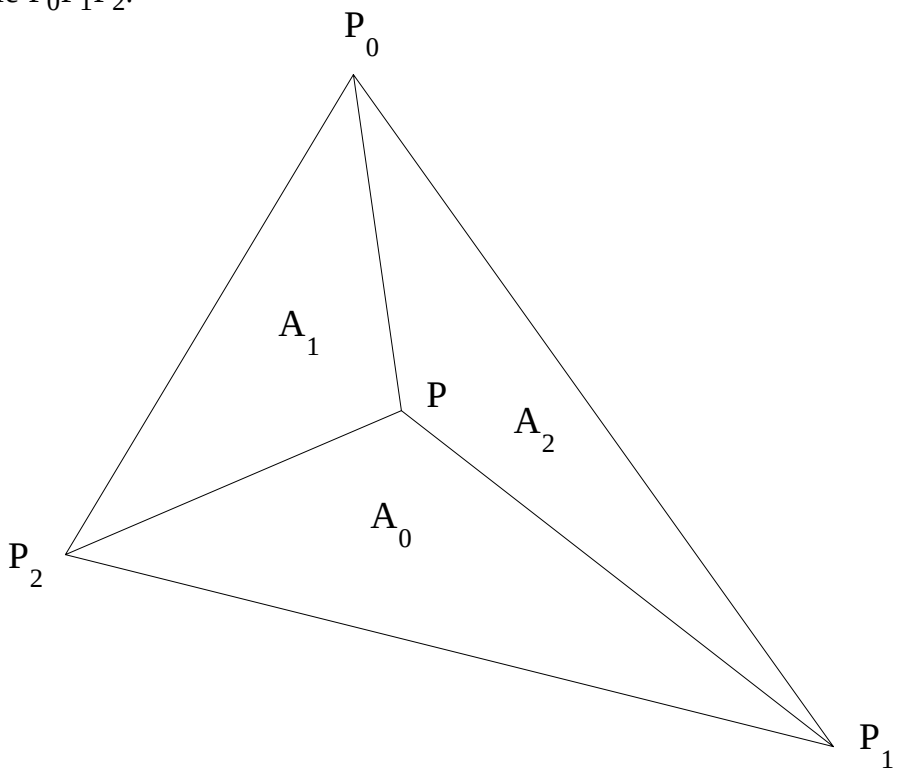
Barycentric Coordinates as Interpolants

Russ Brown

Consider the triangle $P_0P_1P_2$:



The points P_0, P_1 and P_2 are the vertices of the triangle. P is a point at which interpolation will be performed. For purposes of this document, point P_0 has cartesian coordinates (x_0,y_0,w_0), color coordinates (r_0,g_0,b_0) and texture coordinates (u_0,v_0). Points P_1 and P_2 are similarly defined. The cartesian coordinates are defined in perspective space, i.e., they have been multiplied by a perspective matrix and perspective divided. The color and texture coordinates are defined in world space. Note that we use the w-coordinate instead of z. We use this convention because w represents eye space z, i.e., the z-coordinate by which eye space (X,Y) coordinates are divided to produce perspective space (x,y) coordinates. As we will see, w is used to undo the perspective division so that interpolation occurs in eye space, which is equivalent to interpolation in world space. Moreover, for depth priority calculations, w is preferred to perspective z when a fixed-point representation is used for range priority calculations.

The first step of the shading calculation is to construct a set of interpolants for point P situated at each pixel. Barycentric coordinate interpolants may be calculated in a simple and straightforward manner from points P, P_0, P_1 and P_2, as follows. If we define

A_0 to be the screen-space area of triangle PP_1P_2
A_1 to be the screen-space area of triangle PP_2P_0
A_2 to be the screen-space area of triangle PP_0P_1

then we can calculate the barycentric coordinates b_0, b_1 and b_2 as

$$b_0 = A_0/(A_0 + A_1 + A_2)$$
$$b_1 = A_1/(A_0 + A_1 + A_2)$$
$$b_2 = A_2/(A_0 + A_1 + A_2) \qquad \text{(eq. 1)}$$

Note that, because division is performed using the sum of triangle areas, $b_0 + b_1 + b_2 = 1$. Also, if any $b_i < 0$, the pixel is outside the edge opposite P_i, a fact useful in scan conversion.

The areas of triangles PP_1P_2, PP_2P_0 and PP_0P_1 may be computed in a trivial manner. For example, the area A_0 of triangle PP_1P_2 is one-half the magnitude of the cross product $PP_1 \times PP_2$. However, since the factor of one-half appears in both the numerator and denominator of eq. 1, we can ignore this factor and calculate A_0 as

$$A_0 = |PP_1 \times PP_2| = \begin{vmatrix} x_1-x & y_1-y \\ x_2-x & y_2-y \end{vmatrix}$$

$$= (x_1-x)(y_2-y) - (x_2-x)(y_1-y)$$

$$= (y_1-y_2)x + (x_2-x_1)y + (x_1y_2-x_2y_1)$$

$$= \alpha_0 x + \beta_0 y + \gamma_0 \qquad \text{(eq. 2a)}$$

Values of α, β and γ are analogously defined for computing A_1 and A_2. Equations for these triangle areas may be generated via cyclic permutation of the indices 0, 1 and 2

$$A_1 = |PP_2 \times PP_0| = (y_2-y_0)x + (x_0-x_2)y + (x_2y_0-x_0y_2) = \alpha_1 x + \beta_1 y + \gamma_1$$
$$A_2 = |PP_0 \times PP_1| = (y_0-y_1)x + (x_1-x_0)y + (x_0y_1-x_1y_0) = \alpha_2 x + \beta_2 y + \gamma_2 \qquad \text{(eq. 2b)}$$

Thus to calculate the barycentric coordinates for a particular pixel, we need to compute three areas as in eq. 2, then use these areas as in eq. 1. Barycentric coordinates calculated in this manner have the problem that they are defined in perspective space, so any interpolation using them will produce nonlinear errors in eye space. To correct this error, the color and texture coordinates may be transformed into screen space, interpolated in screen space, and then transformed back into eye space, as discussed in the paper "Modeling Specular Highlights using Bézier Triangles." The result may be expressed in terms of transformed barycentric coordinates. To accomplish the tranformation, one simply multiplies the areas of triangles PP_1P_2, PP_2P_0 and PP_0P_1 by the w-coordinates at the two exterior vertices of each triangle to obtain

$$b_0 = w_1w_2A_0/(w_1w_2A_0 + w_2w_0A_1 + w_0w_1A_2)$$
$$b_1 = w_2w_0A_1/(w_1w_2A_0 + w_2w_0A_1 + w_0w_1A_2)$$
$$b_2 = w_0w_1A_2/(w_1w_2A_0 + w_2w_0A_1 + w_0w_1A_2) \qquad \text{(eq. 3)}$$

This surprisingly simple result may be verified by inspection of eq. 2a and remembering that the perspective space (x,y) coordinates are derived from their eye space (X,Y) counterparts through division by w

$$x_1 = X_1 / w_1$$
$$y_1 = Y_1 / w_1$$
$$x_2 = X_2 / w_2$$
$$y_2 = Y_2 / w_2 \qquad \text{(eq. 4)}$$

Hence multiplication of eq. 2a by $w_1 w_2$ undoes the perspective division.

Once the barycentric coordinates have been transformed back into eye space, linear interpolation of color and texture coordinates may be accomplished by a series of dot products, for example

$$g = b_0 g_0 + b_1 g_1 + b_2 g_2$$
$$u = b_0 u_0 + b_1 u_1 + b_2 u_2 \qquad \text{(eq. 5)}$$

This approach is useful not only for interpolation of texture coordinates u and v, but also for calculation of the proper mip-map depth. In his paper "Pyramidal Parametrics", Lance Williams advocates the use of the partial derivatives $\partial u / \partial x$, $\partial u / \partial y$, $\partial v / \partial x$ and $\partial v / \partial y$ for this purpose. These partial derivatives may be calculated analytically because u and v are functions of A_0, A_1 and A_2 (eqs. 3 and 5), which are in turn functions of x and y (eq. 2). Applying the chain rule of partial differentiation creates the following formulae

$$\partial u / \partial x = [w_1 w_2 \alpha_0 (u_0 - u) + w_2 w_0 \alpha_1 (u_1 - u) + w_0 w_1 \alpha_2 (u_2 - u)] / (w_1 w_2 A_0 + w_2 w_0 A_1 + w_0 w_1 A_2)$$
$$\partial u / \partial y = [w_1 w_2 \beta_0 (u_0 - u) + w_2 w_0 \beta_1 (u_1 - u) + w_0 w_1 \beta_2 (u_2 - u)] / (w_1 w_2 A_0 + w_2 w_0 A_1 + w_0 w_1 A_2)$$
$$\partial v / \partial x = [w_1 w_2 \alpha_0 (v_0 - v) + w_2 w_0 \alpha_1 (v_1 - v) + w_0 w_1 \alpha_2 (v_2 - v)] / (w_1 w_2 A_0 + w_2 w_0 A_1 + w_0 w_1 A_2)$$
$$\partial v / \partial y = [w_1 w_2 \beta_0 (v_0 - v) + w_2 w_0 \beta_1 (v_1 - v) + w_0 w_1 \beta_2 (v_2 - v)] / (w_1 w_2 A_0 + w_2 w_0 A_1 + w_0 w_1 A_2)$$

(eq. 6a)

A sometimes more useful form of this equation may be obtained by taking both ∂x and ∂y to be the pixel spacing S, and by multiplying both sides of the equation by S. Then the total differentials du and dv represent the instantaneous rate of change of u and v at the pixel

$$du = S[w_1 w_2 (\alpha_0 + \beta_0)(u_0 - u) + w_2 w_0 (\alpha_1 + \beta_1)(u_1 - u) + w_0 w_1 (\alpha_2 + \beta_2)(u_2 - u)] / (w_1 w_2 A_0 + w_2 w_0 A_1 + w_0 w_1 A_2)$$

$$dv = S[w_1 w_2 (\alpha_0 + \beta_0)(v_0 - v) + w_2 w_0 (\alpha_1 + \beta_1)(v_1 - v) + w_0 w_1 (\alpha_2 + \beta_2)(v_2 - v)] / (w_1 w_2 A_0 + w_2 w_0 A_1 + w_0 w_1 A_2) \qquad \text{(eq. 6b)}$$

In this equation, u and v are the values interpolated at the pixel via eq. 5, α and β are defined in eq. 2, and the denominator is the same as in eq. 3.

Inspection of eqs. 2, 3 and 6 suggests that the setup computation for each polygon involves the creation of nine terms: $w_1w_2\alpha_0$, $w_1w_2\beta_0$, $w_1w_2\gamma_0$, $w_2w_0\alpha_1$, $w_2w_0\beta_1$, $w_2w_0\gamma_1$, $w_0w_1\alpha_2$, $w_0w_1\beta_2$, and $w_0w_1\gamma_2$. Following this setup, the areas of triangles PP_1P_2, PP_2P_0 and PP_0P_1 may be computed on a per-pixel basis as

$$A_0 = w_1w_2\alpha_0x + w_1w_2\beta_0y + w_1w_2\gamma_0$$
$$A_1 = w_2w_0\alpha_1x + w_2w_0\beta_1y + w_2w_0\gamma_1$$
$$A_2 = w_0w_1\alpha_2x + w_0w_1\beta_2y + w_0w_1\gamma_2 \qquad \text{(eq. 7)}$$

The areas can then be used on a per-pixel basis to compute barycentric coordinates b_0, b_1 and b_2 as in eq. 1, then those barycentric coordinates can be used to linearly interpolate color and texture coordinates as in eq. 5, as well as partial derivatives and total differentials as in eq. 6. One more optimization is possible. The area equations in eq. 7 only need to be calculated for the first pixel shaded for each triangle, and may therefore be included in the setup computation for each triangle. Thereafter, these areas may be updated by addition or subtraction via a delta calculation. For example, if the next pixel is located one pixel (i.e., one unit) removed in the positive x-direction, then the areas may be updated by addition of $w_1w_2\alpha_0$, $w_2w_0\alpha_1$ and $w_0w_1\alpha_2$ to the results for the previous pixel. Similarly, if the next pixel is located one pixel away in the negative y-direction, then the areas may be updated by subtraction of $w_1w_2\beta_0$, $w_2w_0\beta_1$ and $w_0w_1\beta_2$ from the results for the previous pixel. So a scan conversion algorithm that proceeds from one pixel to an adjacent pixel permits calculation of barycentric coordinate interpolants via five additions, one reciprocal and three multiplies, as can be appreciated from inspection of eqs. 1 and 7.

Note that the three products w_1w_2, w_2w_0 and w_0w_1 appear in both the numerator and denominator of eqs. 3 and 7, so we can block normalize these products and keep only the mantissas of this operation. For example, if the w-coordinates are defined as 15-bit integers, then the products will be 30-bit integers which we can block normalize back to 15-bit integers, discarding the exponent of the normalization. Further block normalization may be possible for the $w_1w_2\alpha_0$, $w_1w_2\beta_0$, $w_1w_2\gamma_0$, $w_2w_0\alpha_1$, $w_2w_0\beta_1$, $w_2w_0\gamma_1$, $w_0w_1\alpha_2$, $w_0w_1\beta_2$, and $w_0w_1\gamma_2$ terms. The aim of this block normalization is to permit the use of integer arithmetic.