Rational-Linear Interpolation of Texture Coordinates and Their Partial Derivatives

Kevin Wu
Computer Systems Laboratory
HPL-98-113
June, 1998

E-mail: kevinwu@hpl.hp.com

computer graphics,
texture mapping,
interpolation,
texture coordinates,
texture derivatives,
anti-aliasing

This paper describes an algorithm for calculating texture coordinates and their partial derivatives during scan conversion of planar polygons. These values are required in texture mapping for anti-aliasing algorithms, where a resampling filter combines texture elements (a.k.a. texels) in a weighted average. Perspective projection requires an image warp accomplished with rational-linear (a.k.a. hyperbolic) interpolation. A single division per pixel and a few addition and multiplication operations yields the texture coordinates and their partial derivatives.

Internal Accession Date Only

1. Introduction

Texture mapping of planar polygons viewed with perspective projection is accomplished with rational-linear interpolation [1], which also is known as hyperbolic interpolation [2]. The procedure for calculating texture coordinates at each pixel is as follows:

1. Apply a perspective warp (from texture space to screen space) to the texture coordinates at vertices.

2. Linearly interpolate the warped texture coordinates down edges and across spans in screen space.

3. At each pixel, apply the inverse perspective warp (from screen space to texture space) to the warped texture coordinates to recover the texture coordinates.

The perspective warp and its inverse are achieved by division of a homogeneous coordinate. Division is a relatively slow operation, typically requiring on the order of ten clock cycles or more with partial pipelining or none at all. Hence, the inverse perspective warp at each pixel often is implemented with a single division for the reciprocal of the homogeneous coordinate and one multiplication per texture coordinate. This usually is faster than performing a division for each texture coordinate.

Fast shadows and lighting effects can be implemented with texture mapping hardware by augmenting the texture coordinates with their own homogeneous coordinate [3]. The spatial coordinates have a homogeneous coordinate for the perspective projection to the screen. Similarly, a light can project its field onto a texture via a homogeneous coordinate associated with the texture coordinates. These lighting effects can be implemented easily on conventional texture mapping hardware simply by modifying the divisor in the inverse perspective warp performed at each pixel.

In addition to calculating the texture coordinates at each pixel, texture-mapping implementations compute the partial derivatives of the texture coordinates with respect to the screen's spatial coordinates. These partial derivatives appear in the vector gradient operator, which yields the maximum directional derivative of a scalar field, and in the Jacobian, which yields the factor needed for changing variables in a multiple integral [4]. Subsequently, these partial derivatives are sometimes called the *gradients* or *Jacobian components*, respectively.

Texture-mapping implementations use the partial derivatives of the texture coordinates for anti-aliasing algorithms. Convolution of a resampling filter with a texture image is an operation that averages the weighted texture elements (a.k.a. texels) [5]. Filtering can be expensive so real-time systems benefit from doing some filtering in advance as a preprocessing step. A popular storage structure for retaining the preprocessed texture is a MIP map, which is a multi-resolution texture in the form of an image pyramid [6]. Heckbert investigated methods of selecting the resolution

level within the image pyramid and suggested taking the maximum length of the two Jacobian basis vectors [7].

Isotropic or space-invariant filtering based on bilinear or trilinear filtering [8] of MIP map texels works well for polygons parallel to the screen, but textures become blurred as polygons tilt toward becoming parallel to the line of sight. The footprint of a pixel projected into texture space changes in shape and size as scan conversion moves from pixel to pixel across the polygon viewed in perspective. Anisotropic or space-variant filtering considers the change in footprint shape and size, and the result is sharper textures for polygons tilted toward the horizon. A filter with a good balance of quality and performance is the Elliptical Weighted Average (EWA) filter [9] combined with a MIP map [5, 10]. The shape and size of a pixel footprint in texture space for the EWA filter is an ellipse fitting inside a parallelogram, whose sides are the Jacobian basis vectors. Footprint assembly [11] is a real-time approximation of the EWA filter, and it derives the footprint shape from the Jacobian basis vectors.

For every pixel, texture-mapping implementations need the texture coordinates for retrieving texels to be mapped to the screen and the partial derivatives of the texture coordinates for anti-aliasing algorithms. This paper describes an algorithm for calculating these in a scan-line method that requires only a single division per pixel. Then, one multiplication per texture coordinate yields the texture coordinates. In addition, one multiplication per pixel plus half of one multiplication per partial derivative yields the partial derivatives.

2. Background

We seek an algorithm for interpolating texture coordinates and their partial derivatives for planar polygons viewed with a perspective projection. This section briefly describes the background material: rational-linear interpolation and fast shadows and lighting effects.

2.1 Rational-Linear Interpolation

Rational-linear interpolation described by Heckert and Moreton [1] and hyperbolic interpolation described by Blinn [2] are equivalent methods so we will use the name *rational-linear interpolation* hereafter. It is the most efficient method for interpolating coordinates on planar polygons transformed by a perspective projection. Based on Blinn's description, a summary of the procedure is as follows:

1. Let the spatial coordinates in homogeneous screen space be $\begin{bmatrix}\tilde{x} & \tilde{y} & \tilde{z} & \tilde{w}\end{bmatrix}$. These coordinates are obtained by transforming polygon vertices from model space via a 4×4 homogeneous matrix incorporating a perspective projection. Let the texture coordinates be $\begin{bmatrix}\tilde{s} & \tilde{t}\end{bmatrix}$. These are bound to the polygon vertices. Construct an array of coordinates for each vertex: $\begin{bmatrix}\tilde{x} & \tilde{y} & \tilde{z} & \tilde{w} & \tilde{s} & \tilde{t} & 1\end{bmatrix}$.

2. Perform view clipping of the polygons and linearly interpolate all coordinates to clipping boundaries wherever clipping occurs.

3. Perform projection to screen space by dividing all coordinates by the homogeneous spatial coordinate $\tilde{w}$. In practice, calculating the reciprocal $1/\tilde{w}$ and multiplying it by all the coordinates is faster. This yields $\begin{bmatrix}\tilde{x}/\tilde{w} & \tilde{y}/\tilde{w} & \tilde{z}/\tilde{w} & 1 & \tilde{s}/\tilde{w} & \tilde{t}/\tilde{w} & 1/\tilde{w}\end{bmatrix}$. Rename this array to $\begin{bmatrix}x & y & z & 1 & s & t & q\end{bmatrix}$.

4. Linearly interpolate all the coordinates down polygon edges and across spans.

5. At each pixel, perform the *z* buffer test, and on success, project the warped texture coordinates back to texture space by dividing by the homogeneous texture coordinate q: $\begin{bmatrix}\hat{s} & \hat{t}\end{bmatrix} = \begin{bmatrix}s/q & t/q\end{bmatrix}$.

6. Map the texels around $\begin{bmatrix}\hat{s} & \hat{t}\end{bmatrix}$ to the pixel with an anti-aliasing algorithm.

Blinn notes that besides spatial and texture coordinates, other values such as normal and color coordinates can be interpolated in a like manner. In practice, real-time systems rarely apply rational-linear interpolation to normals and color. Graphics hardware that performs a lighting calculation (requiring normals) per pixel is rare because the calculation is slow and expensive. Colors often undergo only linear interpolation instead of rational-linear interpolation because the calculation is faster and visual artifacts are more difficult for human viewers to detect for colors than texture coordinates.

The procedure warps the texture coordinates to screen space, linearly interpolates the warped texture coordinates in screen space, and applies the inverse warp to the warped texture coordinates. The texture coordinates take the general form

$$\hat{s} = \frac{s}{q} = \frac{\tilde{s}/\tilde{w}}{1/\tilde{w}} = \tilde{s} \qquad \text{and} \qquad \hat{t} = \frac{t}{q} = \frac{\tilde{t}/\tilde{w}}{1/\tilde{w}} = \tilde{t}\,. \tag{1}$$

Thus, we recover the original texture coordinates bound to the polygon vertices. For scan-line interpolation, the incremental form of the texture coordinates is

$$\hat{s}_{i+1} = \frac{s_{i+1}}{q_{i+1}} = \frac{s_i + \Delta x \dfrac{\partial s}{\partial x}}{q_i + \Delta x \dfrac{\partial q}{\partial x}} \qquad \text{and} \qquad \hat{t}_{i+1} = \frac{t_{i+1}}{q_{i+1}} = \frac{t_i + \Delta x \dfrac{\partial t}{\partial x}}{q_i + \Delta x \dfrac{\partial q}{\partial x}} \tag{2}$$

where the spacing between pixels Δx is usually 1 in screen space, so it can be ignored safely. In Equation 2, note that we can write finite differences in place of differentials because the texture

coordinates in screen space vary linearly with the spatial coordinates in screen space. The partial derivatives are constant across the planar polygon in screen space so they can be calculated once per polygon. Since division is a slow operation, a faster method than dividing each texture coordinate by q_{i+1} is to calculate the reciprocal $1/q_{i+1}$ and multiply it by each texture coordinate.

2.2 Fast Shadows and Lighting Effects

Segal *et al.* described a method of implementing fast shadows and lighting effects with a simple extension to texture mapping hardware [3]. The method introduces a new space called the *light coordinate system*, where points are described by the vector $\begin{bmatrix} x^l & y^l & z^l & w^l \end{bmatrix}$. This is a homogeneous texture coordinate system, where the texture coordinates x^t and y^t are obtained by dividing by the homogeneous coordinate. The primary mathematical result from that paper is:

$$x^t = \frac{x^l}{w^l} = \frac{x^l/\tilde{w}}{w^l/\tilde{w}} \qquad \text{and} \qquad y^t = \frac{y^l}{w^l} = \frac{y^l/\tilde{w}}{w^l/\tilde{w}} \tag{3}$$

where $\tilde{w}$ is the homogeneous coordinate of homogeneous screen space as before.

Let us make the substitutions $\hat{s} = x^t$, $\hat{t} = y^t$, $\tilde{s} = x^l$, $\tilde{t} = y^l$, and $\tilde{q} = w^l$, then

$$\hat{s} = \frac{\tilde{s}}{\tilde{q}} = \frac{\tilde{s}/\tilde{w}}{\tilde{q}/\tilde{w}} \qquad \text{and} \qquad \hat{t} = \frac{\tilde{t}}{\tilde{q}} = \frac{\tilde{t}/\tilde{w}}{\tilde{q}/\tilde{w}}. \tag{4}$$

Equations 1 and 4 are remarkably similar. Segal *et al.* state that quantities in the numerators and denominators of Equation 3 should be linearly interpolated and the divisions should be performed at each pixel.

As before, let $s = \tilde{s}/\tilde{w}$, $t = \tilde{t}/\tilde{w}$, and $q = \tilde{q}/\tilde{w}$, then Equation (4) becomes

$$\hat{s} = \frac{\tilde{s}}{\tilde{q}} = \frac{\tilde{s}/\tilde{w}}{\tilde{q}/\tilde{w}} = \frac{s}{q} \qquad \text{and} \qquad \hat{t} = \frac{\tilde{t}}{\tilde{q}} = \frac{\tilde{t}/\tilde{w}}{\tilde{q}/\tilde{w}} = \frac{t}{q}. \tag{5}$$

Equation 5 is the general form of Equation 1 for which $\tilde{q} = 1$ because the texture coordinates $\tilde{s}$ and $\tilde{t}$ are bound to the vertices. In the general case, the texture coordinates are transformed by a homogeneous matrix incorporating a perspective projection. This leads to a value of $\tilde{q}$ not equal to 1, in general. The general procedure for interpolating texture coordinates is similar to the special case in Section 2.1 except that the denominator is $q = \tilde{q}/\tilde{w}$ instead of $1/\tilde{w}$.

OpenGL [12, 13] is a de facto industry-standard 3D graphics application programming interface (API) for which Segal is a co-architect [14]. OpenGL has a 4×4 homogeneous texture matrix for transforming the texture coordinates $\begin{bmatrix} s' & t' & r' & q' \end{bmatrix}$, which are bound to the vertices. This transformation yields the texture coordinates $\begin{bmatrix} \tilde{s} & \tilde{t} & \tilde{r} & \tilde{q} \end{bmatrix}$ in lighting space. The procedure in Section 2.1 is applicable to this general case simply by augmenting the vertex arrays with the extra texture mapping coordinates: $\begin{bmatrix} \tilde{x} & \tilde{y} & \tilde{z} & \tilde{w} & \tilde{s} & \tilde{t} & \tilde{r} & \tilde{q} \end{bmatrix}$. Section 4 lists the individual steps for the general case in detail.

3. Partial Derivatives

Anti-aliasing algorithms require the partial derivatives of the texture coordinates in texture space with respect to the spatial coordinates in screen space for determining the size of a pixel's footprint in texture space. Starting from Equation 5, we take the partial derivatives with respect to x:

$$\frac{\partial \hat{s}}{\partial x} = \frac{\partial}{\partial x}\left(\frac{s}{q}\right) = \frac{q\dfrac{\partial s}{\partial x} - s\dfrac{\partial q}{\partial x}}{q^2} \qquad \text{and} \qquad \frac{\partial \hat{t}}{\partial x} = \frac{\partial}{\partial x}\left(\frac{t}{q}\right) = \frac{q\dfrac{\partial t}{\partial x} - t\dfrac{\partial q}{\partial x}}{q^2}. \tag{6}$$

The partial derivatives $\partial s/\partial x$, $\partial t/\partial x$, and $\partial q/\partial x$ are constant across a planar polygon, and they need to be calculated for Equation 2 to incrementally update the texture coordinates. For a triangle, they can be calculated by

- sorting the vertices from top to bottom,
- horizontally extending the middle vertex (point *A*) to the point on the edge joining the top and bottom vertices (point *B*),
- linearly interpolating the values between the top and bottom vertices to point *B*, and
- calculating the ratios of finite differences for points *A* and *B*.

Equation 2 requires us to calculate $1/q$ at each pixel. We simply square it and multiply by the terms in the numerators of Equation 6. Thus, these partial derivatives require only one multiplication per pixel, and three multiplications and one subtraction per texture coordinate.

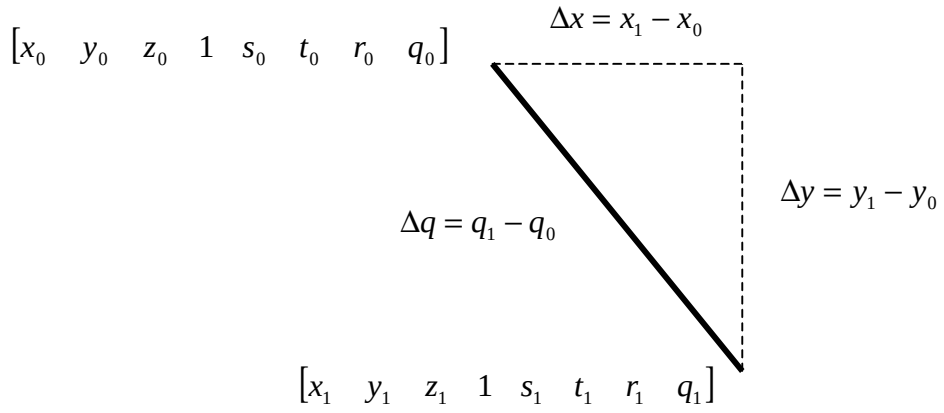

Figure 1: A non-horizontal polygon edge in screen space.

For the partial derivatives with respect to y, consider a non-horizontal polygon edge for which $\Delta y \neq 0$ as illustrated in Figure 1. Each texture coordinate in screen space is a linear function of only x and y. The differentials are

$$ds = \frac{\partial s}{\partial x}dx + \frac{\partial s}{\partial y}dy, \qquad dt = \frac{\partial t}{\partial x}dx + \frac{\partial t}{\partial y}dy, \qquad \text{and} \qquad dq = \frac{\partial q}{\partial x}dx + \frac{\partial q}{\partial y}dy.$$

Since the texture coordinates vary linearly with x and y, the differentials can be replaced by finite differences. Solving for the partial derivatives with respect to y gives

$$\frac{\partial s}{\partial y} = \frac{\Delta s}{\Delta y} - \frac{\partial s}{\partial x}\frac{\Delta x}{\Delta y}, \qquad \frac{\partial t}{\partial y} = \frac{\Delta t}{\Delta y} - \frac{\partial t}{\partial x}\frac{\Delta x}{\Delta y}, \qquad \text{and} \qquad \frac{\partial q}{\partial y} = \frac{\Delta q}{\Delta y} - \frac{\partial q}{\partial x}\frac{\Delta x}{\Delta y}. \tag{7}$$

These partial derivatives are constant across a planar polygon, and we can calculate them from the finite differences of a non-horizontal edge and the partial derivatives $\partial s/\partial x$, $\partial t/\partial x$, and $\partial q/\partial x$.

Using the results in Equation 7, the partial derivatives of the texture coordinates in texture space with respect to the spatial coordinates in screen space are

$$\frac{\partial \hat{s}}{\partial y} = \frac{\partial}{\partial y}\left(\frac{s}{q}\right) = \frac{q\dfrac{\partial s}{\partial y} - s\dfrac{\partial q}{\partial y}}{q^2} \qquad \text{and} \qquad \frac{\partial \hat{t}}{\partial y} = \frac{\partial}{\partial y}\left(\frac{t}{q}\right) = \frac{q\dfrac{\partial t}{\partial y} - t\dfrac{\partial q}{\partial y}}{q^2}. \tag{8}$$

Equations 6 and 8 can be calculated with three multiplications and one subtraction per partial derivative. We can reduce the operations by substituting the incremental forms of the variables into Equations 6 and 8. The incremental forms of the variables are

$$s_{i+1} = s_i + \Delta x \frac{\partial s}{\partial x}, \qquad t_{i+1} = t_i + \Delta x \frac{\partial t}{\partial x}, \qquad \text{and} \qquad q_{i+1} = q_i + \Delta x \frac{\partial q}{\partial x}. \tag{9}$$

Substituting Equation 9 into the numerators of Equations 6 and 8 gives

$$\begin{aligned}
\left(q_{i+1}\frac{\partial s}{\partial x} - s_{i+1}\frac{\partial q}{\partial x}\right) &= \left(q_i\frac{\partial s}{\partial x} - s_i\frac{\partial q}{\partial x}\right) + \Delta x\left(\frac{\partial q}{\partial x}\frac{\partial s}{\partial x} - \frac{\partial s}{\partial x}\frac{\partial q}{\partial x}\right) = \left(q_i\frac{\partial s}{\partial x} - s_i\frac{\partial q}{\partial x}\right) \\
\left(q_{i+1}\frac{\partial t}{\partial x} - t_{i+1}\frac{\partial q}{\partial x}\right) &= \left(q_i\frac{\partial t}{\partial x} - t_i\frac{\partial q}{\partial x}\right) + \Delta x\left(\frac{\partial q}{\partial x}\frac{\partial t}{\partial x} - \frac{\partial t}{\partial x}\frac{\partial q}{\partial x}\right) = \left(q_i\frac{\partial t}{\partial x} - t_i\frac{\partial q}{\partial x}\right) \\
\left(q_{i+1}\frac{\partial s}{\partial y} - s_{i+1}\frac{\partial q}{\partial y}\right) &= \left(q_i\frac{\partial s}{\partial y} - s_i\frac{\partial q}{\partial y}\right) + \Delta x\left(\frac{\partial q}{\partial x}\frac{\partial s}{\partial y} - \frac{\partial s}{\partial x}\frac{\partial q}{\partial y}\right) \\
\left(q_{i+1}\frac{\partial t}{\partial t} - t_{i+1}\frac{\partial q}{\partial y}\right) &= \left(q_i\frac{\partial t}{\partial y} - t_i\frac{\partial q}{\partial y}\right) + \Delta x\left(\frac{\partial q}{\partial x}\frac{\partial t}{\partial y} - \frac{\partial t}{\partial x}\frac{\partial q}{\partial y}\right)
\end{aligned} \tag{10}$$

where the spacing between pixels Δx is usually 1 in screen space so it can be ignored safely. All the partial derivatives in Equation 10 are constant for a planar polygon so the constant increments can be calculated once per polygon.

For scan-line interpolation, the constant increments vanish identically in the numerators of $\partial\hat{s}/\partial x$ and $\partial\hat{t}/\partial x$, and the numerators remain constant across the scan line. Since the denominators of these partial derivatives are equal, the direction of the Jacobian basis vector $\left[\partial\hat{s}/\partial x \quad \partial\hat{t}/\partial x\right]$ remains constant across the scan line, and only the magnitude changes. This result could be anticipated from the property that a perspective projection maps straight lines to straight lines. In particular, an incremental change in x in screen space leads to an incremental change in texture space where the direction given by $\left[\partial\hat{s}/\partial x \quad \partial\hat{t}/\partial x\right]$ is independent of the size of the increment. However, the same incremental change in x in screen space preserves neither direction nor magnitude in the other Jacobian basis vector $\left[\partial\hat{s}/\partial y \quad \partial\hat{t}/\partial y\right]$. This is illustrated in Figure 2.

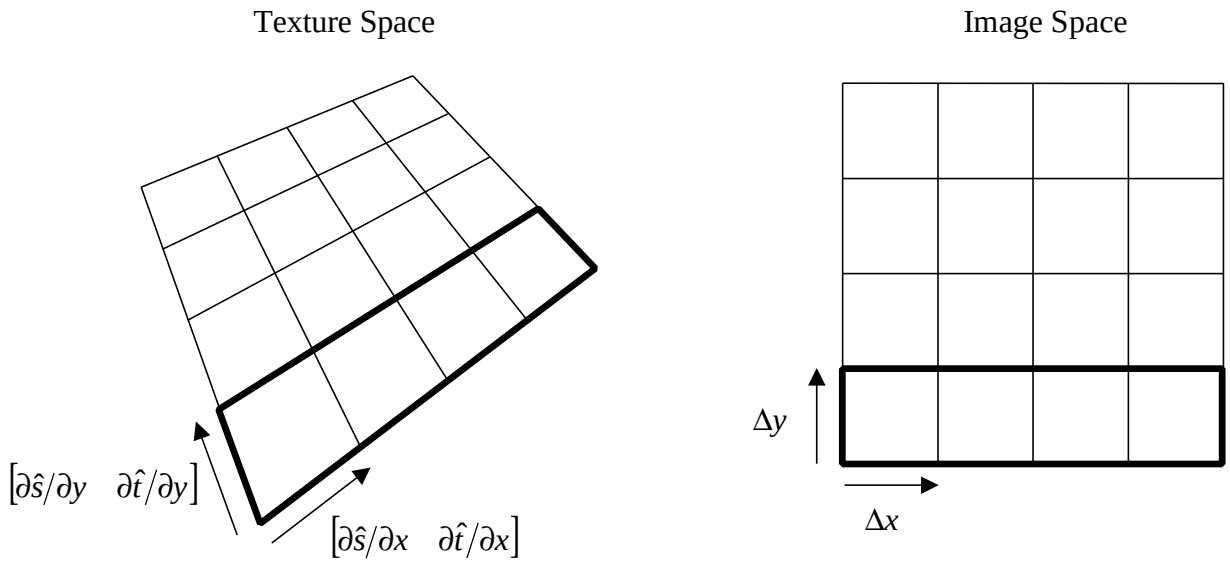

Figure 2: Direction and magnitude of Jacobian basis vectors across a scan line.

From a computational viewpoint, we can interpolate four new variables — the numerators of the partial derivatives — down polygon edges. These new variables are linear combinations of variables that vary linearly in screen space so the new variables also vary linearly in screen space. Equation 10 suggests that we can interpolate two of the four new variables across a scan line, and Equations 6 and 8 yield the partial derivatives with one multiplication per pixel and one multiplication per partial derivative.

4. Interpolation of Texture Coordinates and Partial Derivatives

4.1 General Algorithm

In this section, we tie together the equations of the previous sections to arrive at the general algorithm for calculating texture coordinates and partial derivatives for a planar polygon viewed with perspective. Planarity applies to position and texture coordinates.

1. Let the spatial coordinates in homogeneous screen space be $\left[\tilde{x} \quad \tilde{y} \quad \tilde{z} \quad \tilde{w}\right]$. These coordinates are obtained by transforming polygon vertices from model space via a 4×4 homogeneous geometry matrix incorporating a perspective projection. Let the texture coordinates in lighting space be $\left[\tilde{s} \quad \tilde{t} \quad \tilde{r} \quad \tilde{q}\right]$. These coordinates are obtained by transforming the texture coordinates bound to the vertices via a 4×4 homogeneous texture

matrix. For each vertex, construct an array of coordinates: $\begin{bmatrix}\tilde{x} & \tilde{y} & \tilde{z} & \tilde{w} & \tilde{s} & \tilde{t} & \tilde{r} & \tilde{q}\end{bmatrix}$.

2. For each polygon, perform view clipping and linearly interpolate all coordinates to clipping boundaries wherever clipping occurs.

3. Project the vertices to screen space: for each vertex, calculate the reciprocal $1/\tilde{w}$ and multiply this by all elements of the coordinate array. The coordinate array at each vertex has the form $\begin{bmatrix}\tilde{x}/\tilde{w} & \tilde{y}/\tilde{w} & \tilde{z}/\tilde{w} & 1 & \tilde{s}/\tilde{w} & \tilde{t}/\tilde{w} & \tilde{r}/\tilde{w} & \tilde{q}/\tilde{w}\end{bmatrix}$. Rename the variables in this array to $\begin{bmatrix}x & y & z & 1 & s & t & r & q\end{bmatrix}$.

4. For each polygon, calculate some partial derivatives in screen space: $\partial z/\partial x$, $\partial s/\partial x$, $\partial t/\partial x$, $\partial q/\partial x$, $\partial s/\partial y$, $\partial t/\partial y$, and $\partial q/\partial y$. These are constant across a planar polygon. Section 3 suggests methods of calculation.

5. At each vertex, calculate the numerators of the partial derivatives of the texture coordinates in texture space with respect to the spatial coordinates in screen space.

$$a = \left(q\frac{\partial s}{\partial x} - s\frac{\partial q}{\partial x}\right) \qquad b = \left(q\frac{\partial t}{\partial x} - t\frac{\partial q}{\partial x}\right)$$

$$g = \left(q\frac{\partial s}{\partial y} - s\frac{\partial q}{\partial y}\right) \qquad d = \left(q\frac{\partial t}{\partial y} - t\frac{\partial q}{\partial y}\right)$$

Augment the coordinate array at each vertex with the four new variables to obtain the extended coordinate array $\begin{bmatrix}x & y & z & 1 & s & t & r & q & a & b & g & d\end{bmatrix}$.

6. For each polygon, calculate the scan-line increments for two of these four variables:

$$\Delta g = \Delta x\left(\frac{\partial q}{\partial x}\frac{\partial s}{\partial y} - \frac{\partial s}{\partial x}\frac{\partial q}{\partial y}\right) \qquad \Delta d = \Delta x\left(\frac{\partial q}{\partial x}\frac{\partial t}{\partial y} - \frac{\partial t}{\partial x}\frac{\partial q}{\partial y}\right)$$

where the spacing between pixels Δx is usually 1 in screen space. These are constant across any scan line. Note that $\Delta a = 0$ and $\Delta b = 0$.

7. For each polygon, linearly interpolate all the coordinates in the extended array down polygon edges. For convex planar polygons, interpolation needs to be performed only at the leading edges. For example, the leading edges would be the left-most edges when scan conversion proceeds left to right.

8. Let the variables at the beginning of the span have the subscript 0: s_0, t_0, q_0, a_0, b_0, g_0, and d_0. For each span, linearly interpolate the required variables across the span. Note that a and b are constant across the span so they need not be interpolated.

$$z_{i+1} = z_i + \Delta x \frac{\partial z}{\partial x}$$

$$s_{i+1} = s_i + \Delta x \frac{\partial s}{\partial x}$$

$$t_{i+1} = t_i + \Delta x \frac{\partial t}{\partial x} \qquad g_{i+1} = g_i + \Delta g$$

$$q_{i+1} = q_i + \Delta x \frac{\partial q}{\partial x} \qquad d_{i+1} = d_i + \Delta d$$

9. For each pixel, perform the *z* buffer test, and on success, project the warped texture coordinates back to texture space. To facilitate the projection, calculate the reciprocal of the homogeneous texture coordinate q to obtain $1/q$. Multiply the reciprocal by the texture coordinates in screen space to obtain the texture coordinates in texture space:

$$\hat{s} = \frac{s}{q} \quad \text{and} \quad \hat{t} = \frac{t}{q}.$$

Recall that the interpolated variables include the numerators of the partial derivatives of the texture coordinates in texture space with respect to the spatial coordinates in screen space:

$$a = q^2 \frac{\partial \hat{s}}{\partial x}, \qquad b = q^2 \frac{\partial \hat{t}}{\partial x}, \qquad g = q^2 \frac{\partial \hat{s}}{\partial y}, \quad \text{and} \quad d = q^2 \frac{\partial \hat{t}}{\partial y}.$$

To obtain the partial derivatives, we could square the reciprocal $1/q$ and multiply the result $1/q^2$ by the four variables:

$$\frac{\partial \hat{s}}{\partial x} = \frac{a}{q^2}, \qquad \frac{\partial \hat{t}}{\partial x} = \frac{b}{q^2}, \qquad \frac{\partial \hat{s}}{\partial y} = \frac{g}{q^2}, \quad \text{and} \quad \frac{\partial \hat{t}}{\partial y} = \frac{d}{q^2}.$$

However, this would require one multiplication per partial derivative. Since q^2 is a common multiple on all the numerators, we can save multiplications by deferring multiplication of $1/q^2$.

10. For each pixel, map the texels around $\begin{bmatrix}\hat{s} & \hat{t}\end{bmatrix}$ to the pixel with an anti-aliasing algorithm. This usually requires calculation of the vector norms of the Jacobian basis vectors:

$$\left\| \begin{bmatrix} \frac{\partial \hat{s}}{\partial x} & \frac{\partial \hat{t}}{\partial x} \end{bmatrix} \right\| = \left\| \begin{bmatrix} \frac{a}{q^2} & \frac{b}{q^2} \end{bmatrix} \right\| = \frac{1}{q^2} \left\| \begin{bmatrix} a & b \end{bmatrix} \right\| \quad \text{and}$$

$$\left\| \begin{bmatrix} \frac{\partial \hat{s}}{\partial y} & \frac{\partial \hat{t}}{\partial y} \end{bmatrix} \right\| = \left\| \begin{bmatrix} \frac{g}{q^2} & \frac{d}{q^2} \end{bmatrix} \right\| = \frac{1}{q^2} \left\| \begin{bmatrix} g & d \end{bmatrix} \right\|.$$

In the first Jacobian basis vector, $\begin{bmatrix}a & b\end{bmatrix}$ is invariant across the scan line so the direction of the vector is constant and $\left\|\begin{bmatrix}a & b\end{bmatrix}\right\|$ needs to be calculated only once per span. In addition, note that deferring multiplication of $1/q^2$ to this step saves two multiplications per pixel.

4.2 Remarks

The general algorithm makes no approximations. It is applicable to both space variant and invariant filtering methods including bilinear and trilinear [8], EWA [9, 5, 10], and footprint assembly [9] filtering. The anisotropic methods require the directions and magnitudes of the Jacobian basis vectors. The isotropic methods require only the magnitudes.

An approximation for hypotenuse is valuable because hypotenuse is a slow operation, and in this case, the approximation's error is always bounded. Paeth describes a fast linear approximation of hypotenuse with less than 12% error [15]. A modified approximation with less than 6% error is as follows:

$$\left\|\begin{bmatrix}a & b\end{bmatrix}\right\| = \sqrt{a^2 + b^2} \approx \max\left(|a|, |b|\right) + \frac{11}{32}\min\left(|a|, |b|\right)$$

In this modification, we distribute the error so that the approximation is sometimes larger than the actual value and sometimes smaller. The maximum occurs when the two operands are equal in magnitude. This approximation provides adequate accuracy for anti-aliasing algorithms.

5. Conclusions

Interpolation of texture coordinates and their partial derivatives for planar polygons viewed with perspective requires a total of one division and one multiplication per pixel plus one multiplication per texture coordinate and half of one multiplication per partial derivative. Updates of the variables are linear with at most one addition per component.

While the algorithm in this paper is simple to implement with parallelism in graphics hardware, even less effort may be possible with similar anti-aliasing quality. Future efforts could focus on simplifying the interpolation of partial derivatives. For example, bilinear MIP mapping requires only determination of the MIP map level. By calculating the levels at the ends of the span, a level variable could be interpolated instead of the partial derivatives. Exploiting the coherence of anti-aliasing parameters across a span appears promising.

6. Acknowledgements

I would like to thank Jon Ashburn in the Graphics Technology Lab of Hewlett-Packard, Fort Collins, Colorado, for reviewing this paper and for his helpful comments, which led to improvements in the general algorithm.

1. References

[1] Paul S. Heckbert and Henry P. Moreton, Interpolation for Polygon Texture Mapping and Shading, in *State of the Art in Computer Graphics: Visualization and Modeling*, David F. Rogers and Rae A. Earnshaw, eds., Springer-Verlag, 1991, pp. 101–111.

[2] James F. Blinn, Hyperbolic Interpolation, in *IEEE Computer Graphics and Applications*, 12(4):89–94, July 1992.

[3] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli, Fast Shadows and Lighting Effects Using Texture Mapping, in *Proceedings of SIGGRAPH '92, Computer Graphics*, 26(2):249–252, July 1992.

[4] Harvey P. Greenspan and David J. Benny, *Calculus: An Introduction to Applied Mathematics*, McGraw-Hill, 1973.

[5] Paul S. Heckbert, *Fundamentals of Texture Mapping and Image Warping*, Master's thesis, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, California, June 1989.

[6] Lance Williams, Pyramidal Parametrics, in *Proceedings of SIGGRAPH '83, Computer Graphics*, 17(3):1–11, July 1983.

[7] Paul S. Heckbert, *Texture Mapping Polygons in Perspective*, NYIT Computer Graphics Lab, Technical Memo 13, April 1983.

[8] Alan Watt and Mark Watt, *Advanced Animation and Rendering Techniques: Theory and Practice*, ACM Press, Addison-Wesley, 1992.

[9] Ned Greene and Paul S. Heckbert, Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter, in *IEEE Computer Graphics and Applications*, 6(6):21–27.

[10] Robert C. Lansdale, *Texture Mapping and Resampling for Computer Graphics*, Master's thesis, Dept. of Electrical Engineering, University of Toronto, Toronto, Ontario, Canada, January 1991.

[11] Andreas Schilling, Günter Knittel, and Wolfgang Strasser, Texram: A Smart Memory for Texturing, in *IEEE Computer Graphics and Applications*, 16(3):32–41, May 1996.

[12] OpenGL Architectural Review Board, Jackie Neider, Tom Davis, and Mason Woo, *OpenGL Programming Guide*, Addison-Wesley, 1993.

[13] OpenGL Architectural Review Board, *OpenGL Reference Manual*, Addison-Wesley, 1992.

[14] Mark Segal and Kurt Akeley, *The OpenGL Graphics System: A Specification*, Chris Frasier, ed., Edition 1.1, Silicon Graphics, Inc., June 1996.

[15] Alan W. Paeth, A Fast Approximation to the Hypotenuse, in *Graphics Gems*, Andrew S. Glassner, ed., Academic Press, pp. 427–431, 1990.