# Creating Contour Gradients using 3D Bevels

Paul Asente*
Adobe

Nathan Carr†
Adobe


**Figure 1:** *A contour gradient applied to text, with color contours aligned with the boundaries of the characters and moving linearly inward.*

## Abstract

Contour gradients have color contours that follow the shape of the path being filled. Existing algorithms cannot create them in a resolution- and scale-independent way, causing visible rendering artifacts if enlarged. We describe a new method that approximates them with a set of paths filled by linear gradients. A 3D bevel of the path being filled gives both the shape of these paths and the information needed to compute gradient vectors for the linear gradients. Our representation is efficient, compact, and both resolution and scale independent.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

**Keywords:** vector graphics, contour gradient, bevel

## 1 Introduction and previous work

Color gradients are a common feature in 2D graphic design applications. Basic gradients have color contours that follow simple paths, like straight lines or ellipses. 2D graphics rendering systems like SVG, PDF, and the PostScript® language [W3C 2011; Adobe 2005c; Adobe 1999b] support simple gradients, so authoring programs can represent them in a resolution-independent way. This allows the final renderer to use the device attributes and output size to assign colors in a way that will avoid banding artifacts.

Contour gradients, sometimes called shape gradients, are another kind of gradient. They use the shape of the path being filled, with color contours following inset copies of the path (Figure 1). Because of their complexity, they are not directly supported by any existing low-level rendering systems; instead, applications that support them must represent them using simpler constructs. Fireworks® [Adobe 2012a] creates raster images.

*e-mail:asente@adobe.com
†e-mail:ncarr@adobe.com


CAe 2013, July 19 – 21, 2013, Anaheim, California.


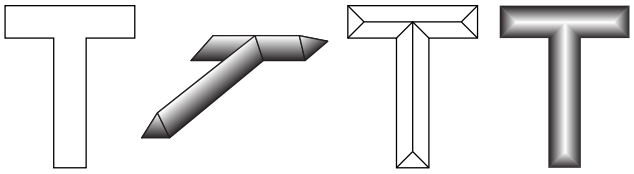

**Figure 2:** *A simple shape, the bevel, the bevel projected into 2D, and a contour gradient derived from the bevel*

FreeHand® [Schulze 2003] and CorelDRAW® [Bouton 2012] create multiple inset copies of the path, filling them with different solid colors.

Neither of these solutions is resolution or scale independent; resolution and contour spacing must be determined by the authoring program when an illustration is saved. If the results are scaled up, pixels or contours can become evident. This can be mitigated to some extent by choosing a high pixel resolution or small contour spacing, but even then there will be limits, and the amount of data required can be quite large.

## 2 Bevel-based contour gradients

Our approach is based upon the observation that one can construct an inset copy of a path by creating a three-dimensional bevel of the path (for example, one based upon a straight skeleton [Aichholzer and Aurenhammer 1996; Eppstein and Erickson 1998]) and intersecting it with a horizontal plane. If each face of the bevel model is colored with a linear gradient so that the gradient contours are horizontal, the result can be projected down to the plane of the path to give a contour gradient (Figure 2).[1] We don't need to actually apply gradients to the bevel faces; instead we can project the faces into 2D and then fill each with a linear gradient.

All common bevel algorithms take polygons as input, and not curves. If the input path contains curved segments, we first approximate them with a series of line segments so that the resulting polyline lies within some small tolerance $t$ of the original path. Figure 3 shows this using a tolerance of 0.05 points. To avoid visible corners if the result is greatly enlarged, we offset the flattened path by $t$, creating a path that completely encloses the original path. We

[1] Please be aware that many of the figures in this paper strongly exhibit the Vasarely illusion [Adelson 2000], in which rays of the center color appear to extend into the corners but do not actually do so.

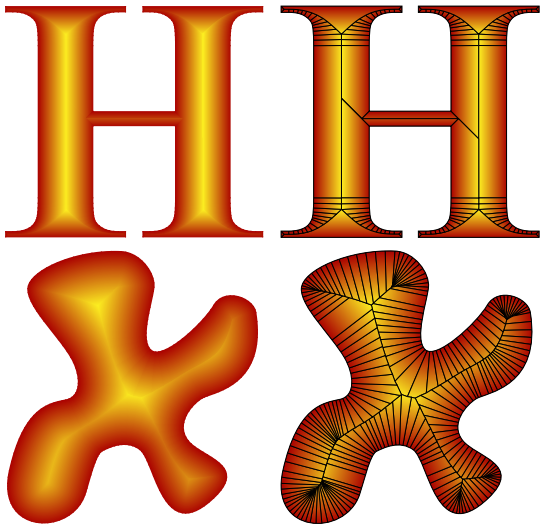

**Figure 3:** *Contour gradients on complex paths, and the linear gradient patches that define them.*

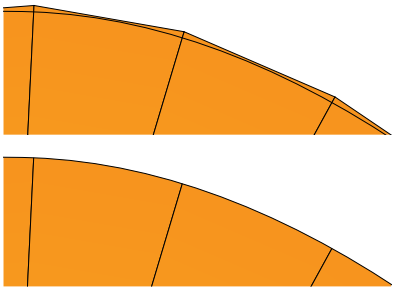

**Figure 4:** *The offset approximated path, clipped by the original path*

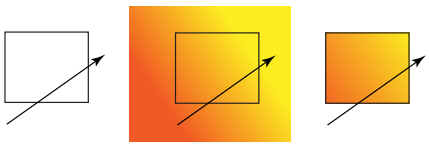

**Figure 5:** *The effect of the gradient vector*

then clip the gradient-filled faces with a copy of the original path to give a smooth curved border (Figure 4).

When a linear gradient is used to fill a path, its appearance is controlled by its *gradient vector*. The start of the gradient vector gets the starting color of the gradient, the end of the gradient vector gets the ending color, and the color contours extend perpendicularly to the gradient vector. The resulting gradient is then clipped by the path being filled (Figure 5).

When filling a projected bevel face with a gradient, we must set the gradient vector so that the color contours of the gradient follow the bevel contours of equal height, and the distance along the gradient vector corresponds to the relative height of each point on the face. Algorithm 1 does this.

To compute $L$, a line on the face perpendicular to the horizontal contour (see Figure 6) we take the normal $N = (N_x, N_y, N_z)$ of the face and construct $H = (-N_y, N_x, 0)$. The vector $D = N \times H$ is perpendicular to both $N$ and $H$. Because $D$ is perpendicular to $N$

**Algorithm 1** Projected Bevel Gradient Fill Algorithm

$P$ is a two-dimensional polygonal path
$B$ is the 3D model created by beveling $P$
$Z_{min}$ is the minimum $z$ coordinate of $B$
$Z_{max}$ is the maximum $z$ coordinate of $B$
$P_{min}$ is the horizontal plane through $Z_{min}$
$P_{max}$ is the horizontal plane through $Z_{max}$
**for** each face $F$ of $B$ **do**
  project $F$ onto $P_{min}$, giving $F_{2d}$
  let $H$ be a horizontal vector ($z = 0$) that lies on $F$
  let $L$ be a line that lies on $F$ and is perpendicular to $H$
  let $S$ be the segment of $L$ between $P_{min}$ and $P_{max}$
  project $S$ onto $P_{min}$, giving $V_{2d}$
  fill $F_{2d}$ with the gradient, using $V_{2d}$ as the gradient vector
**end for**

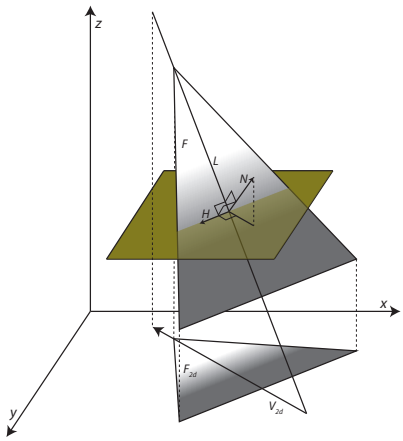


**Figure 6:** *Computing the gradient vector for a projected face*

it lies on $F$ and because it is perpendicular to $H$ is is perpendicular to the gradient contours. Any line through a point on the face (for example, the centroid) with direction $D$ is a suitable $L$.

In the example in Figure 2 each face extends from $Z_{min}$ to $Z_{max}$, so the projected face includes the entire gradient. However, this is true only for very simple paths. Most paths contain areas where the bevel faces do not extend fully to $Z_{max}$ and only part of the gradient will be visible in these areas (Figure 3).

In a perfectly correct 45 degree bevel, there will be no normals that present problems with the above algorithm because they all point upward at 45 degrees. However, in the next section, we introduce variations that may produce faces with normals that point directly upward, or horizontally, or downward. Such faces could also arise from inaccuracies in the bevel algorithm. If a face has a normal that points directly up, it is a horizontal face, so we fill it with the solid color that results from sampling the gradient at the relative height of the face. If the normal is horizontal, it is a vertical face that has no area when projected into two dimensions, so we skip it. If the normal points downward, it is a face of the bevel that is under another face, so again we skip it.

## 3 Variations

There are a number of ways to vary the bevel, creating variations of the contour gradient. However, many of them are not as useful as they might initially seem because they can equally well be achieved by modifying the linear gradients applied to the projected faces.

One simple variation is to truncate the bevel at some maximum height, so that it stops with an inset, horizontal version of the original path. However, this can equally well be achieved by modifying the gradient so that its ending section is a solid color. For example, if the gradient is black at its beginning and white at its end, one can achieve the effect of truncating the bevel at half its original height by adjusting the gradient so that it is black at the beginning, gray in the center, and also gray at the end (Figure 7b).

One could also vary the bevel profile rather than using a simple 45 degree angle. For example, if one used a quarter circle for a bevel profile, the bevel would be very steep near the path edges and flatter in the interior, leading to a contour gradient that changes most quickly near the path and more slowly in the interior. But again, one could achieve the same effect by modifying the gradient, letting its color vary non-linearly (Figure 7c). Modifying the gradient has the additional benefit of avoiding the large number of faces that a curved bevel profile would create.

One useful variation is to perform a horizontal skew operation on the bevel before projecting it into two dimensions. This leads to a non-centered contour gradient with its color contours shifted in one direction. The most pleasing effects occur when the skew is limited to avoid letting the bevel fold over upon itself, leading to faces with normals that point downward. However, by not converting these downward-facing faces, the result is reasonable even in these cases.

One can also use the gradients to control things other than color. Figure 9 uses the linear gradients to control the opacity of solid-colored text, giving a feathered appearance.

## 4 Evaluation

Figure 10 compares our results to the output files generated by Freehand and Fireworks. All three results look fine at the created size. However, scaling the results by 500% makes the non-resolution independence of existing algorithms apparent.

Our method requires curved segments be approximated with a series of line segments. One may ask whether this approximation is visible in the final output. In general, the answer is that they are not. All the figures in this paper were produced with an approximation tolerance of 0.05 point, and the boundaries between faces is not visible. However, Figure 11a shows that small holes or concavities can cause visible artifacts, especially with gradients that have rapid color changes. The concavities lead to fan-shaped faces that make the approximation evident in the center of the shape. However, it is straightforward to estimate the curvature while approximating the path and to use a closer approximation in areas of high curvature. Figure 11b adds the flattening criterion that the direction change between adjacent segments of the same curve may not be more than ten degrees, corresponding to a flatness of .005 point. The rendering artifacts disappear.

## References


ADELSON, E. 2000. Lightness perception and lightness illusions. In *New Cogitive Neurosciences, 2nd Edition*, M. S. Gazzaniga and E. Bizzi, Eds. MIT Press.

ADOBE CREATIVE TEAM. 2012. *Adobe Fireworks CS6 Classroom in a Book*. Adobe Press.

ADOBE SYSTEMS INC. 1999. *PostScript Language Reference (3rd Edition)*. Adobe Press.

ADOBE SYSTEMS INC. 2005. *PDF Reference Version 1.6 (5th Edition)*. Adobe Press.


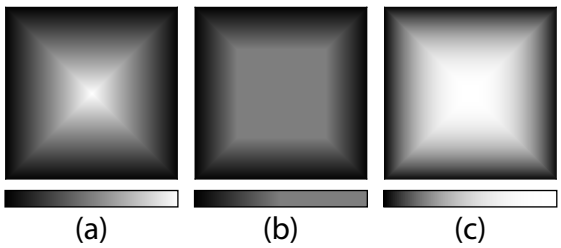


**Figure 7:** *The effect of changing the color distribution in the applied gradients. (a) A basic linear distribution. (b) Having the second part be a uniform color gives the appearance of a truncated bevel. (c) Having a nonlinear distributation gives the appearance of a curved bevel.*

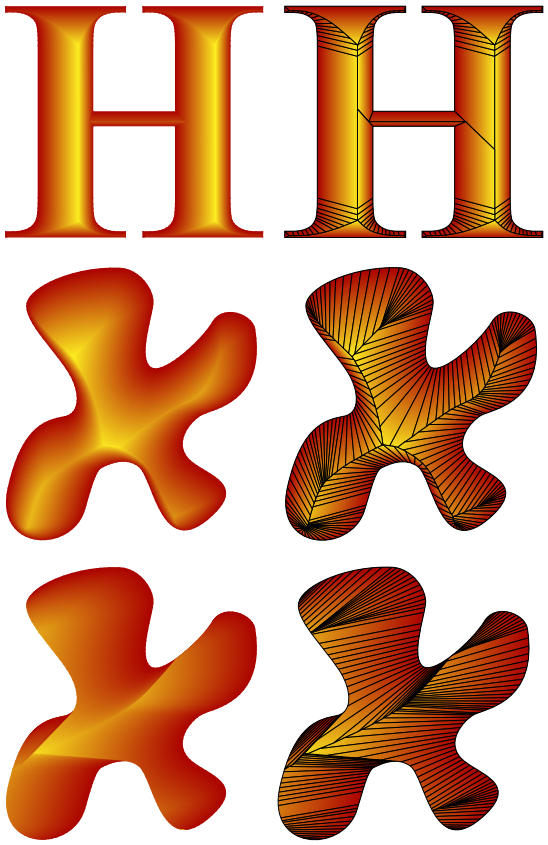

**Figure 8:** *Skewing the 3D bevel before projecting gives noncentered contour gradients. Excessive skewing leads to color discontinuities where the bevel folds over upon itself.*


**Figure 9:** *Creating a feathered appearance by interpreting height as opacity*

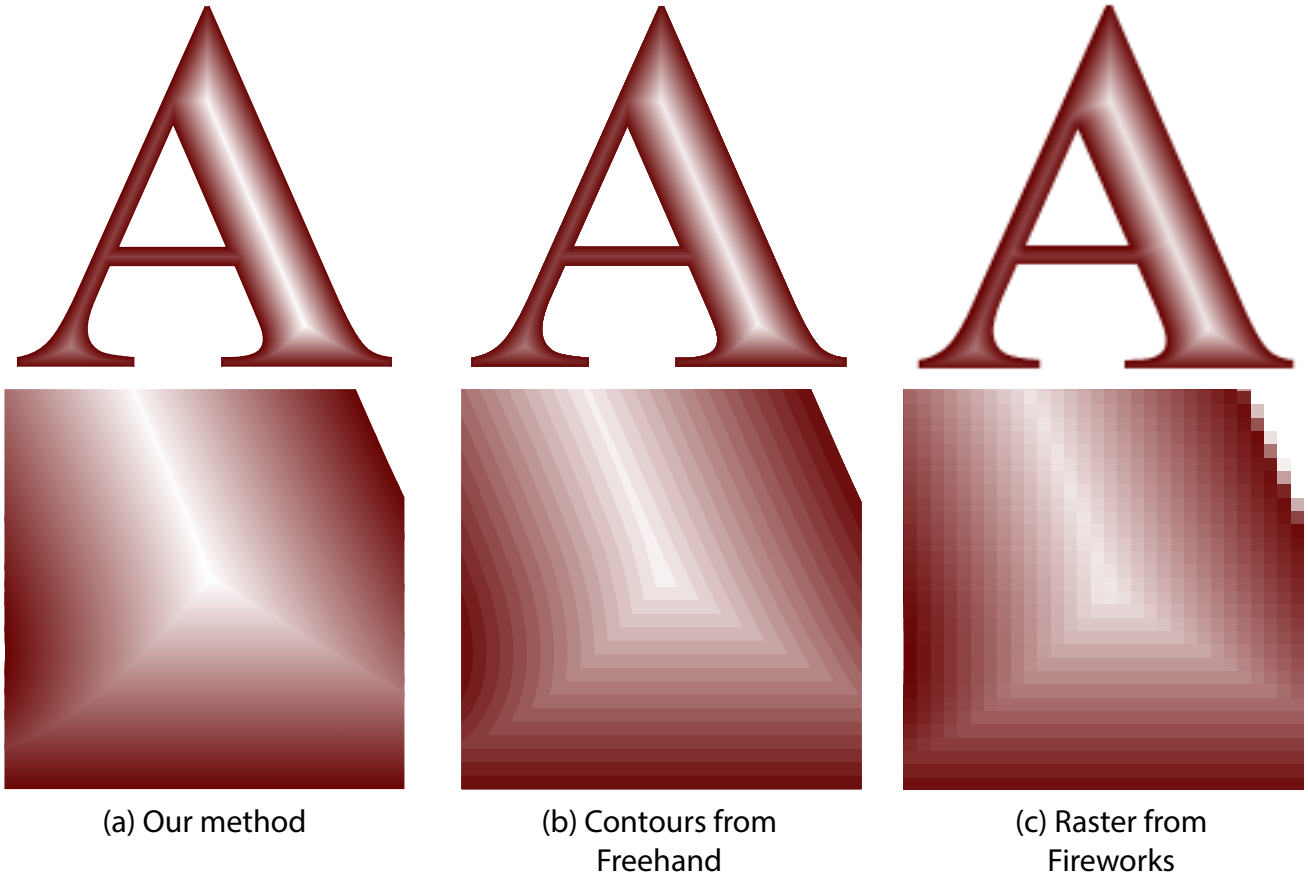


**Figure 10:** *Comparing our method to the output of Freehand and Fireworks on the outline of a 200 point Times New Roman Bold letter "A". We show them at the size they were created, and then scaled up by 500%, showing the bottom of the right stem.*

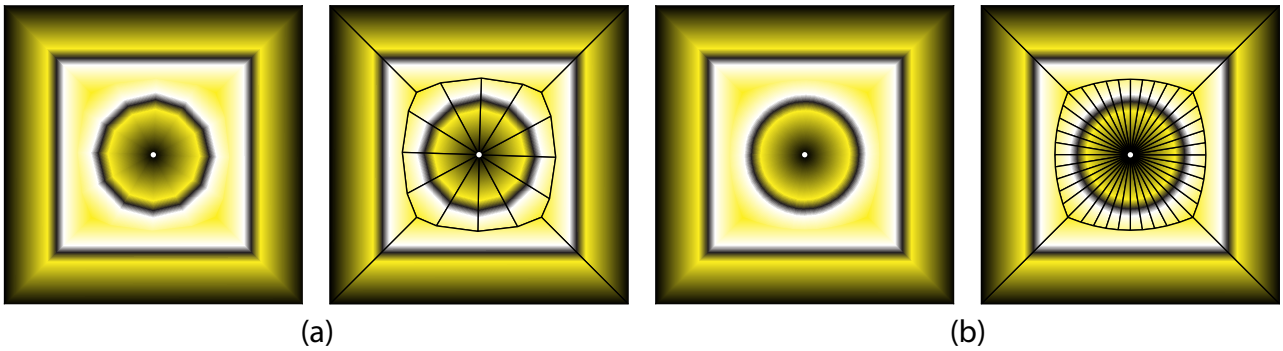


**Figure 11:** *Small concavities and holes cause rendering artifacts (a) that can be addressed by a finer path subdivision (b).*

AICHHOLZER, O., AND AURENHAMMER, F. 1996. Straight skeletons for general polygonal figures in the plane. In *Proceedings of the Second Annual International Conference on Computing and Combinatorics*, Springer-Verlag, London, UK, UK, COCOON '96, 117–126.

BOUTON, G. D. 2012. *CorelDRAW X6 The Official Guide*. McGraw-Hill Osborne Media.

EPPSTEIN, D., AND ERICKSON, J. G. 1998. Raising roofs, crashing cycles, and playing pool: applications of a data structure for finding pairwise interactions. In *Proc. 14th Symp. Computational Geometry*, ACM, 58–67.

SCHULZE, P. 2003. *Macromedia FreeHand MX: Training from the Source*. Macromedia Press.

WORLD WIDE WEB CONSORTIUM (W3C), 2011. Scalable vector graphics (SVG) 1.1 (second edition), August.