RainbowSystem

Andrea Latina

COLLABORATORS			
	TITLE : RainbowSystem		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Andrea Latina	January 2, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

RainbowSystem

1.1 RainbowSystem.guide

```
                                    ..::
        RainbowSystem
         ::..



            =-=--=-=--=-=--=-=--=-=--=-=--=-
            ENHANCED GRAPHIC SYSTEM FOR AMIGA OS
            =-=--=-=--=-=--=-=--=-=--=-=--=-
```

Introduction...

     What is RainbowSystem

     System Requirements

     About the Author

     How to Install

     Why Register?
The Rainbow programs...

     The Rainbow Manager

     The video Drivers
Developers only...

     The Autodocs

```
                Legal Policies
                Other topics...



                Greetings!
```

1.2 about

Andrea Latina, the author of RainbowSystem, can be contacted at the
following address:

```
     Piazza Peyron 7
        10143 Torino
                ITALIA
```

or, at the following e-mail addresses:

```
     FIDONET:      2:334/21.14
     AMIGANET:   39:101/402.14
     INTERNET:   alatina@hotmail.com      (<<< my new address!!)
```

1.3 introduction

```
                "RainbowSystem" is a 24  bits  (16  million  of  colours)   ←
                        powerful  graphic
functions library which adapts the video output to the graphic hardware  where
it runs.
```

For example, an application which uses it, will be able to visualize its
graphics directly in 16 million colours on an Amiga with a graphic card
installed, or in 256 colours on an AGA Amiga, or in a gray scale on an ECS
Amiga, without any modify by the programmer or by the user.

So, using 'RainbowSystem' is useful for the users and for the programmers: the
ones can use applications which better work on their hardware (remember that
'RainbowSystem' doesn't need a graphic card, but it uses it if present), the
others can (finally!) forget shared pens, colormap, palette, public screens,
etc. etc. and can create very powerful applications, running on every public
screen directly in 16 million colours without any trouble, having in service a
very simple and powerful instructions set.

Technically, 'RainbowSystem' is made of three cooperating programs:

1)

```
            RainbowManager
            : the heart of  the  project;  runs  in  background  and  has
                essentially two charges:
```

    a) It lets the user select which driver join to a public screen.

    b) It automatically assign the appropriate video driver to every

application which uses the "rainbow.library".

2) "rainbow.library": the shared library that the applications must use.

3) the
            Drivers
             video: actually 6, they hold the code segments strictly tied
                    to the hardware:

    a) amy_grey.driver   output in gray scale

    b) amy_color.driver   colour output , it adapts to the number of
                              available pens

    c) amy_color_256.driver   optimized driver for 256 colours screens

    d) cgfx_15_bit.driver   output in 32768 colors

    e) cgfx_16_bit.driver   output in 65536 colors

    f) cgfx_24_bit.driver   output in 16 million colors

All the drivers use the system graphic library, but the d,e,f ones use also
the 'cybergraphics.library', to keep compatibility with the most diffused
graphic cards.

1.4 hardware

Really RainbowSystem needs only few things:

 – The Operating System 3.0 or above

 – A 68020 (or higher) processor

1.5 install

   There are two ways to install "RainbowSystem": you can click on the
"Install" icon to start the automatic sequence, or you proceed manually
following the next steps:

1) copy the 'rainbow.library' library in your LIBS: drawer.

2) copy the "RainbowManager" icon (placed in the "Installation/WBStart_Icon"
   drawer) in your SYS:WBStartup drawer, then insert in the "Default Tool:"
   field of the copied icon the complete path of the "RainbowManager" program
   (eg: "Work:Utilities/RainbowSystem/RainbowManager")

3) Make sure that you have a "Drivers/" directory in the same drawer where the
   "RainbowManager" program is located.

```
SUGGESTION:    If you want to put RainbowManager icon in  your  WBStartup,  you
==========     can also put the "rainbow.library" library in the same directory
               of RainbowManager, instead of in LIBS:
```

1.6 register

```
                 The demo version of RainbowSystem only has two video drivers:

- amy_grey:            16 gray scale (minimum hardware: OCS)

- amy_color_demo:      up to 27 dithered colors (suggested hardware: AGA)


To have the other previously described
               drivers
               , registering it's enough!


The registration quote changes on which is your needed driver:

  - 15 dollars for the drivers:   amy_color        (from 8 to 256 colors)
                                  amy_color_256   (optimized for 256 colors)

  - 20 dollars for the drivers:   cgfx_15/16/24_bit (for graphic cards)

  - 25 dollars for all the drivers, both "amy" and "cgfx".

  (mailing charges are included)

You can register sending me the  money  with  an  international  Postal  Money
Order,  or  in a closed envelope, in any case specify the address where I will
have to send the floppy disk the drivers and your own personal key.

In Italy, registration quotes are: 20.000, 25.000, 30.000 Lire.
```

1.7 greetings

```
   I grasp the opportunity to thank every people who, directly  or  undirectly,
helped me to keep this project to the end:


...THANKS TO:

- Alain Martini
- Alessandro Zummo
- Efrem Mirolo
- Roberto DeFilippi

   Who have let me test RainbowSystem on their graphics  cards  and  for  the
   various suggestions given to me...
```

– Paolo Serrao

    For translating ALL THIS manual into English!!!

– Andreas R.Kleinert

    For the C sources of a shared library...

– Matthias Meixner

    For his gui-builder "GenGUI", which I have used for the RainbowManager...

– Vision Factory Development

    For their 'cybergraphics.library'...

– Nico Francois & Magnus Holmgren

    For having written the useful 'reqtools.library'...

– Stefan Stuntz

    His docs have inspired me for the legal policies :))

1.8 manager

                The program RainbowManager must be launched before every ←
              program which uses
the RainbowSystem, so a good idea would be to put it in the WBStartup drawer
in your boot disk (See how to
        Install
     to what to do).

After having activated it, RainbowManager will open its preferences window if
you click on its icon. With this you can specify which driver you want to use
on a selected public screen.

Manually adding the name of the screen of which you want to specify the driver
to the public screen list is not necessary, because when an application (which
uses RainbowSystem) will open on a public screen not present in the list, it
will be automatically added and the default driver, amy_grey.driver (grey
scale), will be assigned to it.

To manually add a public screen names, you have to click on the "New" gadget,
then you must insert the screen name, paying attention to the upper and lower
case, because the RainbowManager is case sensitive.

1.9 drivers

```
                Drivers are code segments which access  directly  to  the   ←
                    specific  graphic
functions of the hardware concerning to them: drivers for standard  Amiga  use
graphics.library   (amy_#?)   and   drivers   for   graphic   cards   use
cybergraphics.library (cgfx_#?).

Let's see them in detail:

– amy_grey                  – Default used driver

                            – Visualizes graphic data in  16 gray scale

                            – Uses a dithering algorithm  to  enhance  video
                              efficiency


– amy_color_demo            – Visualizes the output in colour, adapting  the
                              output to the number of pens available on  the
                              used  public  screen  (a  minimum  of  8 and a
                              maximum of 27 are required)

                            – Uses  a  dithering  algorithm  to  enhance
                              chromatic efficiency


– amy_color (*)             – Visualizes the output in colour, adapting  the
                              output  to the number of pens available on the
                              used public screen  (a  minimum  of  8  and  a
                              maximum of 256 are required)

                            – Uses  a  dithering  algorithm  to  enhance
                              chromatic efficiency


 – amy_color_256 (*)        – Optimized Driver for 256 colors screens

                            – It doesn't adapt to the  available  number  of
                              pens

                            – Uses  a  dithering  algorithm  to  enhance
                              chromatic efficiency


– cgfx_15_bit (*)           – Driver for CyberGraphX 15 bit screens
                              (32768 colors)

                            – Uses  a  dithering  algorithm  to  enhance
                              chromatic efficiency


– cgfx_16_bit (*)           – Driver for  CyberGraphX   16  bit  screens
                              (65536 colors)

                            – Uses  a  dithering  algorithm  to  enhance
```

                                chromatic efficiency

  - cgfx_24_bit (*)   - Driver for CyberGraphX 24 bit screens
                                         (16 million of colors)

                                       - It doesn't use a dithering algorithm   to
                                         enhance chromatic efficiency because it is not
                                         necessary :-))

  (*) Only available for
              registered
               users.

1.10 developer

                I.
            Important
               II. First of all, a good
            example
               III. Autodocs:

      a. Locking a public screen:

            ObtainScreen

            ReleaseScreen
                    b. The only function that all developers must use :)

            AboutRainbow
                    c. Initializing the graphics structures:

            BeginDraw

            EndDraw
                    d. Getting some informations about the 'object':

            GetRastPort

            GetScreen
                    e. Drawing primitives:

        1.
            Move_RGB
                        2.
            Draw_RGB
                        3.
            DrawLine_RGB
                        4.
            DrawCircle_RGB

```
                      5.
            DrawEllipse_RGB
                      6.
            DrawPolygon_RGB
                      7.
            FillCircle_RGB
                      8.
            FillEllipse_RGB
                      9.
            FillPolygon_RGB
                     10.
            FillRectangle_RGB
                     11.
            WritePixel_RGB
                     12.
            WritePixelLine_RGB
                     13.
            WritePixelArray_RGB
```

1.11   important

```
            If you want to develop using RainbowSystem,  you  must  send
            me
              an  email  to
receive the include files (specifying what compiler do you use).
```

1.12   obtainscreen

```
                SYNOPSIS

     object=ObtainScreen(name, error_code)
                        a0    a1

     APTR ObtainScreen(STRPTR, LONG *);


   FUNCTION

     Allocates an object and initializes the specified public screen.
     This function invokes the help of
             RainbowManager
              to know what driver
     must be used by the graphics functions in the desired screen.


   INPUTS

     name        = name string for public screen or NULL for default public
                   screen.  The string "Workbench" indicates the Workbench  ←
                       screen
```

```
       error_code = a pointer to a LONG variable which stores the possible error
                    code (see file RainbowSystem.h for details)
```

   RESULT

```
   object = APTR pointer to an 'object' to use with other functions.
```

   NOTES

```
   When you have finished, before closing the "rainbow.library", you must
   release this 'object' (to unlock the screen) using
           ReleaseScreen()
```

1.13 releasescreen

```
                   SYNOPSIS

   ReleaseScreen(object )
                   a0

   void ReleaseScreen(APTR );
```

   FUNCTION

```
   Releases the allocated resources and unlocks the public screen (previously ←↩
       locked
   with
           ObtainScreen()
           )
```

   INPUTS

```
   obj   - pointer to an object returned by
           ObtainScreen()
               NOTES

   Before using this function, remember to call
           EndDraw()
            which follows a
   previous
           BeginDraw()
```

1.14 aboutrainbow

```
                   SYNOPSIS

   AboutRainbow(object )
                   a0
```

```
          void AboutRainbow(APTR );
```

FUNCTION

    Show the "About" of RainbowSystem on the public screen previously
    locked with
            ObtainScreen()
                INPUTS

    obj - pointer to an object returned by
            ObtainScreen()
                NOTES

    You should use this function in all your RainbowSystem-dependent ←
        applications.

1.15 begindraw

                SYNOPSIS

    success=BeginDraw(object, rastport)
                        a0   a1

    BOOL BeginDraw(APTR, struct RastPort *);

FUNCTION

    Initializes some internal variables and instructs the object about
    what RastPort must be used

INPUTS

    object   - pointer to an object returned by
            ObtainScreen()
                    rastport - pointer to a RastPort structure

RESULT

    success = TRUE if successful operation
              FALSE if run out of memory

NOTES

    If you want to change the RastPort, before recall this function,
    you must call
            EndDraw()
                SEE ALSO

```
              EndDraw()
```

1.16 enddraw

```
              SYNOPSIS

    EndDraw(object )
            a0

    void EndDraw(APTR );
```

```
  FUNCTION

    Releases everything that was allocated by
          BeginDraw()
              INPUTS

    object   – pointer to an object returned by
          ObtainScreen()
           and
             initialized by
          BeginDraw()
              SEE ALSO


          BeginDraw()
```

1.17 getrastport

```
              SYNOPSIS

    rastport=GetRastPort(object )
                         a0

    struct RastPort *GetRastPort(APTR );
```

```
  FUNCTION

    Get the pointer to the RastPort structure previously transfered to
          BeginDraw()
              INPUTS

    object –  pointer to an object returned by
          ObtainScreen()
           and
             initialized by
          BeginDraw()
              RESULT

    rastport = pointer to a RastPort structure
```

1.18 getscreen

```
                SYNOPSIS

    screen=GetScreen(object )
                     a0

    struct Screen *GetScreen(APTR );
```

```
  FUNCTION

    Get the pointer to the Screen previously locked using
          ObtainScreen()
                INPUTS

    object -  pointer to an object returned by
          ObtainScreen()
                RESULT

    screen = pointer to a Screen structure
```

1.19 move_rgb

```
                SYNOPSIS

    Move_RGB(obj, x,  y)
             a1   d0  d1

    void Move_RGB(APTR, WORD, WORD );
```

```
  FUNCTION
    Moves graphics pen position to (x,y) relative to upper left (0,0)
    of RastPort. This sets the starting point for subsequent
          Draw_RGB()
           calls.
```

```
  INPUTS

    obj - pointer to an object returned by
          ObtainScreen()
           and
        initialized by
          BeginDraw()
                 x,y - point in the RastPort
```

1.20 draw_rgb

```
          SYNOPSIS

   Draw_RGB(obj, x, y, r, g, b)
          a1    d0 d1

   void Draw_RGB(APTR, WORD, WORD, UBYTE, UBYTE, UBYTE );
```

FUNCTION

  Draws a coloured line from the current pen position to (x,y).

INPUTS

```
   obj - pointer to an object returned by
          ObtainScreen()
           and
         initialized by
          BeginDraw()
                  x,y - coordinates of where to end the line in the RastPort ←
                    .

   r,g,b - the color of the line, with:

                 r = 8-bit red component   (0..255)
                 g = 8-bit green component (0..255)
                 b = 8-bit blue component  (0..255)
```

1.21   drawline_rgb

```
          SYNOPSIS

   DrawLine_RGB(obj, x0, y0, x1, y1, r, g, b )
               A0    D0   D1   D2   D3  D4 D5 D6

   void DrawLine_RGB(APTR, ULONG , ULONG, ULONG, ULONG, UBYTE, UBYTE, UBYTE ) ←
      ;
```

INPUTS

```
   obj   - pointer to an object returned by
          ObtainScreen()
           and
          initialized by
          BeginDraw()
                  x0,y0 - coordinates of the initial point of the line

   x1,y1 - coordinates of the final point of the line

   r,g,b - the color of the line, with:

                 r = 8-bit red component   (0..255)
```

```
                    g = 8-bit green component (0..255)
                    b = 8-bit blue component  (0..255)

              example:    0,  0,  0 for black,
                        255,255,255 for white,
                        255,255,  0 for yellow...
```

1.22 drawcircle_rgb

```
              SYNOPSIS

    DrawCircle_RGB(obj, x, y, radius, r, g, b )

    void DrawCircle_RGB(APTR, WORD, WORD, WORD, LONG, LONG );


FUNCTION

    Creates a circular outline within the rectangular region specified
    by the parameters.


INPUTS

    obj - pointer to an object returned by
           ObtainScreen()
            and
          initialized by
           BeginDraw()
                  x,y - the coordinates of the centerpoint

    radius - the radius of the circle (must be > 0)

    r,g,b - the color of the circle, with:

                  r = 8-bit red component   (0..255)
                  g = 8-bit green component (0..255)
                  b = 8-bit blue component  (0..255)

            example:    0,  0,  0 for black,
                      255,255,255 for white,
                      255,255,  0 for yellow...


NOTES

    This function is a macro which calls
          DrawEllipse_RGB
          (obj,x,y,radius,radius,r,g,b)
```

1.23 drawellipse_rgb

```
            SYNOPSIS

    DrawEllipse_RGB(obj, x, y, rx, ry, r, g, b )
                   a0    d0 d1 d2  d3   d4 d5 d6

    void DrawEllipse_RGB(APTR, WORD, WORD, WORD, WORD, UBYTE, UBYTE, UBYTE );
```

FUNCTION

    Creates an elliptical outline within the rectangular region specified
    by the parameters.

INPUTS

```
    obj - pointer to an object returned by
            ObtainScreen()
             and
          initialized by
            BeginDraw()
                    x,y - the coordinates of the centerpoint

    rx  - the horizontal radius of the ellipse (must be > 0)

    ry  - the vertical radius of the ellipse (must be > 0)

    r,g,b - the color of the line, with:

                  r = 8-bit red component   (0..255)
                  g = 8-bit green component (0..255)
                  b = 8-bit blue component  (0..255)

              example:    0,  0,  0 for black,
                        255,255,255 for white,
                        255,255,  0 for yellow...
```

1.24   drawpolygon_rgb

```
            SYNOPSIS

    DrawPolygon_RGB(obj, count, array, r, g, b )
                   A0    D0     A1    D1 D2 D3

    void DrawPolygon_RGB(APTR, UWORD, WORD *, UBYTE, UBYTE, UBYTE  );
```

FUNCTION

    Starting with the first pair in the array, draw connected lines to
    it and every successive pair.

```
INPUTS

    obj  - pointer to an object returned by
           ObtainScreen()
            and
           initialized by
           BeginDraw()
                    count - number of (x,y) pairs in the array

    array - pointer to first (x,y) pair of an array containing
            the coordinates of the vertex of the polygon

    r,g,b - the color of the polygon, with:

                    r = 8-bit red component   (0..255)
                    g = 8-bit green component (0..255)
                    b = 8-bit blue component  (0..255)

                example:    0,  0,  0 for black,
                          255,255,255 for white,
                          255,255,  0 for yellow...
```

1.25 fillcircle_rgb

```
                SYNOPSIS

    FillCircle_RGB(obj, x, y, radius, rgb0, rgb1);

    void FillCircle_RGB(APTR, WORD, WORD, WORD, LONG, LONG );


INPUTS

    obj - pointer to an object returned by
            ObtainScreen()
             and
          initialized by
            BeginDraw()
                    x,y - the coordinates of the centerpoint

    radius - the radius of the circle (must be > 0)

    rgb0 - the color of the circle (a longword in the format: 0xRRGGBB).

            To calculate this value you can use the macro 'RGB(r,g,b)'
            (defined in 'RainbowSystem.h') where:

                    r = 8-bit red component of the color (0..255)
                    g = 8-bit green component (0..255)
                    b = 8-bit blue component  (0..255)

            or a predefined color (see Colors.h)
```

```
rgb1 - the outline color (in the same format of rgb0), or 'RGB_NONE'
       for no outline.


NOTES

   This function is a macro which calls
           FillEllipse_RGB
            (obj,x,y,radius,radius,rgb0,rgb1)
```

1.26 fillellipse_rgb

```
              SYNOPSIS

   FillEllipse_RGB(obj, x, y, rx, ry, rgb0, rgb1);
                   a0    d0 d1 d2  d3  d4    d5

   void FillEllipse_RGB(APTR, WORD, WORD, WORD, WORD, LONG, LONG );


 INPUTS

   obj - pointer to an object returned by
           ObtainScreen()
            and
         initialized by
           BeginDraw()
                   x,y - the coordinates of the centerpoint

   rx  - the horizontal radius of the ellipse (must be > 0)

   ry  - the vertical radius of the ellipse (must be > 0)

   rgb0 - the color of the ellipse (a longword in the format: 0xRRGGBB).

        To calculate this value you can use the macro 'RGB(r,g,b)'
        (defined in 'RainbowSystem.h') where:

             r = 8-bit red component of the color (0..255)
             g = 8-bit green component (0..255)
             b = 8-bit blue component  (0..255)

        or a predefined color (see Colors.h)

   rgb1 - the outline color (in the same format of rgb0), or 'RGB_NONE'
          for no outline.
```

1.27 fillpolygon_rgb

```
              SYNOPSIS
```

```
FillPolygon_RGB(obj, count, array, rgb0, rgb1 )
                a0    d0      a1      d1,   d2


void FillPolygon_RGB(APTR, UWORD, WORD * , LONG, LONG );
```

INPUTS

```
obj  - pointer to an object returned by
          ObtainScreen()
           and
          initialized by
          BeginDraw()
                  count - number of (x,y) pairs in the array

array - pointer to first (x,y) pair of an array containing
          the coordinates of the vertex of the polygon

rgb0  - the color of the polygon (a longword in the format: 0xRRGGBB).

          To calculate this value you can use the macro 'RGB(r,g,b)'
          (defined in 'RainbowSystem.h') where:

              r = 8-bit red component of the color (0..255)
              g = 8-bit green component (0..255)
              b = 8-bit blue component  (0..255)

          or a predefined color (see Colors.h)

rgb1  - the outline color (in the same format of rgb0), or 'RGB_NONE'
          for no outline.
```

1.28 fillrectangle_rgb

```
              SYNOPSIS

FillRectangle_RGB(obj, x, y, width, height, rgb0, rgb1)
                  a0    d0 d1 d2      d3       d4    d5


void FillRectangle_RGB(APTR, ULONG, ULONG, ULONG, ULONG, LONG , LONG );
```

INPUTS

```
obj - pointer to an object returned by
        ObtainScreen()
         and
       initialized by
        BeginDraw()
                x,y - the coordinates of the upper left corner of the  ←
                    rectangle.

width,height - size of the rectangle

rgb0  - the color of the rectangle (a longword in the format: 0xRRGGBB).
```

```
            To calculate this value you can use the macro 'RGB(r,g,b)'
            (defined in 'RainbowSystem.h') where:

                r = 8-bit red component of the color (0..255)
                g = 8-bit green component (0..255)
                b = 8-bit blue component  (0..255)


            or a predefined color (see Colors.h)

    rgb1  - the outline color (in the same format of rgb0), or 'RGB_NONE'
            for no outline.
```

1.29 writepixel_rgb

```
              SYNOPSIS

    result=WritePixel_RGB(oby, x, y, r, g, b )
                          a0   d0 d1 d2 d3 d4


    LONG WritePixel_RGB(APTR, LONG, LONG, UBYTE, UBYTE, UBYTE );


  INPUTS

    obj   - pointer to an object returned by
            ObtainScreen()
             and
            initialized by
            BeginDraw()
                    x,y   - the coordinates of the pixel

    r,g,b - the color of the pixel, with:

                r = 8-bit red component of the pixel (0..255)
                g = 8-bit green component (0..255)
                b = 8-bit blue component  (0..255)

             example:      0,  0,  0 for black,
                         255,255,255 for white,
                         255,255,  0 for yellow...


  RESULT

    result = 0 if pixel succesfully changed
           = -1 if (x,y) is outside the RastPort
```

1.30 writepixelline_rgb

```
              SYNOPSIS
```

```
        result=WritePixelLine_RGB(obj, xstart, ystart, width, array )
                              a0    d0      d1      d2      a1


        LONG WritePixelLine_RGB(APTR , ULONG , ULONG , ULONG , UBYTE * );
```

    INPUTS

```
        obj  - pointer to an object returned by
                ObtainScreen()
                 and
              initialized by
               BeginDraw()
                        x,y  - the coordinates of a point

        width - count of horizontal pixels to write (must be <= 4096 pixels)

        array - pointer to an array of RRGGBB triplets (3 bytes per pixel):

                  RR, GG, BB,     RR, GG, BB,     RR, GG, BB, ....
                  first pixel,   second pixel,  third pixel,  etc. etc.

               where:

                  RR = 8-bit red component of the pixel (0..255)
                  GG = 8-bit green component (0..255)
                  BB = 8-bit blue component  (0..255)
```

    RESULT

```
        result = the number of pixels plotted
```

    NOTES

```
        'Array' should point to at least width*3 UBYTEs (in any case must be  ←
           greater
        than 16 UBYTEs).

        Just another (little) note: this function destroys the content of 'array'  ←
           :-)
```

1.31 writepixelarray_rgb

```
                SYNOPSIS

        result=WritePixelArray_RGB(obj, xstart, ystart, width, height array )
                              a0    d0      d1      d2     d3     a1


        LONG WritePixelArray_RGB(APTR , ULONG , ULONG , ULONG , ULONG, UBYTE * );
```

    INPUTS

```
obj  - pointer to an object returned by
          ObtainScreen()
           and
        initialized by
         BeginDraw()
                  x,y   - the coordinates of starting point

width,height - size of the rectangle that should be transfered
                  ('width' must be <= 4096 pixels)

array - pointer to an array of RRGGBB triplets (3 bytes per pixel) from
         which to fetch the pixel data. Something like:


            line_0:  RR, GG, BB,     RR, GG, BB,     RR, GG, BB, ....
                       first pixel,   second pixel,  third pixel,  etc. etc.

            line_1:  RR, GG, BB,     RR, GG, BB,     RR, GG, BB, ....
              .        first pixel,   second pixel,  third pixel,  etc. etc.
              .
            etc. etc.

         where:

           line_0 = array
           line_1 = array + width*3
             .
             .
           line_n = array + n*width*3

           ('3' is simply the number of RGB components).

         and:

           RR = 8-bit red component of the pixel (0..255)
           GG = 8-bit green component (0..255)
           BB = 8-bit blue component  (0..255)
```

RESULT

```
result = the number of pixels plotted
```

NOTES

```
'Array' should point to at least width*height*3 UBYTEs (in any case must  ←
    be
greater than 16 UBYTEs).

Just another (little) note: this function destroys the content of 'array'  ←
    :-)
```

1.32 example

```
                    #include <exec/types.h>
#include <exec/memory.h>
#include <intuition/intuition.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <math.h>

#include <proto/dos.h>
#include <proto/exec.h>
#include <proto/graphics.h>
#include <proto/intuition.h>

#include <RainbowSystem.h>

struct RainbowSystemBase *RainbowSystemBase;

int main(void )
{
    APTR obj;
    LONG error_code;

    if (RainbowSystemBase=(struct RainbowSystemBase *)OpenLibrary("rainbow.library ←
        ",1L)) {

        if (obj=
                ObtainScreen
                (NULL,&error_code)) {

            const int width=256,height=256;
            struct Window *wnd;

            if (wnd=OpenWindowTags(NULL,WA_Left,        16,
                                        WA_Top,         16,
                                        WA_Title,       "Demo",
                                        WA_InnerWidth,  width,
                                        WA_InnerHeight, height,
                                        WA_CustomScreen,
                GetScreen
                (obj),
                                        WA_IDCMP,       IDCMP_CLOSEWINDOW,
                                        WA_Flags,       WFLG_CLOSEGADGET| ←
                                            WFLG_DRAGBAR|WFLG_DEPTHGADGET| ←
                                            WFLG_SMART_REFRESH,TAG_DONE)) {

                /*
                ** Communicate the dest RastPort to obj
                */

                if (
                BeginDraw
                (obj,wnd->RPort)) {

                    const int offx=wnd->BorderLeft,
                            offy=wnd->BorderTop;
                    int i;
```

```
    /*
    ** Clear Window
    */


FillRectangle_RGB
(obj,offx,offy,width,height,RGB_BLACK,RGB_NONE);

    /*
    ** Draw 50 random lines
    */

    for (i=0;i<50;i++)
    {
        UWORD x0=offx+rand()%width,     // 0 .. width
              y0=offy+rand()%height,    // 0 .. height
              x1=offx+rand()%width,
              y1=offy+rand()%height;
        UBYTE r=rand()&0xFF,            // 0 .. 255
              g=rand()&0xFF,
              b=rand()&0xFF;


DrawLine_RGB
(obj,x0,y0,x1,y1,r,g,b);
    }

    /*
    ** Wait 3 secs and clear the window
    */

    Delay(150);


FillRectangle_RGB
(obj,offx,offy,width,height,RGB_BLACK,RGB_NONE);

    /*
    **  Draw a red circle
    */


DrawCircle_RGB
(obj,width/2,height/2,width/2,0xFF,0,0);

    Delay(150);


FillRectangle_RGB
(obj,offx,offy,width,height,RGB_BLACK,RGB_NONE);

    /*
    **  Draw a filled blue circle, with a red outline
    */
```

```
                    FillCircle_RGB
                    (obj,width/2,height/2,width/2,RGB_BLUE,RGB_RED);

                        Delay(150);


                    FillRectangle_RGB
                    (obj,offx,offy,width,height,RGB_BLACK,RGB_NONE);

                        /*
                        ** Draw 40 random triangles (without outline)
                        */

                        for (i=0;i<40;i++)
                        {
                            UBYTE r=rand()&0xFF,
                                  g=rand()&0xFF,
                                  b=rand()&0xFF;
                            WORD array[6];

                            array[0]=offx+rand()%width;  array[1]=offy+rand()%height;
                            array[2]=offx+rand()%width;  array[3]=offy+rand()%height;
                            array[4]=offx+rand()%width;  array[5]=offy+rand()%height;

                    FillPolygon_RGB
                    (obj,3,array,RGB(r,g,b),RGB_NONE);
                        }

                        /*
                        ** Stop drawing!
                        */


                    EndDraw
                    (obj);

                        WaitPort(wnd->UserPort);
                    }

                    CloseWindow(wnd);
                }

                    ReleaseScreen
                    (obj);

            } else printf("Error code: %d\n",error_code);

            CloseLibrary((struct Library*)RainbowSystemBase);
        }
    return(0L);
}
```

1.33 policies

```
--------------------------------------------------------------------------------
                      Using RainbowSystem in your own applications
--------------------------------------------------------------------------------
```

The following text describes the rules and caveats if you want to use the
RainbowSystem in one of your applications. Please read the complete document,
following the rules are some paragraphs that try to give reasons why things are
handled this way.

Since the rules are different for freely distributable and commercial
applications, some definitions follow before we get started:

In this document, the term "freely distributable" refers to software which is
either really for free (costs nothing) or which lets the user decide if he wants
to pay. Some restrictions for not paying users (better: enhancements for paying
users) are acceptable, but the software has to work even without paying. Freely
distributable software is one of public domain (not copyrighted), freeware
(copyrighted but for free) or shareware (copyrighted and requesting a rather low
fee).

Every program that doesn't fit into the freely distributable group is considered
commercial. If you are unsure about the type of your application, just ask.

```
--------------------------------------------------------------------------------
                          Freely Distributable Software
--------------------------------------------------------------------------------
```

Freely distributable software may use RainbowSystem for free, no special license
agreements   are   needed.   However,   redistributing parts of RainbowSystem
(libraries, drivers, preferences) together with your application is neither
allowed nor necessary. Users of freely distributable applications are usually
enough experienced to look out for the complete RainbowSystem package
themselves. Not redistributing RainbowSystem helps eliminating network traffic
and keeps down archive size. If you really feel that your application absolutely
needs a RainbowSystem coming with it, just contact me. I am sure we will find a
solution.

The copyright information contained in all programs using RainbowSystem and the
accompanying documentation should state that this program uses RainbowSystem and
that RainbowSystem was written by Andrea Latina.

Freely distributable software should also contain some basic information about
RainbowSystem to help unexperienced users to find it and to make some little
advertisement for my system. You can either directly use the supplied
"RainbowSystem.redme" for this purpose or say something similiar with your own
words. If you really dislike the advertisement, I won't mind if you remove the
registration part from the readme file. But hey... you got this fantastic
RainbowSystem for free so why not help me making some money? :-)

```
--------------------------------------------------------------------------------
                               Commercial Software
--------------------------------------------------------------------------------
```

RainbowSystem within commercial software is not for free. Your company will have

to pay a licensing fee somewhere between US$ 50.- for very small and US$ 500.for very big applications. Usually, the price is calculated by multiplying the suggested retail price of your product with a factor of five, but this is only some kind of very rough example. Rather expensive applications with probably very few customers (e.g. "special purpose" software) will of course get other conditions. Also, if you plan to use RainbowSystem for more applications, multi application licenses are available. Just contact me and ask.

The license agreement will allow you to use the current and all following versions of RainbowSystem with the current and all following versions of your product. You will also get the rights to reproduce and redistribute some of the files from the RainbowSystem distribution, including the Rainbowsystem library, the drivers and the RainbowManager program. Special commercial versions of this preferences program without shareware reminders are available on demand.

The copyright information contained in all programs using RainbowSystem and the accompanying documentation should state that this program uses RainbowSystem and that RainbowSystem is copyrighted by and reproduced under license from Andrea Latina.

                                   Discussion

First of all, these policies are not some kind of quick hack. I considered lots of other possibilities and it took quite a long time for me to decide. Please read the following paragraphs carefully, I hope you will understand my reasons.

RainbowSystem shall be used in all kinds of applications, regardless whether they are distributed as Public Domain, Freeware, Giftware, Shareware, Commercial Ware or whatever else.

First of all, if something wants to become a standard on the Amiga, the public domain and freeware scene is the most important thing to consider. There is a really huge number of programmers that work just for fun, supplying all the little (and sometimes big) tools that make our lifes easier. These people do a really great job and surely will help keeping the Amiga alive for a long long time.

Of course I could have released RainbowSystem as a completely commercial product, sold for a somewhat high price. Some companies might have bought it to create some of their applications, but only very few public domain or shareware programmers would have been willing to pay such a considerable amount of money. And even if some of them would, RainbowSystem would never have the chance to become a real standard. Besides this fact, I don't think that it's a good idea to take money from people who spend their spare time in writing public domain applications. If an application is for free, the use of RainbowSystem has to be free too.

Since charging programmers is not what I wanted to do, the only way for me to get some money out of RainbowSystem is to have the users of applications pay for it. Well, in fact they are the ones who benefit from flexible and configurable programs, charging them seems quite reasonable. Luckily, there are a lot more users than programmers. This results in a very low price which seems to be even more cheap if you consider that a single registration allows configuration of all currently existing and all future RainbowSystem applications.

Furthermore, I do not force people to register. Most other shareware products allow some period of evaluation time after that one either has to register or to delete the program. This is not true for RainbowSystem. Registration is only necessary when some advanced configuration options are wanted.

Distribution policies for commercial applications are kind of different. If I see someone making real money with the aid of my work, it should be easily understandable that I also want to get a little piece of that cake. That's why the use of RainbowSystem is not for free in commercial programs.

My first ideas were to have some kind of percentage fee per sold application but this would become uncontrollable and too complicated to handle quite soon. So I decided to have a fixed license fee which's amount depends on the size of the product. Thus, small and relatively cheap programs with probably not too much financial profit will be able to get a cheap RainbowSystem license whereas big products will have to pay a bit more.

I understand that it's nearly impossible to sell a commercial product together with a RainbowSystem preferences program with some disabled options and shareware reminders. Therefore, commercial licensees may get a special stripped version of this tool which only contains the possible settings of an unregistered RainbowSystem but doesn't contain any reminders or other stuff unsuitable for commercial applications. I am also thinking of a system that allows commercial programs to come with full featured preferences, restricted only to the specific application.

I really hope that these policies will satisfy the requirements of both, freeware authors and commercial companies and of course also of application users. Currently, this seems to work quite well. Anyway, if you have some other ideas or suggestions how things could be handled better, feel free to tell me about them. I am always looking for new ideas. But please keep in mind my main destinations mentioned above since I won't give up any of them.

                              Andrea Latina