```
//
//      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                          lR, dLR, rR, dRR;
        sint32                          lG, dLG, rG, dRG;
        sint32                          lB, 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;
                lR = _GRColorFloatToFix(minRed) + (dLR >> 1);
                dLG = (_GRColorFloatToFix(avgGreen) - _GRColorFloatToFix(minGreen)) / height;
                lG = _GRColorFloatToFix(minGreen) + (dLG >> 1);
```

```
    dLB = (_GRColorFloatToFix(avgBlue) - _GRColorFloatToFix(minBlue)) / height;
    lB = _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 - lR) / width;         r = lR + (dR >> 1);
    dG = (rG - lG) / width;         g = lG + (dG >> 1);
    dB = (rB - lB) / width;         b = lB + (dB >> 1);
    for (x = 0; x < width; x++)
    {
        *pixelPtr++ = r >> 16;
        *pixelPtr++ = g >> 16;
        *pixelPtr++ = b >> 16;
        r += dR; g += dG; b += dB;
    }
    lR += dLR; rR += dRR;
    lG += dLG; rG += dRG;
    lB += 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				lI, dLI, rI, 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;
```

```
        lI = _GRColorFloatToFix(startBrightness) + (dLI >> 1);
        dRI = (_GRColorFloatToFix(endBrightness) - _GRColorFloatToFix(averageBrightness))
           / height;
        rI = _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, dI;

        dI = (rI - lI) / width;     i = lI + (dI >> 1);
        for (x = 0; x < width; x++)
        {
            *pixelPtr++ = i >> 16;
            i += dI;
        }
        lI += dLI;rI += dRI;
    }

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

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