Perspective Mappings Between Quadrilaterals

David Eberly
Magic Software, Inc.
http://www.magic-software.com

Created: March 2, 1999

A standard problem in computer graphics is to take a texture stored as a rectangular bitmap and apply it to a rectangular face of a polyhedron which has been rendered using perspective projection. The texture itself needs to conform to the perspective projection. Since the polyhedron face is a convex quadrilateral in the viewing plane, the problem reduces to mapping points from the texture rectangle to the quadrilateral. More generally, we show how to map points between any two convex quadrilaterals using perspective projection. Let the planar vertices of the source quadrilateral be labeled in counterclockwise order as $\vec{p}_{00}$, $\vec{p}_{10}$, $\vec{p}_{11}$, and $\vec{p}_{01}$. Similarly let the planar vertices of the target quadrilateral be labeled as $\vec{q}_{00}$, $\vec{q}_{10}$, $\vec{q}_{11}$, and $\vec{q}_{01}$.

Consider the problem as a perspective projection from three dimensions to two dimensions. Assume the viewing plane is $z = 0$ and the eye point is $\vec{E} = (e_0, e_1, e_2)$ where $e_2 \neq 0$. The perspective projection of a point $\vec{r} = (x, y, z)$ to the viewing plane is the point $\vec{r}_0 = (x_0, y_0, 0)$ which lies on the ray emanating from $\vec{E}$ and containing $\vec{r}$. The linear relationship is $\vec{r}_0 = (1-t)\vec{E}+t\vec{r}$ where $t = e_2/(e_2-z)$, $x_0 = (e_2x-e_0z)/(e_2-z)$, and $y_0 = (e_2y - e_1z)/(e_2 - z)$.

Embed the quadrilateral vertices in the viewing plane as $\vec{P}_{ij} = (\vec{p}_{ij} - \vec{p}_{00}, 0)$ and $\vec{Q}_{ij} = (\vec{q}_{ij} - \vec{q}_{00}, 0)$. Rotate the vertices $\vec{Q}_{ij}$ so that they lie in a plane defined by $\vec{N} \cdot (x, y, z) = 0$. Let the rotated points be denoted $\vec{R}_{ij}$, so $\vec{N} \cdot \vec{R}_{ij} = 0$ for all i and j. The problem is to select $\vec{E}$ and $\vec{N}$ so that each pair $\vec{R}_{ij}$ and $\vec{P}_{ij}$ lie on the same ray. Once selected, the general transformation between $\vec{R}$ and $\vec{P}$ is known.

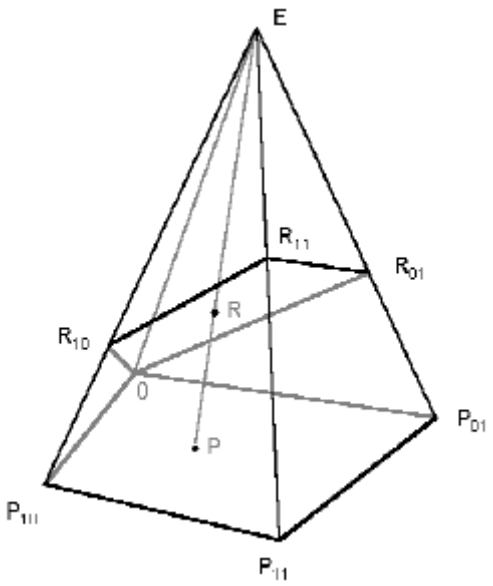

Figure 1: Projection of one quadrilateral onto another.

Applying the inverse rotation to $\vec{R}$ to obtain $\vec{Q}$ and extracting the first two components yields the perspective projection from $\vec{p}$ in the source quadrilateral to $\vec{q}$ in the target quadrilateral. The mathematical details are given below.

Perspective projection requires that $\vec{R}_{ij} = (1-t_{ij})\vec{E}+t_{ij}\vec{P}_{ij}$ for some t_{ij}. Since the points lie on the specified plane, $0 = \vec{N}\cdot\vec{R}_{ij} = (\vec{N}\cdot\vec{E}) + t_{ij}\vec{N}\cdot(\vec{P}_{ij}-\vec{E})$. Solving this equation for t_{ij} and replacing in the projection equation yields

$$\vec{R}_{ij} = \frac{(\vec{N}\cdot\vec{E})\vec{P}_{ij} - (\vec{N}\cdot\vec{P}_{ij})}{\vec{N}\cdot(\vec{E}-\vec{P}_{ij})}.$$

Since $\vec{Q}_{10}$ and $\vec{Q}_{01}$ are linearly independent vectors, we can write $\vec{Q}_{11} = \alpha\vec{Q}_{10} + \beta\vec{Q}_{01}$. Convexity of the quadrilateral implies $\alpha > 0$, $\beta > 0$, and $\alpha + \beta > 1$. The same relationship holds for the rotated points $\vec{R}_{ij}$. Similarly, $\vec{P}_{11} = \gamma\vec{P}_{10} + \delta\vec{P}_{01}$ where $\gamma > 0$, $\delta > 0$, and $\gamma + \delta > 1$. Thus,

$$\begin{aligned}
\alpha\vec{R}_{10} + \beta\vec{R}_{01} &= \vec{R}_{11} \\
&= \frac{(\vec{N}\cdot\vec{E})\vec{P}_{11} - (\vec{N}\cdot\vec{P}_{11})\vec{E}}{\vec{N}\cdot(\vec{E}-\vec{P}_{11})} \\
&= \frac{(\vec{N}\cdot\vec{E})(\gamma\vec{P}_{10}+\delta\vec{P}_{01}\vec{E}) - (\vec{N}\cdot(\gamma\vec{P}_{10}+\delta\vec{P}_{01}))}{\vec{N}\cdot(\vec{E}-\gamma\vec{P}_{10}-\delta\vec{P}_{01})} \\
&= \frac{\gamma[(\vec{N}\cdot\vec{E})\vec{P}_{10} - (\vec{N}\cdot\vec{P}_{10})\vec{E}] + \delta[(\vec{N}\cdot\vec{E})\vec{P}_{01} - (\vec{N}\cdot\vec{P}_{01})\vec{E}]}{\vec{N}\cdot(\vec{E}-\gamma\vec{P}_{10}-\delta\vec{P}_{01})} \\
&= \frac{\gamma\vec{N}\cdot(\vec{E}-\vec{P}_{10})}{\vec{N}\cdot(\vec{E}-\gamma\vec{P}_{10}-\delta\vec{P}_{01})}\vec{R}_{10} + \frac{\delta\vec{N}\cdot(\vec{E}-\vec{P}_{01})}{\vec{N}\cdot(\vec{E}-\gamma\vec{P}_{10}-\delta\vec{P}_{01})}\vec{R}_{01}.
\end{aligned}$$

By linear independence of $\vec{R}_{10}$ and $\vec{R}_{01}$ it follows that

$$\alpha = \frac{\gamma\vec{N}\cdot(\vec{E}-\vec{P}_{10})}{\vec{N}\cdot(\vec{E}-\gamma\vec{P}_{10}-\delta\vec{P}_{01})} \text{ and } \beta = \frac{\delta\vec{N}\cdot(\vec{E}-\vec{P}_{01})}{\vec{N}\cdot(\vec{E}-\gamma\vec{P}_{10}-\delta\vec{P}_{01})}. \quad (1)$$

Rearranging terms yields

$$\vec{N} \cdot [\alpha(\vec{E} - \gamma\vec{P}_{10} - \delta\vec{P}_{01}) - \gamma(\vec{E} - \vec{P}_{10})] = 0,$$
$$\vec{N} \cdot [\beta(\vec{E} - \gamma\vec{P}_{10} - \delta\vec{P}_{01}) - \delta(\vec{E} - \vec{P}_{01})] = 0$$

which implies $\vec{N}$ is orthogonal to the two listed vectors. The cross products of the two vectors may be used for $\vec{N}$.

$$\vec{N} = [\alpha(\vec{E} - \gamma\vec{P}_{10} - \delta\vec{P}_{01}) - \gamma(\vec{E} - \vec{P}_{10})] \times [\beta(\vec{E} - \gamma\vec{P}_{10} - \delta\vec{P}_{01}) - \delta(\vec{E} - \vec{P}_{01})].$$

The perspective transformations involve only dot products of the normal with various vectors. Some algebraic manipulations yield

$$\begin{aligned}
\vec{N} \cdot \vec{P}_{10} &= \delta(\gamma - \alpha + \alpha\delta - \beta\gamma)\Delta e_2, \\
\vec{N} \cdot \vec{P}_{01} &= \delta(\delta - \beta + \beta\gamma - \alpha\delta)\Delta e_2, \\
\vec{N} \cdot \vec{E} &= \gamma\delta(1 - \alpha - \beta)\Delta e_2, \\
\vec{N} \cdot (\vec{E} - \vec{P}_{10}) &= \alpha\delta(1 - \gamma - \delta)\Delta e_2, \\
\vec{N} \cdot (\vec{E} - \vec{P}_{01}) &= \beta\gamma(1 - \gamma - \delta)\Delta e_2,
\end{aligned}$$

where $\Delta \neq 0$ is defined by $\vec{P}_{10} \times \vec{P}_{01} = (0, 0, \Delta)$.

Finally, we construct the mapping. Let $\vec{R} = u\vec{R}_{10} + v\vec{R}_{01}$ be a point in the rotated quadrilateral which is projected onto a point $\vec{P} = x\vec{P}_{10} + y\vec{P}_{01}$ in the view plane quadrilateral. The conditions for containment in the quadrilaterals are obtained by the point–in–polygon tests: $u \geq 0$, $v \geq 0$, $(1-\beta)u + \alpha(v-1) \leq 0$, $\beta(u-1)+(1-\alpha)v \leq 0$ and $x \geq 0$, $y \geq 0$, $(1-\delta)x+\gamma(y-1) \leq 0$, $\delta(x-1)+(1-\gamma)y \leq 0$. The construction that led to equation (1) may be repeated, but with α, β, γ, δ, and $\vec{R}_{11}$ replace by u, v, x, y, and $\vec{R}$, respectively, to obtain

$$u(x,y) = \frac{\alpha\delta(1-\gamma-\delta)x}{\gamma\delta(1-\alpha-\beta) + \delta(\alpha-\gamma+\beta\gamma-\alpha\delta)x + \gamma(\beta-\delta-\beta\gamma+\alpha\delta)y}$$

and

$$v(x,y) = \frac{\beta\gamma(1-\gamma-\delta)y}{\gamma\delta(1-\alpha-\beta) + \delta(\alpha-\gamma+\beta\gamma-\alpha\delta)x + \gamma(\beta-\delta-\beta\gamma+\alpha\delta)y}.$$

Rotating the plane containing the $\vec{R}_{ij}$ back to $z = 0$ and translating to the original coordinates of the quadrilaterals yields the perspective transformation

$$\vec{q} = [1 - u(x,y) - v(x,y)]\vec{q}_{00} + u(x,y)\vec{q}_{10} + v(x,y)\vec{q}_{01}$$

where $\vec{p} = (1-x-y)\vec{p}_{00} + x\vec{p}_{10} + y\vec{p}_{01}$ with $x \geq 0$, $y \geq 0$, $(1-\delta)x+\gamma(y-1) \leq 0$, and $\delta(x-1)+(1-\gamma)y \leq 0$. Note that both $\vec{p}$ and $\vec{q}$ are barycentric combinations of vertices of triangles.

Intuitively, perspective transformations map lines to lines. They also map conic sections to conic sections. The proof is straightforward. Let the perspective transformation be $u = (a_0 + a_1x + a_2y)/(c_0 + c_1x + c_2y)$ and $v = (b_0 + b_1x + b_2y)/(c_0 + c_1x + c_2y)$. The inverse transformation is of the same form, so it suffices to show a conic section in uv coordinates satisfying $Au^2 + Buv + Cv^2 + Du + Ev + F = 0$ is mapped to a conic section in xy coordinates satisfying $\bar{A}x^2 + \bar{B}xy + \bar{C}y^2 + \bar{D}x + \bar{E}y + \bar{F} = 0$. Substituting the formulas for u and v into the quadratic equation, multiplying by $(c_0 + c_1x + c_2y)^2$, expanding the products, and grouping the appropriate terms yields a quadratic in x and y. The left image in Figure 2 shows a square containing several conic sections. The top curve is a parabola, the left and right curves are hyperbolas, the center curve is a circle, and the bottom curve is an ellipse. The grid consists of straight lines. The right image in Figure 2 shows a perspective mapping of these conic sections.

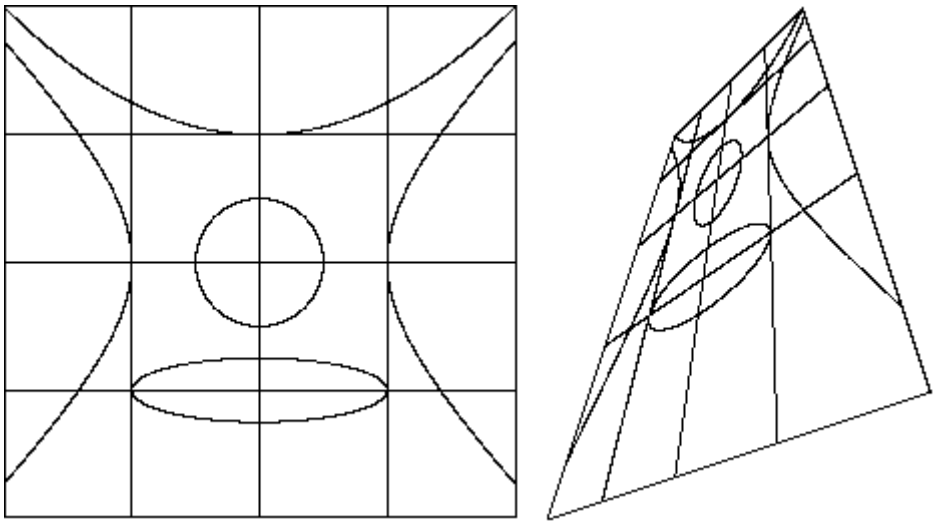

Figure 2: Projection of conic sections in a square to conic sections in a quadrilateral.

The implementation of the perspective mapping has a slight modification from what was described earlier. Solving $\vec{q} - \vec{q}_{00} = \alpha(\vec{q}_{10} - \vec{q}_{00}) + \beta(\vec{q}_{01} - \vec{q}_{00})$ requires setting up the matrix system

$$\begin{bmatrix} (\vec{q}_{10} - \vec{q}_{00}) \cdot (\vec{q}_{10} - \vec{q}_{00}) & (\vec{q}_{10} - \vec{q}_{00}) \cdot (\vec{q}_{01} - \vec{q}_{00}) \\ (\vec{q}_{01} - \vec{q}_{00}) \cdot (\vec{q}_{10} - \vec{q}_{00}) & (\vec{q}_{01} - \vec{q}_{00}) \cdot (\vec{q}_{01} - \vec{q}_{00}) \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} (\vec{q}_{10} - \vec{q}_{00}) \cdot (\vec{q} - \vec{q}_{00}) \\ (\vec{q}_{01} - \vec{q}_{00}) \cdot (\vec{q} - \vec{q}_{00}) \end{bmatrix}$$

where the coefficient matrix M is symmetric. Numerical problems may arise in solving this system by inverting M. The first problem to deal with is the magnitude of the vertex values. These values can be large, especially if they are given in terms of pixel coordinates on a raster of large size. The perspective transformation is invariant to uniform magnification of the quadrilaterals. The code normalizes each quadrilateral by scaling using the lengths of $\vec{p}_{11} - \vec{p}_{00}$ and $\vec{q}_{11} - \vec{q}_{00}$. If all of the vertices are extremely small in magnitude, then the normalization itself may have problems. In this case the calling routine is notified (via a returned boolean flag) that the quadrilateral is too small.

The second problem to deal with is the magnitude of $\det(M)$. The determinant must be positive since $\vec{q}_{10} - \vec{q}_{00}$ and $\vec{q}_{01} - \vec{q}_{00}$ are linearly independent. However, for needlelike quadrilaterals, the determinant can be nearly zero and will cause numerical problems if the system is solved directly. Rather than inverting M, we multiply by the adjoint matrix and avoid the division by $\det(M)$. The problem must be dealt with for each matrix system. Define Δ_p to be the determinant of the matrix for the source quadrilateral and define Δ_q to be the determinant of the matrix for the target quadrilateral. Finally, define $\alpha' = \Delta_q\alpha$, $\beta' = \Delta_q\beta$, $\gamma' = \Delta_p\gamma$, and $\delta' = \Delta_p\delta$. The perspective mapping is

$$u(x,y) = \frac{\alpha'\delta'(\Delta_p - \gamma' - \delta')x}{\gamma'\delta'(\Delta_q - \alpha' - \beta') + \delta'(\Delta_p\alpha' - \Delta_q\gamma' + \beta'\gamma' - \alpha'\delta')x + \gamma'(\Delta_p\beta' - \Delta_q\delta' - \beta'\gamma' + \alpha'\delta')y}$$

and

$$v(x,y) = \frac{\beta'\gamma'(\Delta_p - \gamma' - \delta')y}{\gamma'\delta'(\Delta_q - \alpha' - \beta') + \delta'(\Delta_p\alpha' - \Delta_q\gamma' + \beta'\gamma' - \alpha'\delta')x + \gamma'(\Delta_p\beta' - \Delta_q\delta' - \beta'\gamma' + \alpha'\delta')y}.$$

Note that this requires a slight modification of the point–in–quadrilateral conditions. For source point $\vec{p}$, the tests become $x \geq 0$, $y \geq 0$, $(\Delta_p - \delta)x + \gamma(y-1) \leq 0$, and $\delta(x-1) + (\Delta_p - \gamma)y \leq 0$.

The code is given below:

```
/* points in the plane */
typedef struct { double x, y; } Point2;

/* barycentric coordinates for planar points */
typedef struct { double b00, b10, b01; } Bary2;

/* convex quadrilateral, points in counterclockwise order */
typedef struct { Point2 v00, v10, v11, v01; } Quad;

/* map, u = a*x/(c00+c10*x+c01*y), v = b*y/(c00+c10*x+c01*y) */
typedef struct
{
    double pDet, qDet;
    double alpha, beta, gamma, delta;
    double a, b, c00, c10, c01;
}
PerspectiveMap;
```

```
int QuadToQuad (Quad p, Quad q, PerspectiveMap* map)
{
    static double tolerance = 1e-05;  /* user-selectable */

    Point2 e10, e01, e11;
    double length, m00, m01, m11, r0, r1;
    PerspectiveMap map;

    /* assert:  src and trg are convex quads, counterclockwise order */

    /* compute the sides and diagonal of the source quadrilateral */
    e10.x = p.v10.x-p.v00.x;
    e10.y = p.v10.y-p.v00.y;
    e01.x = p.v01.x-p.v00.x;
    e01.y = p.v01.y-p.v00.y;
    e11.x = p.v11.x-p.v00.x;
    e11.y = p.v11.y-p.v00.y;

    /* normalize the source quadrilateral */
    length = sqrt(e11.x*e11.x+e11.y*e11.y);
    if ( length < tolerance )
        return 0;
    e11.x /= length;
    e11.y /= length;
    e10.x /= length;
    e10.y /= length;
    e01.x /= length;
    e01.y /= length;

    /* solve for gamma and delta of the source quadrilateral */
    m00 = e10.x*e10.x+e10.y*e10.y;
    m01 = e10.x*e01.x+e10.y*e01.y;
    m11 = e01.x*e01.x+e01.y*e01.y;
    r0 = e10.x*e11.x+e10.y*e11.y;
    r1 = e01.x*e11.x+e01.y*e11.y;
    map->pDet = m00*m11-m01*m01;
        /* assert: map.pDet > 0 for convex quad */
    map->gamma = m11*r0-m01*r1;
    map->delta = m00*r1-m01*r0;

    /* compute the sides and diagonal of the target quadrilateral */
    e10.x = q.v10.x-q.v00.x;
    e10.y = q.v10.y-q.v00.y;
    e01.x = q.v01.x-q.v00.x;
    e01.y = q.v01.y-q.v00.y;
    e11.x = q.v11.x-q.v00.x;
    e11.y = q.v11.y-q.v00.y;
```

```
    /* normalize the target quadrilateral */
    length = sqrt(e11.x*e11.x+e11.y*e11.y);
    if ( length < tolerance )
        return 0;
    e11.x /= length;
    e11.y /= length;
    e10.x /= length;
    e10.y /= length;
    e01.x /= length;
    e01.y /= length;

    /* solve for alpha and beta of the target quadrilateral */
    m00 = e10.x*e10.x+e10.y*e10.y;
    m01 = e10.x*e01.x+e10.y*e01.y;
    m11 = e01.x*e01.x+e01.y*e01.y;
    r0 = e10.x*e11.x+e10.y*e11.y;
    r1 = e01.x*e11.x+e01.y*e11.y;
    map->qDet = m00*m11-m01*m01;
        /* assert: map.qDet > 0 for convex quad */
    map->alpha = m11*r0-m01*r1;
    map->beta  = m00*r1-m01*r0;

    /* transformation which maps src points to trg points */
    map->a = map->alpha*map->delta*(map->pDet-map->gamma-map->delta);
    map->b = map->beta*map->gamma*(map->pDet-map->gamma-map->delta);
    map->c00 = map->gamma*map->delta*(map->qDet-map->alpha-map->beta);
    map->c10 = map->delta*(map->pDet*map->alpha-map->qDet*map->gamma
              +map->beta*map->gamma-map->alpha*map->delta);
    map->c01 = map->gamma*(map->pDet*map->beta-map->qDet*map->delta
              -map->beta*map->gamma+map->alpha*map->delta);

    return 1;
}

Bary2 Evaluate (PerspectiveMap* map, Bary2 src)
{
    /* assert:  input point is in source quadrilateral
       src.b10 >= 0,
       src.b01 >= 0,
       (map->pDet-map->delta)*src.b10+map->gamma*(src.b01-1) <= 0,
       map->delta*(src.b10-1)+(map->pDet-map->gamma)*src.b01 <= 0  */

    Bary2 trg;
    double denom;

    denom = map->c00+src.b10*map->c10+src.b01*map->c01;
    trg.b10 = src.b10*map->a/denom;
    trg.b01 = src.b01*map->b/denom;
```

```
    trg.b00 = 1.0-trg.b10-trg.b01;

    /* assert:  output point is in target quadrilateral
       trg.b10 >= 0,
       trg.b01 >= 0,
       (map->qDet-map->beta)*trg.b10+map->alpha*(trg.b01-1) <= 0,
       map->beta*(trg.b10-1)+(map->qDet-map->alpha)*trg.b01 <= 0  */

    return trg;
}
```

The following application shows how text is mapped to a quadrilateral to give the impression of scrolling from near to far (as is done in a number of science fiction films). For applications that require faster texture mapping (but lower quality output), see Section 14.4 for a scan line algorithm which uses spatial coherence, the property that lines are mapped to lines, and linear interpolation to reduce the number of lookups in the texture map. Figure 3 shows the original text stored in a rectangular grid. This rectangle is the target quadrilateral of the process.

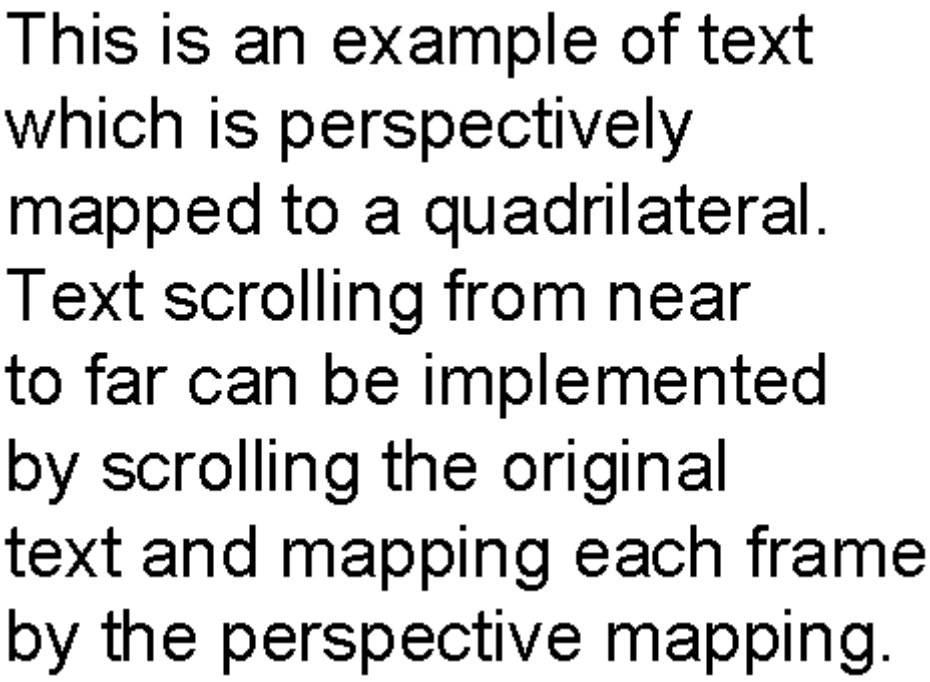

Figure 3: Original text in a texture map.

Figure 4 shows the mapping to a quadrilateral. For each pixel in the source quadrilateral, the perspective map is applied, the target point is truncated to the nearest pixel coordinates, and the texture is looked up in the target rectangle. Notice the aliasing effects caused by the discretization.

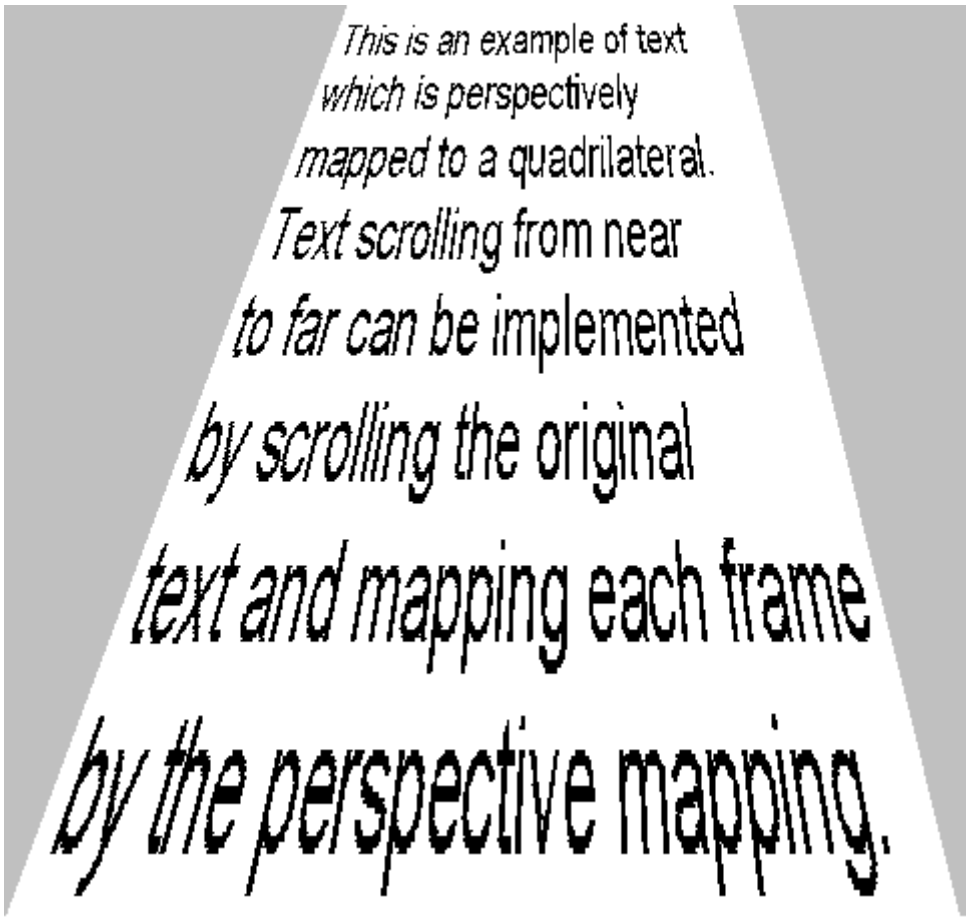

Figure 4: Perspectively mapped text with no anti–aliasing.

Figure 5 shows the same mapping, but instead of truncating the target point to nearest pixel, a bilinear interpolation is applied to provide anti–aliasing.

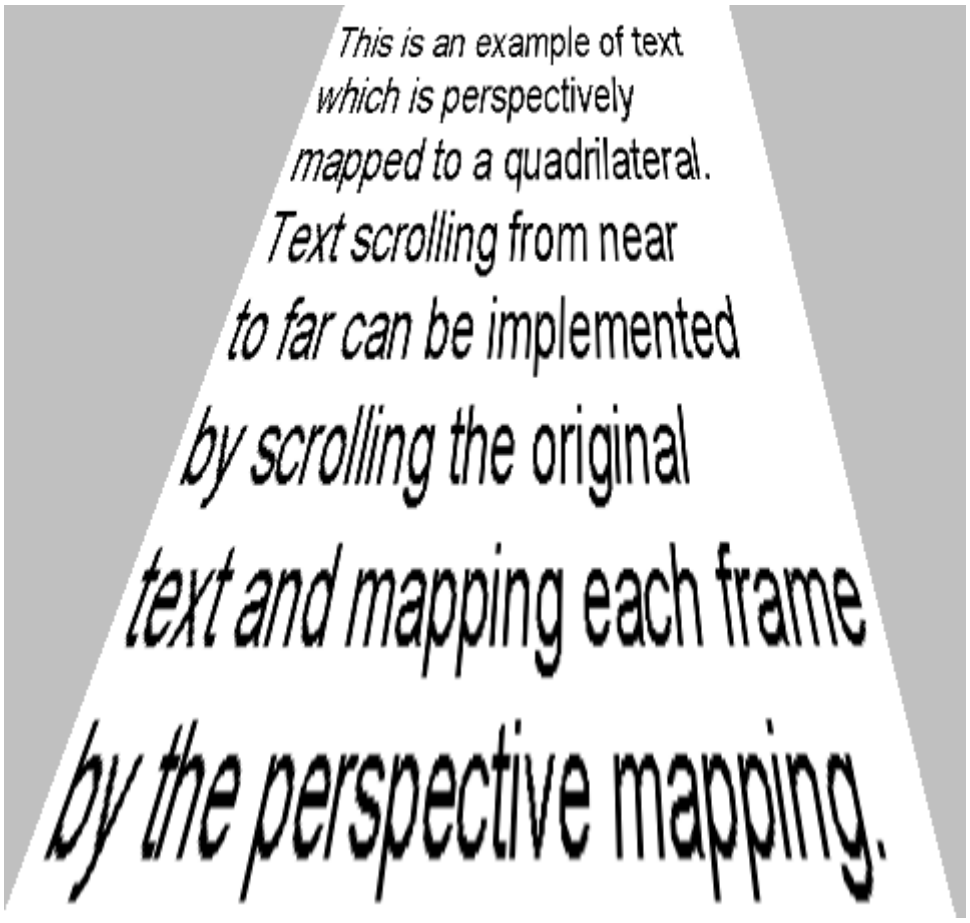

Figure 5: Perspectively mapped text with anti–aliasing.

The code which produces these images is given below. The images are gray–scale, but the same ideas can be applied to color images by applying the lookup/interpolation in each of the separate channels in whatever color space the user desires.

```
typedef struct
{
    int R;  /* number of rows in the image */
    int C;  /* number of columns in the image */
    double** pixel;
        /* Gray-scale values, stored as pixel[y][x] where y is row
           number and x is column number.  Image coordinates are
           left-handed. */
}
Image;

void ScrolledText (Image* text, Image* scroll)
{
```

```
/* assert: memory has been allocated for both images, the
   texture map has been initialized. */

Quad sq, tq;
PerspectiveMap map;
int x, y;
int yLeft;  /* conversion between left-hand image coordinates and
               right-hand quadrilateral coords */

/* define the source quad (the scrolled text) */
sq.v00.x = 0.0;
sq.v00.y = 0.0;
sq.v10.x = scroll.C-1;
sq.v10.y = 0.0;
sq.v01.x = 3*scroll.C/8;
sq.v01.y = scroll.R-1;
sq.v11.x = 3*scroll.C/4;
sq.v11.y = scroll.R-1;

/* define the target quad (the original text) */
tq.v00.x = 0.0;
tq.v00.y = 0.0;
tq.v10.x = text.C-1;
tq.v10.y = 0.0;
tq.v01.x = 0.0;
tq.v01.y = text.R-1;
tq.v11.x = text.C-1;
tq.v11.y = text.R-1;

/* construct the perspective map */
QuadToQuad(sq,tq,&map);

/* process each scan line of the output image */
for (y = 0, yLeft = scroll.R-1; y < scroll.R; y++, yLeft--)
{
    int x0, x1;
    double xdif, ydif, u, v;

    /* find left edge of quad */
    x0 = 0;
    while ( -x0*sq.v01.y+y*sq.v01.x > 0 )
        x0++;

    /* find right edge of quad */
    xdif = sq.v10.x-sq.v11.x;
    ydif = sq.v11.y-sq.v10.y;
    x1 = scroll.C-1;
    while ( (x1-sq.v10.x)*ydif+y*xdif > 0 )
```

```
            x1--;

        /* process each pixel in scan line of quad */
        for (x = x0; x <= x1; x++)
        {
            Bary2 src, trg;

            /* source pixel in barycentric coordinates */
            src.b01 = y/(scroll.R-1.0);
            src.b10 = (x-sq.v01.x*src.b01)/(scroll.C-1.0);
            src.b00 = 1.0-src.b10-src.b01;

            Evaluate(map,src,trg);

            /* target pixel in barycentric coordinates */
            u = trg.b00*tq.v00.x+trg.b10*tq.v10.x+trg.b01*tq.v01.x;
            v = trg.b00*tq.v00.y+trg.b10*tq.v10.y+trg.b01*tq.v01.y;

            /* clip to text rectangle, allow room for bi-interp */
            if ( 1 <= u && u < text.C-1 && 1 <= v && v < text.R-1 )
            {
                int u0, v0;
                double t00, t10, t01, t11, du, dv, omdu, omdv;

                /* truncate to nearest pixel */
                u0 = int(floor(u));
                v0 = int(floor(v));

                /* direct lookup shows aliasing effects
                scroll.pixel[yLeft][x] = texture[v0][u0];
                */

                /* anti-aliasing via bilinear interpolation */
                t00 = texture[v0][u0];
                t10 = texture[v0][u0+1];
                t01 = texture[v0+1][u0];
                t11 = texture[v0+1][u0+1];
                du = u-u0;
                dv = v-v0;
                omdu = 1-du;
                omdv = 1-dv;
                scroll[yLeft][x] = omdu*omdv*t00+omdu*dv*t01
                    +du*omdv*t10+du*dv*t11;
            }
        }
    }
}
```