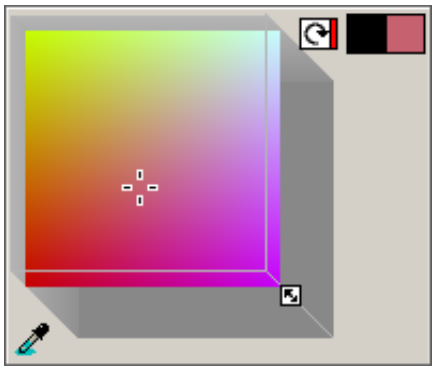
Color Interpolation

Simple Color Interpolation in a Square

Anti-Grain Geometry has classes for linear, Bresenham-like interpolation with subpixel accuracy. They are dda_line_interpolator, dda2_line_interpolator, and line_bresenham_interpolator. And they can be also used to interpolate colors in color selection controls like the one in **Xara X**:



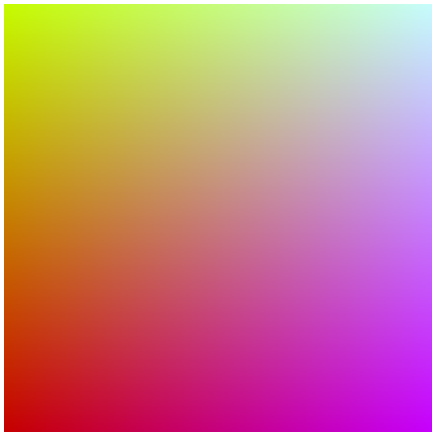
Below is the class that interpolates colors of the rgba8 type, and the function to draw a square with interpolation. It can be easily modified to draw a rectangle, but it's not really necessary to render the color editing control.

```
#include <stdio.h>
#include <string.h>
#include "agg_rendering_buffer.h"
#include "agg_pixfmt_rgb24.h"
#include "agg_renderer_base.h"
#include "agg_dda_line.h"

enum
{
    square_size  = 200
};

// Writing the buffer to a .PPM file, assuming it has
// RGB-structure, one byte per color component
//-----------------------------------------------------
bool write_ppm(const unsigned char* buf,
               unsigned width,
               unsigned height,
               const char* file_name)
{
    FILE* fd = fopen(file_name, "wb");
    if(fd)
```

```
    {
        fprintf(fd, "P6 %d %d 255 ", width, height);
        fwrite(buf, 1, width * height * 3, fd);
        fclose(fd);
        return true;
    }
    return false;
}


namespace agg
{
    class color_interpolator_rgba8
    {
    public:
        color_interpolator_rgba8(agg::rgba8 c1, agg::rgba8 c2, unsigned len) :
            m_r(c1.r, c2.r, len),
            m_g(c1.g, c2.g, len),
            m_b(c1.b, c2.b, len),
            m_a(c1.a, c2.a, len)
        {
        }

        void operator ++ ()
        {
            ++m_r; ++m_g; ++m_b; ++m_a;
        }

        rgba8 color() const
        {
            return rgba8(m_r.y(), m_g.y(), m_b.y(), m_a.y());
        }

    private:
        dda_line_interpolator<16> m_r;
        dda_line_interpolator<16> m_g;
        dda_line_interpolator<16> m_b;
        dda_line_interpolator<16> m_a;
    };



    // Rendering a square with color interpolation between its corners
    // The colors of the corners are ordered CCW started from bottom-left,
    // assuming that the Y axis goes up.
    //----------------------------------------------------------------------
    template<class Renderer>
    void color_square_rgba8(Renderer& r, int x, int y, int size,
                            rgba8 c1, rgba8 c2, rgba8 c3, rgba8 c4)
    {
        int i, j;
        color_interpolator_rgba8 cy1(c1, c4, size);
        color_interpolator_rgba8 cy2(c2, c3, size);
        for(i = 0; i < size; ++i)
        {
            color_interpolator_rgba8 cx(cy1.color(), cy2.color(), size);
            for(j = 0; j < size; ++j)
            {
                r.copy_pixel(x + j, y + i, cx.color());
                ++cx;
            }
```

```
            ++cy1;
            ++cy2;
        }
    }
}


int main()
{
    unsigned char* buffer = new unsigned char[square_size * square_size * 3];

    agg::rendering_buffer rbuf(buffer,
                               square_size,
                               square_size,
                               -square_size * 3); // Flip Y to go up

    agg::pixfmt_rgb24                    pf(rbuf);
    agg::renderer_base<agg::pixfmt_rgb24> rbase(pf);

    agg::color_square_rgba8(rbase, 0, 0, square_size,
                            agg::rgba8(0xc6, 0,    0),     // Bottom-left
                            agg::rgba8(0xc6, 0,    0xff),  // Bottom-right
                            agg::rgba8(0xc6, 0xff, 0xff),  // Top-right
                            agg::rgba8(0xc6, 0xfe, 0));    // Top-left


    write_ppm(buffer, square_size, square_size, "agg_test.ppm");

    delete [] buffer;
    return 0;
}
```

Here is the result:



It's not included into the distribution package because it's rather a specific class. Besides, it depends on the rgba8 color type.

Copyright © 2002-2006 **Maxim Shemanarev**
Web Design and Programming **Maxim Shemanarev**