a)^2+(y-a)^2)^0.5

Remove the square root if having the square of the distance is not a problem.

If you let the (..)^0.5, Floating point must be used.

Sine waves

If you want a colour component to be a sine, do the following right after the sine (or cosine):

sin(...)*127+128

This will output a value within 0 and 255.

Implicit graphes

To "lighten" the places where f(x',y') is near zero:

(h*k)/(f(x',y')^2+k)

The bigger k is, the wider the line will be.

the output will be between 0 and h. (So h is usually 255, to use the full range)

Colour extraction

To extract the pixels of project zero having value (r;g;b):

r(1,x,y)=255/((r(0,x,y)-r)^2+1)

g(1,x,y)=255/((g(0,x,y)-g)^2+1)

b(1,x,y)=255/((b(0,x,y)-b)^2+1)

Relief

k, l and m are usually one, but raise them if the relief can't be well seen. (Contrast)

To take only horizontal relief into account:

r(1,x,y)=128+(r(0,x,y)-r(0,x+1,y))*k

g(1,x,y)=128+(g(0,x,y)-g(0,x+1,y))*l

b(1,x,y)=128+(b(0,x,y)-b(0,x+1,y))*m

To take only vertical relief into account:

r(1,x,y)=128+(r(0,x,y)-r(0,x,y+1))*k

g(1,x,y)=128+(g(0,x,y)-g(0,x,y+1))*l

b(1,x,y)=128+(b(0,x,y)-b(0,x,y+1))*m

To use both dimensions:

r(1,x,y)=128+(2*r(0,x,y)-r(0,x+1,y)-r(0,x,y+1))*k

g(1,x,y)=128+(2*g(0,x,y)-g(0,x+1,y)-g(0,x,y+1))*l

b(1,x,y)=128+(2*b(0,x,y)-b(0,x+1,y)-b(0,x,y+1))*m

Blurring

r(1,x,y)=(r(0,x,y)+r(0,x+1,y)+r(0,x,y+1)+r(0,x+1,y+1))/4

g(1,x,y)=(g(0,x,y)+g(0,x+1,y)+g(0,x,y+1)+g(0,x+1,y+1))/4

b(1,x,y)=(b(0,x,y)+b(0,x+1,y)+b(0,x,y+1)+b(0,x+1,y+1))/4

You can add/remove elements, to de/increase blurring effect.

Also multiply some elements by a constant to give them more importance.

Mask

r(2,x,y)=r(1,x,y)*r(0,x,y)/255

g(2,x,y)=g(1,x,y)*r(0,x,y)/255

b(2,x,y)=b(1,x,y)*r(0,x,y)/255

This will use red component of project zero as a "mask" through which project one will go.

You may have got this mask using colour extraction, see above.

Alpha

If you want to copy a project over another one using something like an alpha channel, try this:

Project zero contains the image to be copied

Project one contains the destination image

Project two contains the mask

Project three is where the result will be rendered

r(3,x,y)=(r(0,x,y)*r(2,x,y)+r(1,x,y)*(255-r(2,x,y)))/255

g(3,x,y)=(g(0,x,y)*r(2,x,y)+g(1,x,y)*(255-r(2,x,y)))/255

b(3,x,y)=(b(0,x,y)*r(2,x,y)+b(1,x,y)*(255-r(2,x,y)))/255

Here only the red component of project two is used. Put g and b if you want to use all colours.

1.8 History of PicFX

I have made a generic operation, that uses inter-referencing, all precision steps and some operators. The time required to do this operation is shown at each version, so you can see the optimisation :-)

Started developement at the beginning of Mai '98

Version 0.78 Alpha: Still private version (08/07/98)

* With optimisation, PicFX became about 150% faster than before

* Time:35.61

Version 0.79 Alpha: Almost released (10/07/98)

* So much bugs removed...

* Wrote documentation (that I am probably the only one to read ;-)

Version 0.80 Alpha: First Public Version (12/07/98)

* Enforcer hits removed; features added; other bugs removed ...

* A little optimisation

* Time:31.92

Version 0.81 Alpha: Private update (02/08/98)

* Added Picasso96 support

* Added mod,min and max functions.

* Functions using commas (r,g,b,mod,min,max) can now be imbricated.

* A little better error-checking.

* Coordinates+colour component display added in messages-window.

* Increased from ten to fifteen messages the size of the messages-window

* Time:30.46

Version 0.82 Alpha: Second Public release (17/08/98)

* I really can't find out what makes my preferences system crash. I tried to reactivate it, and hasn't crashed anymore since then?!?

* Some bugs removed.

Version 0.83 Alpha: Little bug-fix (22/08/98)

* Bug-fix: Setting-refreshing could be disabled [i.e. selecting new project didn't make the settings updated]. This happened when trying to remove a frozen project.

* Some enforcer hits have been removed

* Loading too small pictures with "change size" mode could crash (the window was not yet resized when rendering started).

* Some mistakes in this guide/catalog have been corrected.

* Removed the tutorial out of this guide

Version 0.84 Alpha: A few improvements (22/10/98)

* Lots of output possibilities available for each function (r,g,b)

1.9 PicFX alpha version

Notes:

a. There is one feature yet to be implemented. I will then switch to Beta state.

b. If picFX crashed your system / behaved strangely, please tell me about it, unless it is a known bug from the list below.

c. Please note well that there may be (=> there are) unknown bugs, that may (=> will ;-) crash your system/hard disk. As I already said it, use it at your own risk and do not try to experiment "extreme" situations, I can do it myself :-).

UNIMPLEMENTED FEATURES

1) You can't save pictures. Use a screen grabber (I use Cybergrab, works quite good :-).

KNOWN BUGS

2) Loading jpeg pictures on a picasso96 system crashes.

3) Entering an illegal function may crash the program or produce unpredictable results (see n°5).

I have begun to write some error checks. So sometimes the lamp gets red when a mistake was found. It is getting now quite efficient, but you can still do subtle mistakes that won't be seen.

4) When output to the screen is not possible (e.g. the right mouse button is pressed or window is being opened), some parts of the currently rendering projects will be missing. You have to redraw the window to see them (e.g. resize the window).

5) Writing negative values in a function makes PicFX crash !! (The unary operator -x is not known).

Some examples of expressions that make PicFX crash:

-(x^2+y^2) !!!

r(2,-x,y) !!!

-10*(x^2-y^2) !!!

6) This vertical scroller in the right border of each project window is simply ugly.

Quite serious refreshing problems appear if I try to put the right scroller like the bottom one, in the border.

7) If the main window is (re-)opened [iconification/preferences change] while a project is working, picfx will probably crash, because my program may close and re-open the graphic library used, and the projects that render use this library for each pixel..

8) Preferences may cause enforcer hits/crash.

SERIOUS RISK

9) Avoid trying to close a project (e.g. exit the application) if it is loading a picture, because it may try to send a message to the application, which does not exist anymore...

1.10 I will make picFX approach the perfection as much as possible

Here is what how I intend to improve PicFX, most important at beginning and least important at the end:

1) First: Beta version and then bug-free version :-)

Usage of datatypes.library to do loading and saving of pictures.

2) Possibility to select a rectangular part of a project where rendering will be done (instead of the whole project)

3) As much optimisation as possible (Assembly?)

4) Compilation of functions: you write a function, click a "compile" button, and it will be about twice or thrice faster then :-) Also, automatic compilation possibility.

5) Macros, if a same expression is evaluated lots of times in one project, a macro could be created for it, to gain some rendering time.

7) Maybe some user definable-toolbars that contain function definition, to quickly create an often used function.

8) (Quite unlikely to become reality), an AGA version, using patterened display.

Any suggestion is welcome.

I have thought of a sound editor that would work like this (handle a sound like a single-var function; that software would also support inter-referencing). Tell me what you think about it.

1.11 Catalog and source code

LOCALE

In directory "Translation" are the catalog description and translations.

If you want to translate picFX in your language, do the following:

Let’s say you want to translate into martian.

Choose either an existing language (#?.ct) or the description (picFX.cd)

and copy it to martian.ct

Now edit martian.ct in your favorite ascii editor.

The first lines (preceeding the first descriptions) must be like this:

##version $VER: picFX.catalog 0.0 (10/07/98)

##language martian

##codeset 0

(You may put the correct date in the version information)

If you used the catalog description (.cd), remove each "(//)" at the end of the names.

The rest of the file is structured like this:

variable1

contents1

variable2

contents2

...

Each variable starts with "msg"

When you translate, do not modify the variable names but only the contents !!!

Lines beginning with ";" will be ignored. You can either let the original comments, remove them, or add yours. Please only put comments at the beginning of a line.

"\n" means new-line.

You can split a content over several lines putting a "\" at the end of it:

Hello \

World.

Is the same than

Hello World.

And

Hello\nWorld.

is the same than

Hello\n\

World.

IMPORTANT:

If you translated a .ct file, send it to me and I will send you the catalog as soon as possible (usually the next day or two days after)

If you translated my program using a catalog editor (e.g. CatEdit) then please generate a .ct file for it before sending it to me.

I have seen catalog editors that generate variable names themselves. I do not want anything like this, because like I said above variable names are the identificators of the strings!

SOURCE CODE

Included in "E-source" directory is the full source code of PicFX.

Here are some explanations:

parser.m is a module that converts string functions in a function structure and later evaluates it.

when giving -1 (or TRUE) to the x variable with evaluate, it does not really calculate the function but pre-calculate it for sub-expressions that are constant when only x change.

Please do not use this module in your programs, I have designed it especially for my program. It would probably result in memory and speed waste to do it.

(Ask me if you want to have a general purpose function evaluator, but there are already some, like MathTerm.)

picFXlocale.m is the catalog manager, generated by FlexCat. Look for that program on Aminet is you feel like recompiling my program.

IMPORTANT:

I allow you to pump some code out of my program but please mention my name in the documentation of the program you wrote then and if possible tell me about it.

Of course I do not take any responsibility if you copied a part of my code that was completely buggy. Copy the code at your own risk ;-)

Please do not recompile the code and spread it out saying it is your work, as a [free/share]ware or commercial!

If you find a bug or optimisation possibility, I will be glad to know about it.

1.12 Who wrote this nice piece of software?

My Addresses:

E-Mail: maxime@intelcom.ch

S-Mail:

Maxime Gamboni

Chemin de Rouvenne 8

1800 Vevey, Switzerland

Send all your flames, suggestions, thanks, bug-reports, questions (and postcards :-) to this address. I am always happy to recieve some feed-back from the users of my programs.

I already wrote three other programs, all an Aminet:

1) Convertor (misc/math/convertor.lha)

This is a (giftware) unit converter, very powerful, fast, complete (and so on :-)

2) Birdie Prefs (util/misc/bprefs.lha)

If you are a user of Birdie, rush on this archive, easy-to-use preferences program for that wonderful patch.

3) Circuit (misc/sci/circuit.lha)

Logic circuit simulator, put NAND, OR, XOR and many other doors together. Macro creation feature.

1.13 Acknowledgements

Here are peoples I'd like to thank (No particular order):

* All members of the AmigaE/MUI mailing lists, who answered quickly all questions I posted.

* Fabio Rotondo (I use his Amiga Foundation Classes for picture handling)

* Wouter van Oortmerssen, author of the AmigaE language

* Stefan Stuntz, author of Magic User Interface. Several parts of my program are directely copied from his example programs :-)

* Maik Schreiber, for lamp.mcc

* Ingo Weinhold, for SettingsWindow.mcc

* Nick Clover and Kresimir Farkas, who tested my program while I was writting picasso96 compatibility.

* Some other peoples I forgot. (Even if most are members of the mailing lists cited above :-)