
Subject: [Proposal] Blur algorithm (fast box blur with gaussian approximation)

Posted by [Oblivion](#) on Fri, 17 Apr 2020 22:51:27 GMT

[View Forum Message](#) <> [Reply to Message](#)

Hello,

Blur seems to be "missing" from the image utility functions.

I've adapted Ivan Kutsir's fast blur implementation to U++.

The only missing thing is alpha channel blurring. I commented out the alpha channel blurring in the below code. (it leaves artifacts at the margins of output image)

```
Image Blur(const Image& img, int radius)
{
    // This code is adapted from Ivan Kutsir's fast blur implementation, published under MIT license.
    // See: http://blog.ivank.net/fastest-gaussian-blur.html

    // An implementation of well known fast box and gaussian blur
    // approximation algorithms by Wojciech Jarosz and Peter Kovesi.
    // See: https://elynxsdk.free.fr/ext-docs/Blur/Fast_box_blur.pdf
    // See: https://www.peterkovesi.com/papers/FastGaussianSmoothing.pdf

    auto GetBoxes = [](int sigma, int n) -> Vector<int>
    {
        double wl = ffloor(sqrt((12 * sqr(sigma) / n) + 1));
        if(fmod(wl, 2) == 0) wl--;
        double wu = wl + 2;
        double m = fround((12 * sqr(sigma) - n * sqr(wl) - 4 * n * wl - 3 * n) / (-4 * wl - 4));
        Vector<int> sizes;
        for (int i = 0; i < n; i++)
            sizes.Add(i < m ? wl : wu);
        return pick(sizes);
    };

    auto ApplyBoxBlur = [] (const Image& src, int r) -> Image
    {
        double avg = 1.0 / (r + r + 1);

        Size sz = src.GetSize();
        ImageBuffer tmp(sz);
        ImageBuffer out(sz);

        Premultiply(tmp);
        Premultiply(out);
```

```

for(int i = 0; i < sz.cy; i++) {
    int ti = 0;
    int li = ti;
    int ri = r;
    const RGBA& fv = src[i][0];
    const RGBA& lv = src[i][sz.cx - 1];
    dword rsum = fv.r * (r + 1);
    dword gsum = fv.g * (r + 1);
    dword bsum = fv.b * (r + 1);
    // dword asum = fv.a * (r + 1);
    for(int j = 0; j < r; j++) {
        const RGBA& p = src[i][j];
        rsum += p.r;
        gsum += p.g;
        bsum += p.b;
        // asum += p.a;
    }
    for(int j = 0; j <= r; j++) {
        const RGBA& p = src[i][ri++];
        RGBA& q = tmp[i][ti++];
        q.r = (rsum += p.r - fv.r) * avg;
        q.g = (gsum += p.g - fv.g) * avg;
        q.b = (bsum += p.b - fv.b) * avg;
        // q.a = (asum += p.a - fv.a) * avg;
    }
    for(int j = r + 1; j < sz.cx - r; j++) {
        const RGBA& p = src[i][ri++];
        const RGBA& q = src[i][li++];
        RGBA& t = tmp[i][ti++];
        t.r = (rsum += p.r - q.r) * avg;
        t.g = (gsum += p.g - q.g) * avg;
        t.b = (bsum += p.b - q.b) * avg;
        // t.a = (asum += p.a - q.a) * avg;
    }
    for(int j = sz.cx - r; j < sz.cx ; j++) {
        const RGBA& p = src[i][li++];
        RGBA& q = tmp[i][ti++];
        q.r = (rsum += lv.r - p.r) * avg;
        q.g = (gsum += lv.g - p.g) * avg;
        q.b = (bsum += lv.b - p.b) * avg;
        // q.a = (bsum += lv.a - p.a) * avg;
    }
}

for(int i = 0; i < sz.cx; i++) {
    int ti = 0;
    int li = ti;
    int ri = r;

```

```

const RGBA& fv = tmp[ti][i];
const RGBA& lv = tmp[sz.cy - 1][i];
dword rsum = fv.r * (r + 1);
dword gsum = fv.g * (r + 1);
dword bsum = fv.b * (r + 1);
// dword asum = fv.a * (r + 1);
for(int j = 0; j < r; j++) {
    const RGBA& p = tmp[j][i];
    rsum += p.r;
    gsum += p.g;
    bsum += p.b;
    // asum += p.a;
}
for(int j = 0; j <= r; j++) {
    const RGBA& p = tmp[ri++][i];
    RGBA& q = out[ti++][i];
    q.r = (rsum += p.r - fv.r) * avg;
    q.g = (gsum += p.g - fv.g) * avg;
    q.b = (bsum += p.b - fv.b) * avg;
    // q.a = (asum += p.a - fv.a) * avg;
}
for(int j = r + 1; j < sz.cy - r; j++) {
    const RGBA& p = tmp[ri++][i];
    const RGBA& q = tmp[li++][i];
    RGBA& t = out[ti++][i];
    t.r = (rsum += p.r - q.r) * avg;
    t.g = (gsum += p.g - q.g) * avg;
    t.b = (bsum += p.b - q.b) * avg;
    // t.a = (asum += p.a - q.a) * avg;
}
for(int j = sz.cy - r; j < sz.cy; j++) {
    const RGBA& p = tmp[li++][i];
    RGBA& q = out[ti++][i];
    q.r = (rsum += lv.r - p.r) * avg;
    q.g = (gsum += lv.g - p.g) * avg;
    q.b = (bsum += lv.b - p.b) * avg;
    // q.a = (asum += lv.a - p.a) * avg;
}
}

out.SetHotSpots(src);
out.SetResolution(src.GetResolution());
return out;
};

if(radius < 1 || IsNull(img))
    return img;

```

```
Vector<int> boxes = GetBoxes(radius, 3);

Image pass1 = ApplyBoxBlur(img, (boxes[0] - 1) / 2);
Image pass2 = ApplyBoxBlur(pass1, (boxes[1] - 1) / 2);
Image output = ApplyBoxBlur(pass2, (boxes[2] - 1) / 2);

return pick(output);
}
```

If you think that it needs more refining, let me know, or feel free to modify it as it suits your needs.

Note: It should be easy to add a MT variant, using CoDo if needed.

Best regards,
Oblivion
