AmigaMail

COLLABORATORS

	TITLE : AmigaMail		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 14, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

AmigaMail

1.1   IV-91: Color Wheel and Gradient Slider Boopsi Classes

Color Wheel and Gradient Slider Boopsi Classes

by Mark Ricci, Martin Taillefer, and David Miller

The Color Wheel and Gradient Slider classes are V39 additions to
Boopsi. They are specifically intended for use with the expanded
color capabilities of the Advanced Graphics ArchitectureÔ (AGA). In
order to understand their use and the reasons they were added, a
brief background on color theory follows.

The color model in Amiga systems from V30 (Release 1.0) through V38
(Release 2.1) has been RGB based, that is, color was specified as
some combination of red, green and blue. These are known as the
primary additive colors. If you start at a value of zero for each
primary--the color black--adding varying levels of them enables you
to specify any of the Amiga's colors till you've added them all at
their highest level which is the color white.

The opposite of the primary additive colors are the subtractive
primary colors, cyan, magenta, and yellow, respectively. These
colors are created by completely subtracting one of the primary
colors from white. Cyan is green-blue, magenta is red-blue, and
yellow is red-green. Subtracting all of them brings you back to the
color black.

Another method of specifying colors is in terms of hue, saturation
and brightness, HSB. Hue is the dominant wavelength in the light we
receive; we associate color names with hue. Saturation is a measure
of the purity of the dominant wavelength, that is, the more dominant
the wavelength, the more pure or saturated it is. Brightness is a
measure of the luminance or brilliance of the wavelength.

The RGB method of color specification is used because it corresponds
to the red, green and blue receptor cones in our eyes. Through the
varying intensities of these wavelengths that the receptors receive,
we are able to see colors. It's a natural way for us to represent

colors and till now, was the only OS-supported way on the Amiga.
Some applications did allow users to specify colors in terms of HSV
(V stands for value and corresponds to brightness), but the HSV
calculations had to be done by the application.

Under V39 (Release 3.0), the color model includes both RGB and HSB.
It also expands from a twelve bit color definition--four bits each
for red, green and blue--to a twenty-five bit color definition -
eight bits each for red, green and blue plus one bit for genlocking.
With the expanded color model, programmers and users have finer
control over color selection than ever before.

The new color model is based on a color cylinder where the set of
available hues, saturations and brightnesses is represented as a
cylinder. The cylinder is divided radially into hues, with the
saturation level of a hue increasing from zero at the center of the
cylinder to its maximum at the outer edge of the cylinder. The
altitude of the cylinder represents the brightness of the hue. At
the top of the cylinder, the brightness is maximized; at the bottom,
the brightness is minimized. We see the minimum brightness as black,
regardless of the hue; the maximum brightness of a hue, however, is
not white.

If you refer to color theory literature, you will notice the V39
color model closely resembles the Munsell color model. The
similarity is not by design, but makes for interesting reading
nonetheless.

To implement the new color model, V39 adds two new Boopsi classes,
the Color Wheel and Gradient Slider classes. The classes provide two
gadgets for working with the expanded palettes of AGA machines, the
color wheel and gradient slider gadgets. The color wheel gadget is a
two dimensional representation of the hue and saturation elements of
the cylinder called a color wheel. The gradient slider gadget is
based on the brightness element of the color cylinder.

Visually, the color wheel gadget is a display of all of the available
hues and their saturations depicted as a multicolored wheel with a
selector knob that can be moved anywhere on the wheel. The gradient
slider is typically a vertically oriented slider displaying a smooth
transition of a color from its brightest to its darkest. The
gradient slider knob selects the intensity of the brightness.

This is a huge change in how a user selects colors. The old method
was to display a color in an indicator box with three sliders for the
red, green and blue components of the color and each slider's knob
set at the proper level of the components. The user would then
select the desired color by manipulating the three sliders.

With the color wheel gadget, the user can move the knob to the
approximate color he wishes to use, and then, if sliders are provided
as they are in the V39 Palette Preferences editor, manipulate the
sliders to refine the components of the selected color.

In the same way that the color wheel knob can be moved to the
approximate color, the gradient slider gadget knob can be moved to
the approximate brightness level. The color wheel and gradient

slider gadgets provide a much more intuitive method for selecting
colors.

For developers, the new gadgets do the heavy lifting like all Boopsi
gadgets. As the user moves the mouse across the color wheel, the
gadget will return the RGB or HSB value (depending on which you
request) corresponding to the knob location, and the gradient slider
will return the current position of the knob as a value between the
highest brightness value and the lowest brightness value. In
addition, the color wheel has two functions that convert RGB values
to HSB and vice versa.

The Color Wheel Gadget

The color wheel gadget has two rendering modes of display: monochrome
and color. The rendering mode is dependent on the number of
bitplanes its screen has, and the number of available pens. Four or
less bitplane screens generally result in a monochrome wheel, though
if there are enough pens available, you will get a minimal color
wheel with four bitplanes. Five through eight bitplane screens
result in successively more colors as the bitplanes increase. The
V39 Palette Preferences editor, for example, uses eight bitplanes.

The monochrome rendering mode exists for instances where the color
wheel is being used on an ECS system and there aren't enough pens
available, or when the color wheel screen cannot get four bitplanes.
If you are using an ECS machine, you can be fairly certain of getting
a minimal color wheel if you limit the gradient slider pen array to
four pens.

A monochrome color wheel is broken into six sectors, one each for the
additive and subtractive primaries, The sectors are denoted by the
first letter of its associated primary as shown below.

You can localize the sector markers by using the WHEEL_Abbrv tag.
The default string is "GCBMRY." Make sure your localized tag is in
the proper order.

The color wheel gadget can handle both RGB and HSB 32-bit values.
This enables developers and users to work with it in either mode.
When you create the color wheel, the RGB or HSB values you set
determine the initial position of the knob. These values default if
you do not supply them. The RGB defaults are full red, no green, no
blue; the HSB defaults are hue 0, full saturation, full brightness.
The default knob position is, accordingly, the topmost edge of the
red section of the color wheel.

RGB and HSB values can be passed to the gadget either as separate tag
items, or in an RGB or HSB structure. The tags and the structures
are defined in <gadgets/colorwheel.h>.

        Tag Name   Purpose

        WHEEL_Hue   set the hue component
        WHEEL_Saturation   set the saturation component

```
        WHEEL_Brightness            set the brightness component
        WHEEL_HSB                   set HSB using an HSB structure

        WHEEL_Red                   set the red component
        WHEEL_Green                 set the green component
        WHEEL_Blue                  set the blue component
        WHEEL_RGB                   set RGB using an RGB structure
```

```
 struct ColorWheelHSB
     {
         ULONG cw_Hue;
         ULONG cw_Saturation;
         ULONG cw_Brightness;
     };

 struct ColorWheelRGB
     {
         ULONG cw_Red;
         ULONG cw_Green;
         ULONG cw_Blue;
     };
```

Keep in mind that although you can specify brightness, it will not be
displayed because the colorwheel's two dimensions are hue and
saturation. Instead, the colorwheel gadget can pass this value to
the gradient slider gadget if you set the WHEEL_GradientSlider tag
with the address of the gradient slider. Then, any time the wheel
brightness is modified, the slider knob position will automatically
display the change.

The color wheel will return either RGB or HSB values depending on how
you query it. If you wish to continuously receive mouse position
reports as the user moves the knob, set the GA_FollowMouse tag when
you create it.

The Gradient Slider Gadget

The gradient slider gadget is a non-proportional gadget for the
brightness component of a color. It has a special attribute--it can
render a pen array as a color gradient in its box. If you properly
specify the pen array, the gradient slider will render it as a smooth
transition, most times from brightest to darkest, although you may
choose to go from darkest to brightest. The intent of the gradient
is to allow the user to move the knob to the appropriate brightness
level of the selected color.

The position of the gradient slider knob is set by the value of the
GRAD_CurrVal tag. However, there are two potential pitfalls that you
must take into account when setting this tag. The first is that the
top of the slider is zero and the bottom of the slider is the largest
value in its range. This is the opposite of what you would expect.
The other is that the gradient slider works with 16-bit values, not
32-bit like the color wheel. Make sure you account for these.

If you modify the brightness of the color wheel, the gradient slider
will pick up the change via the link established by the color wheel's
WHEEL_GradientSlider tag. The gradient slider converts the 32-bit
brightness value to a 16-bit value and sets the knob position to the
proper position.

Creating the Pen Array

The pen array you pass to the gradient slider is a set of pen numbers
you obtain using the V39 ObtainPen() function, or from an existing
set of pens like the screen's pens. In most cases, you will use the
ObtainPen() function.

Before you create the pen array, you need to decide where you will
begin. You have two choices, you can use a color from the current
palette or design a color of your own. In the former case, you
access the color registers of the screen's colormap; in the latter
case, you set the RGB or HSB values yourself. In this article, color
0 (color registers 0, 1, and 2) of the screen's palette will be used.

To access a colormap's registers, you call the V39 function
GetRGB32(), passing it the colormap, the starting color register, the
number of registers and an array large enough to hold three times the
number of registers you request. This is because a color register
has a 32-bit value for each of its red, green and blue components.

```
    ULONG colortable[12];  /* space for the first four colors */
    struct Screen *Colorscreen;

    /* Get the first four colors */
    GetRGB32(Colorscreen->ViewPort.ColorMap,0L,4L,colortable);
```

Once you decide which color to use, the RGB values must be converted
to HSB values in order to modify the brightness. You do this with
the color wheel function ConvertRGBToHSB() which converts the RGB
values stored in a ColorWheelRGB structure to HSB values and returns
them in a ColorWheelHSB structure.

```
    struct ColorWheelRGB rgb;
    struct ColorWheelHSB hsb;

    /* set RGB values for color 0 in ColorWheelRGB structure */
    rgb.cw_Red   = colortable[0];
    rgb.cw_Green = colortable[1];
    rgb.cw_Blue  = colortable[2];

    /* now convert the RGB values to HSB */
    ConvertRGBToHSB(&rgb,&hsb);
```

Since the topmost color is going to be the brightest, its brightness
must be set to the maximum value, $FFFFFFFF.

```
    /* max out the brightness component */
    hsb.cw_Brightness = 0xffffffff;
```

The final decision is the number of pens to use to render the

gradient. A good rule of thumb is eight pens as a minimum and
sixteen as a maximum. Eight will give you a nice display and sixteen
will give you a very nice display. If you're wondering, the Palette
Preferences editor uses thirty-two pens.

With the starting color converted to HSB, its brightness maximized
and the number of pens set, the pen array can be built by converting
each new HSB value back to RGB with the ConvertHSBToRGB() function
and calling ObtainPen(). Remember, member 0 of the array will be the
starting color and each successive member will be proportionately
dimmer. To facilitate updating the pen array later on, a structure
for use with the V39 LoadRGB32() function is set up for each pen
number. The code below demonstrates how the array is built.

```
    #define GRADCOLORS   16        /* Set to 4 for ECS to ensure enough color  ↩
        wheel pens */
    struct load32 color_list[GRADCOLORS + 1];
    WORD penns[GRADCOLORS +1];
    WORD numPens;

    numPens = 0;
    while (numPens < GRADCOLORS)
    {
        hsb.cw_Brightness = 0xffffffff - ((0xffffffff / GRADCOLORS) * numPens);
        ConvertHSBToRGB(&hsb,&rgb);

        penns[numPens] = ObtainPen(Myscreen->ViewPort.ColorMap,-1,
                           rgb.cw_Red,rgb.cw_Green,rgb.cw_Blue,PEN_EXCLUSIVE);
        if (penns[numPens] == -1)
            break;

        /* Set up LoadRGB32() structure for this pen */
        color_list[numPens].l32_len = 1;
        color_list[numPens].l32_pen = penns[numPens];
        numPens++;
    }
    penns[numPens] = ~0;
    color_list[numPens].l32_len = 0;
```

The pen array is now complete and ready to be passed to the gradient
slider when you create it.

Updating The Pen Array

In our scheme, we said the color gradient is updated every time the
user moves the mouse to a new position on the color wheel. This is
done by updating the pen array whenever a mousemove message is
received.

The algorithm is to query the color wheel for the current HSB setting
and then loop through all the pens as we did above to gradually
decrease the brightness. There is one difference, though, instead of
obtaining pens, we will change the colors of the pens using either
SetRGB32() or LoadRGB32(), both of which were added for V39.

If you use SetRGB32(), you must call it for each pen you change; if

you use LoadRGB32(), you can pass it a structure containing a
variable number of pens to change in one call. As we said, we will
use LoadRGB32().

```
    case IDCMP_MOUSEMOVE:

        /*
         * Change gradient slider color each time
         * colorwheel knob is moved.  This is one
         * method you can use.
         */

        /* Query the colorwheel */
        GetAttr(WHEEL_HSB,colwheel,(ULONG *)&hsb);

        i = 0;

        while (i < numPens)
        {
            hsb.cw_Brightness = 0xffffffff - ((0xffffffff / numPens) * i);
            ConvertHSBToRGB(&hsb,&rgb);

            color_list[i].l32_red = rgb.cw_Red;
            color_list[i].l32_grn = rgb.cw_Green;
            color_list[i].l32_blu = rgb.cw_Blue;
            i++;
        }
        LoadRGB32(&Myscreen->ViewPort,(ULONG *)color_list);
        break;
```

Using the Gadgets

The color wheel and gradient slider gadgets are found on the system
in the Classes/Gadgets drawer of Workbench as colorwheel.gadget and
gradientslider.gadget instead of being part of Intuition. They must
be opened with OpenLibrary() before they can be used.

```
        struct Library *ColorWheelBase, *GradientSliderBase;

        main()
        {
        ...

        OpenLibrary("Classes/Gadgets/colorwheel.gadget",39L);
        OpenLibrary("Classes/Gadgets/gradientslider.gadget",39L);
```

Once opened, you use them as you would any Boopsi gadgets, that is,
create them using NewObject() and dispose of them using
DisposeObject(). However, you must use CloseLibrary() because you
used OpenLibrary() earlier.

```
        CloseLibrary(ColorWheelBase);
        CloseLibrary(GradientSliderBase);
```

The example program below creates a color wheel and gradient slider
on the deepest screen possible. As you move the color wheel knob

around the wheel, it will update the gradient of the gradient slider.

 WheelGrad.c