

```
//
//  GRGradientFunctions
//
//  By Anders Bertelrud
//  Copyright (c) 1995-1996 Anders Bertelrud
//

#import <math.h>
#import <libc.h>
#import "GRGradientFunctions.h"

//
//  Integer types with well-defined number of bits, unlike "int" or "short". These should be
//  in some architecture-specific file where they are always defined to be the native type
//  that provides the specified number of bits.
//
typedef unsigned char    uint8;
typedef signed long    sint32;

//
//  _GRHSBToRGB (private to this file)
//
static inline void _GRHSBToRGB (float h, float s, float l, float * r, float * g, float * b)
{
    NXColor        color;

    color = NXConvertHSBToColor(h, s, l);
    NXConvertColorToRGB(color, r, g, b);
}
```

```

//
//  GRDrawHSBGradient
//
void GRDrawHSBGradient (NXRect rectangle, float hue, float saturation, float startBrightness,
                        float endBrightness)
{
    uint8 *          pixelData;
    register uint8 * pixelPtr;
    sint32           IR, dLR, rR, dRR;
    sint32           IG, dLG, rG, dRG;
    sint32           IB, dLB, rB, dRB;
    int              y, height, width;

    // Figure out the width and height of the resulting bitmap, and if either dimension is less
    // than or equal to zero, we leave.
    NXIntegralRect(&rectangle);
    width = rectangle.size.width;
    height = rectangle.size.height;
    if (width <= 0 || height <= 0)
        return;

    // Compute the 16.16 fixed-point minimum and maximum pixel values.
    {
        #define      _GRColorFloatToFix(floatval)  ((sint32)floor(floatval*255.0 * 65536.0))
        float        averageBrightness = (startBrightness + endBrightness) / 2.0;
        float        minRed, minGreen, minBlue;
        float        avgRed, avgGreen, avgBlue;
        float        maxRed, maxGreen, maxBlue;

        _GRHSBToRGB(hue, saturation, startBrightness, &minRed, &minGreen, &minBlue);
        _GRHSBToRGB(hue, saturation, averageBrightness, &avgRed, &avgGreen, &avgBlue);
        dLR = (_GRColorFloatToFix(avgRed) - _GRColorFloatToFix(minRed)) / height;
        IR = _GRColorFloatToFix(minRed) + (dLR >> 1);
        dLG = (_GRColorFloatToFix(avgGreen) - _GRColorFloatToFix(minGreen)) / height;
        IG = _GRColorFloatToFix(minGreen) + (dLG >> 1);
    }
}

```

```

dLB = (_GRCOLORFLOATTOFIX(avgBlue) - _GRCOLORFLOATTOFIX(minBlue)) / height;
IB = _GRCOLORFLOATTOFIX(minBlue) + (dLB >> 1);
_GRHSBTORGB(hue, saturation, endBrightness, &maxRed, &maxGreen, &maxBlue);
dRR = (_GRCOLORFLOATTOFIX(maxRed) - _GRCOLORFLOATTOFIX(avgRed)) / height;
rR = _GRCOLORFLOATTOFIX(avgRed) + (dRR >> 1);
dRG = (_GRCOLORFLOATTOFIX(maxGreen) - _GRCOLORFLOATTOFIX(avgGreen)) / height;
rG = _GRCOLORFLOATTOFIX(avgGreen) + (dRG >> 1);
dRB = (_GRCOLORFLOATTOFIX(maxBlue) - _GRCOLORFLOATTOFIX(avgBlue)) / height;
rB = _GRCOLORFLOATTOFIX(avgBlue) + (dRB >> 1);
}

```

// Allocate memory for the pixels.

```

pixelData = malloc(sizeof(uint8) * 3*width*height);

```

// Run the loop, interpolating pixel values.

```

pixelPtr = (uint8 *)pixelData;
for (y = 0; y < height; y++)
{
    register int x;
    sint32          r, g, b, dR, dG, dB;

    dR = (rR - IR) / width;      r = IR + (dR >> 1);
    dG = (rG - IG) / width;      g = IG + (dG >> 1);
    dB = (rB - IB) / width;      b = IB + (dB >> 1);
    for (x = 0; x < width; x++)
    {
        *pixelPtr++ = r >> 16;
        *pixelPtr++ = g >> 16;
        *pixelPtr++ = b >> 16;
        r += dR; g += dG; b += dB;
    }
    IR += dLR; rR += dRR;
    IG += dLG; rG += dRG;
    IB += dLB; rB += dRB;
}

```

```
// Render the bitmap.
NXDrawBitmap(&rectangle, width, height, 8, 3, 24, width*3, NO, NO, NX_RGBColorSpace,
    &pixelData);
```

```
// Deallocate the pixel storage.
free(pixelData);
```

```
}
```

```
//
// GRDrawGrayGradient
//
```

```
void GRDrawGrayGradient (NXRect rectangle, float startBrightness, float endBrightness)
{
```

```
    uint8 *          pixelData;
    register uint8 * pixelPtr;
    sint32           ll, dLI, rl, dRI;
    int              y, height, width;
```

```
// Figure out the width and height of the resulting bitmap, and if either dimension is less
// than or equal to zero, we leave.
NXIntegralRect(&rectangle);
width = rectangle.size.width;
height = rectangle.size.height;
if (width <= 0 || height <= 0)
    return;
```

```
// Compute the 16.16 fixed-point minimum and maximum pixel values.
{
```

```
    #define          _GRColorFloatToFix(floatval)  ((sint32)floor(floatval*255.0 * 65536.0))
    float            averageBrightness = (startBrightness + endBrightness) / 2.0;

    dLI = (_GRColorFloatToFix(averageBrightness) - _GRColorFloatToFix(startBrightness))
        / height;
```

```

    ll = _GRColorFloatToFix(startBrightness) + (dLI >> 1);
    dRI = (_GRColorFloatToFix(endBrightness) - _GRColorFloatToFix(averageBrightness))
        / height;
    rl = _GRColorFloatToFix(averageBrightness) + (dRI >> 1);
}

```

```

// Allocate memory for the pixels.
pixelData = malloc(sizeof(uint8) * width*height);

```

```

// Run the loop, interpolating pixel values.
pixelPtr = (uint8 *)pixelData;
for (y = 0; y < height; y++)
{
    register int    x;
    register sint32 i, dl;

    dl = (rl - ll) / width;          i = ll + (dl >> 1);
    for (x = 0; x < width; x++)
    {
        *pixelPtr++ = i >> 16;
        i += dl;
    }
    ll += dLI;  rl += dRI;
}

```

```

// Render the bitmap.
NXDrawBitmap(&rectangle, width, height, 8, 1, 8, width, NO, NO, NX_OneIsWhiteColorSpace,
    &pixelData);

```

```

// Deallocate the pixel storage.
free(pixelData);

```

```

}

```