◊ VIII.1

Fast Embossing Effects on Raster Image Data

John Schlag
Computer Graphics Group
Industrial Light and Magic
PO Box 2459
San Rafael, CA 94912

By viewing an image as a 3D surface over a regular grid, one can apply standard 3D shading techniques to it to good effect. This is a useful tool to have in a paint system. The shading can be considerably optimized due to the constrained nature of the surface. This note concentrates on embossing — better known in 3D rendering circles as bump mapping (Blinn 1978) — although once a surface position and normal are available, many of the standard 3D shading techniques can be applied. Input images can be obtained from a variety of sources, including paint work, scanned logos, text (from outline fonts), or even z-buffers.

Consider a bump map image as a 3D surface $p(u, v) = [u\ v\ i(u, v)]$. This interprets black pixels (0) as low, and white (255) as high. With $i_u = \partial i/\partial u$, etc., the surface normal is then

$$N(p) = p_u \times p_v = \begin{vmatrix} 1 & 0 & i_u \\ 0 & 1 & i_v \\ \mathbf{x} & \mathbf{y} & \mathbf{z} \end{vmatrix} = [-i_u\ \ -i_v\ 1]$$

The partial derivatives i_u and i_v at a point p can be estimated from the image by convolving with the following masks:

$$i_u(p) \approx \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad i_v(p) \approx \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge conditions for evaluating these masks are left to the reader. Note that since the masks are 3 by 3 pixels, this will blur the bump map image a bit, lending some antialiasing to sharp edges.

The normal thus obtained must be corrected for the response of the masks. It is also useful to be able to control the "bump height." Both of these aims can be accomplished by scaling $i(u, v)$, or, equivalently, by scaling N_z. The latter is preferable since N_z is a constant. (Note that $[-\alpha i_u\ \ -\alpha i_v\ 1]$ is the same normal as $[-i_u\ \ -i_v\ 1/\alpha]$.) If the bump

IBM ISBN 0-12-336155-9
Macintosh ISBN 0-12-336156-7

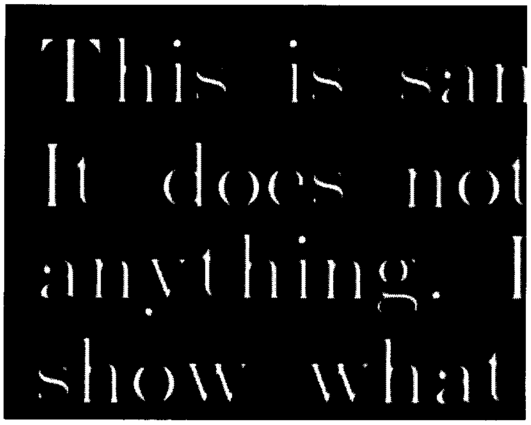

Figure 1. A bump map.

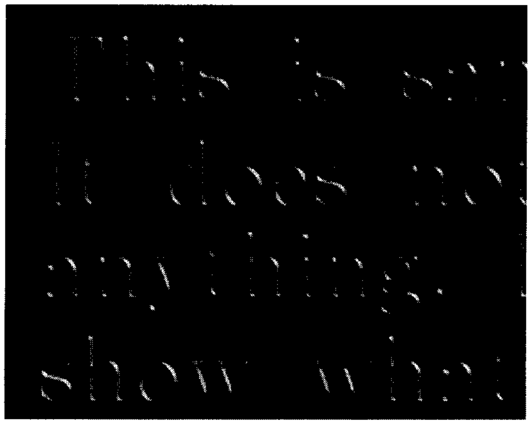

Figure 2. Embossed image lit from above right.

map is obtained from a decal or logo of some sort, as is often the case, the size of the transition region, or "bevel" — and, hence, the apparent depth of the effect — can be modulated by blurring the map image. In this case, a useful control is the edge transition width w that corresponds to a 45° angle for N (i.e., the diameter of the blurring filter). Consider the image function $i(u, v) = au$. The convolution will produce an estimate $i_u = 3(a(u+2)) - 3au = 6a$. The slope is $a = 255/w$, so we set $N_z = 6 * 255/w$.

By restricting ourselves to a single distant light source, we can get cheap but pleasing results from the Lambertian shading formula $N \cdot L/(|N||L|)$. The multiplications necessary to scale the shading result to a pixel value can be avoided by scaling L to length 256. The calculations can be further optimized by noting that often many of the bump map pixels are black. This results in vertical normals, all of which yield the same pixel value. This value can be computed ahead of time and is simply $[0\,0\,1] \cdot L$ or L_z. In these cases, we can skip the calculation of N (especially the normalization). When the length of N must be computed, there are other optimizations that apply (Ritter 1990). Figure 1 shows a bump map and Figure 2 the result of shading.

The shading result can be used directly as a monochrome pixel value, put through a color map to produce a color value, or multiplied by another texture image. When using a texture image, most of the texture pixels get darkened by a factor of L_z. If this is undesirable, shading results larger than L_z can be used to lighten pixels, and shades less than L_z to darken pixels. Color plate VIII.1.1 shows an embossed, textured image.

One interesting psychophysical note on embossing is that people typically perceive the result of this process as embossing if the light is coming from the top of the image, and as engraving if the light is coming from the lower half.

Although not as heavily optimizable as embossing, other effects such as reflection and refraction can be applied by choosing an eyepoint somewhere above the image. The view vector from the eyepoint to each pixel can then be reflected into an environment map, or refracted into a background texture. For highest quality results, however, some technique for both texture and highlight antialising must be applied, as in 3D (Williams 1983, Crow 1984).

◊ Source Code ◊

```
/*
 * Emboss - shade 24-bit pixels using a single distant light source.
 * Normals are obtained by differentiating a monochrome 'bump' image.
 * The unary case ('texture' == NULL) uses the shading result as output.
 * The binary case multiples the optional 'texture' image by the shade.
 * Images are in row major order with interleaved color components (rgbrgb...).
 * E.g., component c of pixel x,y of 'dst' is dst[3*(y*xSize + x) + c].
 */

#include <math.h>
#include <sys/types.h>

void
Emboss(
    double azimuth, double elevation,    /* light source direction */
    u_short width45,                     /* filter width */
    u_char *bump,                        /* monochrome bump image */
    u_char *texture, u_char *dst,        /* texture & output images */
    u_short xSize, u_short ySize         /* image size */
)
{
    long Nx, Ny, Nz, Lx, Ly, Lz, Nz2, NzLz, NdotL;
    register u_char *s1, *s2, *s3, shade, background;
    register u_short x, y;

    #define pixelScale 255.9

    /*
     * compute the light vector from the input parameters.
     * normalize the length to pixelScale for fast shading calculation.
     */
    Lx = cos(azimuth) * cos(elevation) * pixelScale;
    Ly = sin(azimuth) * cos(elevation) * pixelScale;
    Lz = sin(elevation) * pixelScale;

    /*
     * constant z component of image surface normal - this depends on the
     * image slope we wish to associate with an angle of 45 degrees, which
     * depends on the width of the filter used to produce the source image.
     */
    Nz = (6 * 255) / width45;
    Nz2 = Nz * Nz;
    NzLz = Nz * Lz;

    /* optimization for vertical normals: L.[0 0 1] */
    background = Lz;

    /* mung pixels, avoiding edge pixels */
    dst += xSize*3;
    if (texture) texture += xSize*3;
    for (y = 1; y < ySize-1; y++, bump += xSize, dst += 3)
    {
        s1 = bump + 1;
        s2 = s1 + xSize;,
        s3 = s2 + xSize;
        dst += 3;
        if (texture) texture += 3;
        for (x = 1; x < xSize-1; x++, s1++, s2++, s3++)
        {
            /*
             * compute the normal from the bump map. the type of the expression
             * before the cast is compiler dependent. in some cases the sum is
             * unsigned, in others it is signed. ergo, cast to signed.
             */
            Nx = (int)(s1[-1] + s2[-1] + s3[-1] - s1[1] - s2[1] - s3[1]);
            Ny = (int)(s3[-1] + s3[0] + s3[1] - s1[-1] - s1[0] - s1[1]);

            /* shade with distant light source */
            if ( Nx == 0 && Ny == 0 )
                shade = background;
            else if ( (NdotL = Nx*Lx + Ny*Ly + NzLz) < 0 )
                shade = 0;
            else
                shade = NdotL / sqrt(Nx*Nx + Ny*Ny + Nz2);

            /* do something with the shading result */
            if ( texture ) {
                *dst++ = (*texture++ * shade) >> 8;
                *dst++ = (*texture++ * shade) >> 8;
                *dst++ = (*texture++ * shade) >> 8;
            }
            else {
                *dst++ = shade;
                *dst++ = shade;
                *dst++ = shade;
            }
        }
        if (texture) texture += 3;
    }
}
```

◇ Bibliography ◇

(Blinn 1978) James Blinn. Simulation of wrinkled surfaces. *Computer Graphics*, 12(3), 1978.

(Crow 1984) Franklin C. Crow. Summed-area tables for texture mapping. *Computer Graphics*, 18(3):207–212, 1984.

(Ritter 1990) Jack Ritter. A fast approximation to 3D Euclidean distance. In Andrew S. Glassner, editor, *Graphics Gems*, pages 432–433. Academic Press, Boston, 1990.

(Williams 1983) Lance Williams. Pyramidal parametrics. *Computer Graphics*, 17(3), 1983.