Accurate constant-z scanline perspective texture mapping

Keith H. Randall
Compaq Systems Research Center
130 Lytton Avenue, Palo Alto, CA 94301
randall@src.dec.com

March 17, 1999

Abstract

Modern perspective texture mappers scan-convert polygons into horizontal scanlines and then draw each horizontal scanline in turn. Because a horizontal scanline is not uniformly distant in the field of view, its texture coordinates are a nonlinear function of the screen coordinates. Computing this nonlinear function requires an expensive divide operation or a potentially inaccurate approximation such as quadratic or piecewise linear.

A different approach to texture mapping is to use ***constant-z scanlines***, scanlines that traverse a polygon at a fixed distance from the viewer, allowing their texture coordinates to be computed easily and efficiently. In general, such scanlines will not be horizontal but will instead be at some arbitrary angle on the screen. Angled scanlines can only be approximated on a real display, however, because pixels must be addressed by integral coordinates. The resulting approximation leads to an image that appears "wavy".

This paper provides a new algorithm for correcting the texture coordinates of the constant-z scanline to accomodate the fact that it cannot be drawn exactly. Our algorithm computes a first-order approximation to the deviation of the texture coordinates from the constant-z scanline and gives an efficient linear recurrence for computing this deviation incrementally. With this correction, constant-z scanlines produce accurate results and require no expensive operations (multiplies and divides) in the inner loop, making constant-z scanlines practical for accurate perspective texture mapping.

Keywords

texture mapping, algorithm, perspective, scanline, subpixel resolution

1 Introduction

Modern perspective texture mappers scan-convert polygons into horizontal scanlines and then draw each horizontal scanline to the screen [4]. Because a horizontal scanline is not uniformly distant in the field of view, its texture coordinates are not a linear function of the screen coordinates. As a result, perspective texture mappers that scan-convert polygons into horizontal scanlines must either compute this nonlinear function using a divide [7], which can be slow, or approximate the nonlinearity with quadratic or piecewise linear approximations [5], which can be inaccurate.

A different approach to texture mapping a polygon is to use scanlines that traverse the polygon at a fixed distance from the viewer, also called ***constant-z scanlines***. In general, such scanlines will not be horizontal but will instead be at some arbitrary angle on the screen. The constant-z nature of these angled scanlines allows their texture coordinates to be computed easily and efficiently. Constant-z scanlines also have the desirable property that many depth-dependent calculations can be lifted out of the inner loop. Such computations include parts of the algorithms for z-buffering [6], mip-mapping [9], and depth-cued effects like fog and haze [3]. Unfortunately, angled scanlines can only be approximated on a real display because pixels must be addressed by integral coordinates. The resulting approximation leads to an image that appears "wavy".

The utility of constant-z scanlines has largely been overshadowed by this inaccuracy. The prevailing current view on diagonal, constant-z scanline texture mapping is elegantly summarized by Chris Hecker [5]:

> Finally, and this one is the kiss of death as far as I'm concerned, it is totally impossible to achieve subpixel accuracy with a lines-of-constant-z texture mapper. To achieve subpixel accuracy we must always sample the texture from the pixel centers of the destination but our arbitrary line-of-constant-z doesn't hit the pixel center in the destination. However, you can't step off the line-of-constant-z to the pixel center or you'll need to divide to take the nonconstant-z step into account. Damned if you do, damned if you don't.

This paper provides a new algorithm for "stepping off" the constant-z scanline to provide a good approximation to sampling at the pixel centers without doing a divide. This algorithm makes constant-z scanlines practical for accurate perspective texture mapping. Our algorithm uses a first order approximation to the deviation of the texture coordinates from the constant-z scanline and gives an efficient method for computing this deviation incrementally. In particular, we show that the corrected texture coordinates can be computed using a simple linear recurrence using only additions. As a consequence, our constant-z scanline algorithm produces accurate texture mapping using a principled approximation and requires no expensive operations (multiplies and divides) in the inner loop.

The remainder of this paper is organized as follows. Section 2 gives an overview of texture mapping and the notational conventions used in the remainder of the paper. Section 3 overviews conventional horizontal scanline algorithms and shows why the expensive divide operation is required. Section 4 describes how the diagonal, constant-z scanline algorithm eliminates the expensive divide. This section also describes the error these scanlines introduce by not sampling at pixel centers. Section 5 gives our new algorithm for stepping off the scanline to fix this error. Section 6 further optimizes our algorithm to completely eliminate the expensive operations (multiplies and divides) from the inner loop. Finally, Section 7 presents some experimental results and possible extensions.

2 Preliminaries

We will maintain typical coordinate conventions, with $\langle x,y,z\rangle$ being the coordinates of a point in the world, relative to a viewer at the origin looking along the $+z$ axis with the $+x$ axis to his right and the $+y$ axis upward [2]. The screen is parameterized by $\langle h,v\rangle$, where h is the horizontal screen coordinate and v is the vertical screen coordinate. The projection we use from three-dimensional world space $\langle x,y,z\rangle$ to two-dimensional screen space $\langle h,v\rangle$ is a ***perspective projection*** [2], which computes h and v from the equations $h = Hx/z$ and $v = Vy/z$, where H and V are scaling factors that encode the angular extent of the screen. In this paper, we will assume for simplicity that $H = V = 1$, so we have

$$x = hz \text{ and } y = vz \,. \tag{1}$$

A polygon is represented by the equation of the plane that it resides in:

$$ax + by + cz = d \,, \tag{2}$$

where a, b, c, and d are constants, and by the coordinates of its vertices. We will not deal with the vertices of the polygon explicitly in this paper; instead we assume that the polygon is appropriately mapped to the screen using the perspective projection equations and clipped so that its extent in screen space is known.

Texture mapping is the process of drawing a polygon on the screen using a specified texture in a perspective-correct fashion. We define texture coordinates in the polygon using two texture coordinate vectors $R = \langle R_x, R_y, R_z\rangle$ and $S = \langle S_x, S_y, S_z\rangle$ that lie in the polygon plane. Texture coordinates $\langle r,s\rangle$ for any point on the polygon's plane are computed from the dot product of the point with these two vectors.

$$r = R_x x + R_y y + R_z z \tag{3}$$
$$s = S_x x + S_y y + S_z z \tag{4}$$

We will assume that a function *texture* is provided that maps a (potentially fractional) texture coordinate $\langle r,s\rangle$ to a color. A simple function of this kind would be

$$\mathit{texture}(r,s) = \mathit{bitmap}[\lfloor r\rfloor \;\mathit{mod}\; 32][\lfloor s\rfloor \;\mathit{mod}\; 32] \,,$$

where *bitmap* is a 32×32 bitmap of color values. However, we also allow for more complex *texture* functions that perform various kinds of antialiasing [4], such as a mip-map [9] or summed-area table [1]. Our constant-z scanline algorithm will work for any *texture* function that computes a texture color given texture coordinates $\langle r,s\rangle$.

Because the two texture coordinates r and s are symmetric with respect to all of our derivations, we only give formulas for computing r; the corresponding formulas for computing s can be derived simply by replacing all occurrences of R with S. When we wish to distinguish the two texture axes, we will use an r or s subscript.

Finally, we will be drawing to a screen with pixels arranged in row-major order, with N pixels per row.

3 Horizontal scanlines

This section describes the traditional horizontal-scanline texture-mapping algorithm. Each polygon is rendered by drawing the pixels of the polygon one row, or one scanline, at a time. In order to calculate the correct texture coordinates for each pixel, a divide, or some approximation to a divide, must be performed for each pixel. To see why, let us calculate the correct texture coordinates for a pixel. We begin by computing the z value of the pixel at $\langle h,v\rangle$ by substituting Equations (1) into Equation (2):

$$\begin{aligned} ahz + bvz + cz &= d \\ z &= \frac{d}{ah + bv + c} \end{aligned} \tag{5}$$

Then, we compute the texture coordinate r using Equation (3):

$$\begin{aligned} r &= R_x hz + R_y vz + R_z z \\ &= (R_x h + R_y v + R_z)z \\ &= \frac{d(R_x h + R_y v + R_z)}{ah + bv + c} \end{aligned} \tag{6}$$

Since horizontal scanlines have a constant v coordinate, both the numerator and denominator in Equation (6) are linear in h and therefore can be computed using only one add for each pixel using the technique of forward differencing [3]. Then, to compute r for each pixel we must perform a division. Pseudocode for the inner loop of such a horizontal-scanline texture mapper is given in Figure 1. The pixels comprising the horizontal scanline are enumerated using the pixel pointer p. The variables N_r and D represent the numerator and denominator of Equation (6), and N_s represents the numerator of the corresponding equation for s. These three values are incrementally maintained in lines 6–8. The texture coordinates for the scanline are computed by the divides in lines 2 and 3.

Division is a slow operation on many modern processors, however, and can easily dominate the per-pixel cost of texture mapping. Many optimizations and approximations have been proposed to avoid the divides, including performing the two divides using one reciprocal and two multiplies, and approximating the reciprocal with a quadratic or piecewise linear function (also called subdividing affine) [5]. The tradeoff between the performance, simplicity, and accuracy of current texture mappers is largely controlled by the scheme chosen to avoid the divide in Equation (6). This paper provides a means for completely eliminating the divide in Equation (6) without sacrificing the simplicity and accuracy of the texture mapper in Figure 1, sigificantly improving its performance. In comparison, most other means of avoiding the divide are either quite complex or not very accurate.

1 while $p \neq p_{end}$ do
2 $\quad r = N_r / D$
3 $\quad s = N_s / D$
4 $\quad$ draw $texture(r,s)$ at pixel p.
5 $\quad p = p + 1$
6 $\quad N_r = N_r + R_x d$
7 $\quad N_s = N_s + S_x d$
8 $\quad D = D + a$

Figure 1: Pseudocode for the inner loop of a horizontal-scanline texture mapper.

4 Eliminating the divide

This section shows how to eliminate the divide required by the horizontal scanline code in Figure 1. We eliminate the divide by relaxing the restriction that scanlines must be horizontal. Instead, we will allow scanlines to exist at any angle on the screen, and require instead that they cross the polygon plane with a constant z value. The constant-z condition will allow us to compute the texture coordinates of the scanline without using a divide for every pixel.

We define an ***ideal scanline*** as the set of points $\langle h, v \rangle$ parameterized by integral h and satisfying the equation

$$v = v_0 + kh \,,$$

where k is the ***slope*** of the scanline and v_0 is an integer. In this paper, we shall restrict ourselves to the case $0 \leq k \leq 1$, as other scanline slopes can be treated by swapping h and v and/or changing their sign where appropriate. For constant-z ideal scanlines, we show that the texture coordinates of an ideal scanline are a linear function of h. First, we compute the slope of the scanline that guarantees it has a constant z value. We start by substituting Equations (1) into Equation (2):

$$\begin{aligned} ahz + bvz + cz &= d \\ ahz + bz(v_0 + kh) + cz &= d \\ h(a + bk)z + (bv_0 + c)z &= d \,. \end{aligned} \tag{7}$$

Taking the derivative of Equation (7) with respect to h and enforcing the constant-z condition $dz/dh = 0$,

$$\begin{aligned} (a+bk)z &= 0 \\ k &= -\frac{a}{b} \ . \end{aligned} \tag{8}$$

The scanline slope k is independent of v_0, so the same slope applies to all ideal scanlines of a polygon. The depth z for each scanline is derived by substituting Equation (8) into Equation (7):

$$\begin{aligned} h(a+(-\frac{a}{b})b)z+(bv_0+c)z &= d \\ (bv_0+c)z &= d \\ z &= \frac{d}{bv_0+c} \ . \end{aligned} \tag{9}$$

As advertised, the depth z of the ideal scanline is constant (does not vary with h). The texture coordinates corresponding to the screen coordinates $\langle h, v_0+kh\rangle$ are then

$$\begin{aligned} r &= R_x x + R_y y + R_z z \\ &= R_x hz + R_y(v_0+kh)z + R_z z \\ &= z(R_y v_0 + R_z) + hz(R_x + R_y k) \ . \end{aligned} \tag{10}$$

From this formula it is clear that since z is constant, r is linearly related to h and thus we compute it with additions using forward differencing (superscripts in parentheses denote evaluation at different values of h):

$$\begin{aligned} r^{(0)} &= z(R_y v_0 + R_z) \\ r^{(h)} &= r^{(h-1)} + z(R_x + R_y k) \ . \end{aligned}$$

Unfortunately, computing the texture coordinates on the ideal scanline is not sufficient for an accurate texture mapper. Because the ideal scanline does not necessarily consist of integer screen coordinates, it cannot be drawn exactly. Instead, we must draw an approximation to the ideal scanline, which we call a ***scanpath***, satisfying the related equation[1]

$$v = v_0 + \lfloor kh \rceil \ .$$

Note that the scanpath does not lie on the ideal scanline, but it approximates the ideal scanline to within half a pixel. We define the vertical distance Δ between the scanpath and the ideal scanline as follows:

$$\begin{aligned} \Delta &= v_0 + \lfloor kh \rceil - (v_0+kh) \\ &= \lfloor kh \rceil - kh \ . \end{aligned} \tag{11}$$

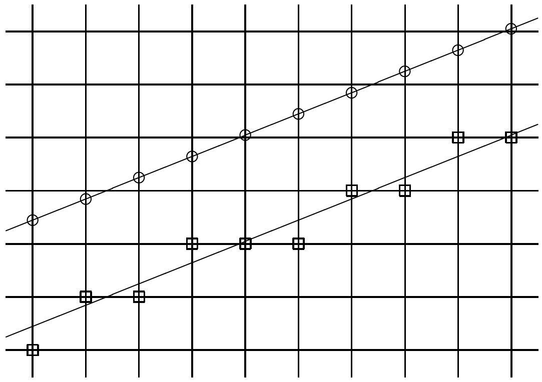

Figure 2: Diagram of an ideal scanline (circles) and a scanpath (squares). Pixel centers occur at the intersections of the gridlines. Accurate texture mapping requires sampling on the scanpath, whereas traditional constant-z texture mapping samples at the non-integral points along the ideal scanline.

Figure 2 shows an example of an ideal scanline and a scanpath. It is easy to verify that the scanpaths of a given slope cover every pixel in the display and no two scanpaths (ones with different v_0) cover the same pixel.

One possible texture mapping algorithm, shown in Figure 3, is to draw the texture samples obtained

[1] $\lfloor a \rceil$ is the nearest integer to a, with fractional values of $1/2$ rounded up.

```
1  while p ≠ p_end do
2      draw texture(r,s) at pixel p
3      Δ = Δ − k
4      if Δ > −1/2
5          p = p + 1
6      else
7          p = p + (N + 1)
8          Δ = Δ + 1
9      r = r + z(R_x + R_y k)
10     s = s + z(S_x + S_y k)
```

Figure 3: Pseudocode for the inner loop of an ideal scanline constant-z texture mapper.

from the ideal scanline at the pixels specified by its corresponding scanpath. The scanpath is computed using a DDA-style algorithm with Δ as the fractional v coordinate [8] in lines 3–8, and the texture coordinates are computed using Equation (10) in lines 9 and 10.

This simple texture mapper, however, is not very accurate, because samples taken from the ideal scanline are drawn not at the sampled location, but on the scanpath, up to a half a pixel away. Half a pixel does not sound like a large error, but this error is the same for all pixels in the same column of the screen, so if the error changes from column to column, a shearing effect of up to one pixel in size occurs. This shearing can cause the resulting texture to look "wavy".

The next section describes our new algorithm for fixing this waviness by adding a correcting term to the texture coordinates to more accurately sample the texture from the scanpath instead of from the ideal scanline.

5 Eliminating the Waviness

This section gives a new algorithm for correcting the simple constant-z scanline scheme described in the previous section to take into account the fact that the scanpath does not lie exactly on the ideal scanline. Our new algorithm corrects our approximation of the texture coordinate r by computing a first-order approximation in the error Δ from the ideal scanline. Since Δ is small ($|\Delta| \leq 1/2$), this approximation will improve our sampling accuracy.[2] To step off the ideal scanline, we first calculate the differential change in the texture coordinate $\partial r/\partial v$ and then compute our improved approximation to the texture coordinate as

$$r = \bar{r} + \Delta \partial r/\partial v \ , \tag{12}$$

where $\bar{r}$ is our approximation for r on the ideal scanline (Equation (10)).

We now compute $\partial r/\partial v$. We start by keeping h constant and taking the partial derivative of Equations (1) with respect to v:

$$\frac{\partial x}{\partial v} = \frac{\partial}{\partial v}(hz) = h\frac{\partial z}{\partial v} \tag{13}$$

$$\frac{\partial y}{\partial v} = \frac{\partial}{\partial v}(vz) = z + v\frac{\partial z}{\partial v} \tag{14}$$

Taking the partial derivative of Equation (2) with respect to v and substituting these derivatives, we solve for $\partial z/\partial v$:

$$\begin{aligned} a\frac{\partial x}{\partial v} + b\frac{\partial y}{\partial v} + c\frac{\partial z}{\partial v} &= 0 \\ ah\frac{\partial z}{\partial v} + bz + bv\frac{\partial z}{\partial v} + c\frac{\partial z}{\partial v} &= 0 \\ \frac{\partial z}{\partial v} &= \frac{-bz}{ah + bv + c} \end{aligned}$$

[2] Our simplifying choice of $H = V = 1$ underestimates the strength of this approximation. Working through the math in the general case reveals that our approximation is first order in Δ/V, a much smaller quantity since we typically have $V \gg 1$.

$$\frac{\partial z}{\partial v} = \frac{-bz^2}{ax+by+cz} \quad (15)$$

$$\frac{\partial z}{\partial v} = -\frac{bz^2}{d}\,, \quad (16)$$

since $ax+by+cz=d$. Substituting Equation (16) back into Equations (13) and (14), we get the following expressions for $\partial x/\partial v$ and $\partial y/\partial v$:

$$\begin{aligned}
\frac{\partial x}{\partial v} &= h\frac{\partial z}{\partial v} = -h\frac{bz^2}{d} && (17)\\
\frac{\partial y}{\partial v} &= z + v\frac{\partial z}{\partial v}\\
&= z - \frac{bvz^2}{d}\\
&= z\frac{d-bvz}{d}\\
&= z\frac{ax+by+cz-bvz}{d}\\
&= z\frac{ax+cz}{d}\\
&= \frac{z^2}{d}(ah+c)\,. && (18)
\end{aligned}$$

On the ideal scanline, z is constant, so these derivatives are linear in h![3] This crucial fact allows us to compute any linear combination of these derivatives using forward differencing along the ideal scanline and then use the resulting values to correct the texture coordinates for the deviation Δ from the ideal scanline. In particular, we derive $\partial r/\partial v$ from Equation (3):

$$\begin{aligned}
\frac{\partial r}{\partial v} &= R_x\frac{\partial x}{\partial v} + R_y\frac{\partial y}{\partial v} + R_z\frac{\partial z}{\partial v}\\
&= \frac{z^2}{d}(-R_xbh + R_y(ah+c) - R_zb)\\
&= \frac{z^2}{d}(R_yc - R_zb) + h\frac{z^2}{d}(R_ya - R_xb)\,.
\end{aligned}$$

[3]This fact appears to be serendipitous — we got lucky when the denominator of Equation (15) turned out to be a constant.

Since $\partial r/\partial v$ is linear in h, we compute it using forward differencing along the scanline, and then compute r itself using Equation (12).

Putting these equations together, we have the following per-scanline constants:

$$\begin{aligned}
P_r &= z(R_yv_0 + R_z)\\
Q_r &= z(R_x + R_yk)\\
T_r &= \frac{z^2}{d}(R_yc - R_zb)\\
U_r &= \frac{z^2}{d}(R_ya - R_xb)
\end{aligned}$$

and the following equation for r:

$$r = P_r + hQ_r + \Delta(T_r + hU_r) \quad (19)$$

Pseudocode for our first-order calculation of $\langle r,s\rangle$ is shown in Figure 4. To compute r incrementally along a scanline, we first compute k, z, P_r, Q_r, T_r, and U_r, which are constant for each scanline. Then, for each pixel, we compute p (a pointer to the current pixel) and Δ using the same incremental line-drawing algorithm from Figure 3 (lines 5–10), and compute the values $\bar{r} = P_r + hQ_r$ and $C_r = T_r + hU_r$ using forward differencing (lines 11–12). The texture coordinate r can then be computed using Equation (19) (line 2).

In the next section, we show how to further optimize the code in Figure 4 by replacing the multiplies by Δ in lines 2 and 3 with more forward differencing. The resulting code has no expensive operations (multiplies or divides) in the inner loop, and calculates the same first-order approximation as does the code in Figure 4.

6 Eliminating the multiply

This section shows how to eliminate the expensive multiply operations in lines 2 and 3 in Figure 4.

```
while p ≠ p_end do
        r = r̄ + Δ * C_r
        s = s̄ + Δ * C_s
        draw texture(r, s) at pixel p.
        Δ = Δ − k
        if Δ > −1/2
                p = p + 1
        else
                p = p + (N + 1)
                Δ = Δ + 1
        r̄ = r̄ + Q_r
        C_r = C_r + U_r
        s̄ = s̄ + Q_s
        C_s = C_s + U_s
```

Figure 4: Pseudocode for a constant-z scanline texture mapper with a first-order correction in Δ.

```
while p ≠ p_end do
        draw texture(r, s) at pixel p.
        Δ = Δ − k
        if Δ > −1/2
                p = p + 1
                α_r = α_r − 2kU_r
                r = r + α_r
                α_s = α_s − 2kU_s
                s = s + α_s
        else
                p = p + (N + 1)
                Δ = Δ + 1
                α_r = α_r + (1 − 2k)U_r
                r = r + α_r + β_r
                α_s = α_s + (1 − 2k)U_s
                s = s + α_s + β_s
        β_r = β_r + U_r
        β_s = β_s + U_s
```

Figure 5: Pseudocode listing for the optimized constant-z texture mapper. Only the inner loop is shown.

Our trick is to use forward differencing to compute those multiplies using only additions. To derive these new forward difference equations, we compute the change in the texture coordinate r from one iteration to the next:

$$
\begin{aligned}
& r^{(h+1)} - r^{(h)} \\
&= P_r + (h+1)Q_r + \Delta^{(h+1)}(T_r + (h+1)U_r) \\
&\quad -P_r - hQ_r - \Delta^{(h)}(T_r + hU_r) \\
&= Q_r + (\lfloor kh+k \rceil - (kh+k))(T_r + (h+1)U_r) - \\
&\quad (\lfloor kh \rceil - kh)(T_r + hU_r) \\
&= Q_r + (\lfloor kh+k \rceil - \lfloor kh \rceil)(T_r + hU_r) \\
&\quad -k(T_r + U_r) - 2kU_r h + \lfloor kh+k \rceil U_r
\end{aligned}
$$

We define some new terms for pieces of this difference that we maintain incrementally:

$$
\begin{aligned}
\alpha_r &= Q_r - k(T_r + U_r) - 2kU_r h + \lfloor kh+k \rceil U_r \\
\beta_r &= T_r + hU_r
\end{aligned}
$$

We then define the following forward difference equations for r, depending on the value of $\lfloor kh+k \rceil$ versus $\lfloor kh \rceil$:

- $\lfloor kh+k \rceil = \lfloor kh \rceil$

$$
\begin{aligned}
\alpha_r^{(h)} &= \alpha_r^{(h-1)} - 2kU_r \\
r^{(h+1)} &= r^{(h)} + \alpha_r^{(h)} \\
\beta_r^{(h+1)} &= \beta_r^{(h)} + U_r
\end{aligned}
$$

- $\lfloor kh+k \rceil = \lfloor kh \rceil + 1$

$$
\begin{aligned}
\alpha_r^{(h)} &= \alpha_r^{(h-1)} - 2kU_r + U_r \\
r^{(h+1)} &= r^{(h)} + \alpha_r^{(h)} + \beta_r^{(h)} \\
\beta_r^{(h+1)} &= \beta_r^{(h)} + U_r
\end{aligned}
$$

Figure 5 gives pseudocode for the inner loop of our final constant-z scanline texture mapper. In addition, the constants k, U_r, $-2kU_r$, $(1-2k)U_r$, U_s,

$-2kU_s$, $(1-2k)U_s$, p_{end}, and $N+1$, as well as the initial values of p, Δ, r, α_r, β_r, s, α_s, and β_s need to be calculated before this loop. The code uses 8 or 11 additions, 2 comparisons, and 1 store per iteration, and has 19 live variables.[4] All expensive arithmetic operations (multiplications and divisons) have been eliminated from the inner loop while retaining the accuracy of the first-order approximation algorithm from Figure 4.

7 Conclusion

This section describes some experiments performed with our diagonal-scanline texture mapper and offers some concluding remarks.

We performed some simple experiments to demonstrate that the accuracy of our algorithm is comparable to or better than existing polygon rendering algorithms. In particular, we compared the RMS error of three approximation codes: our first-order correction code from Figure 5, the ideal scanline code from Figure 3, and a traditional horizontal scanline code that approximates the texture coordinates by linearly interpolating between exact coordinates computed every 16 pixels. Our experiment consisted of drawing a single polygon at various orientations on a screen with $H = V = 256$ and measuring the RMS error of the computed texture coordinates versus the correct coordinates computed with Equation (6). The experiments showed that the ideal scanline algorithm is almost two orders of magnitude less accurate than the other algorithms, demonstrating the inaccuracy observed in the traditional application of diagonal scanlines. Our first-order correction fixes this deficiency by achieving an error comparable with the interpolation algorithm for most orientations. For viewing from nearly edge-on, however, our algorithm does even better, beating the linear interpolation algorithm by more than an order of magnitude. Thus, our algorithm provides accurate polygon rendering at all possible polygon orientations.

By drawing scanpaths with nearly constant z values, we can reduce the cost of computing a variety of depth-cued effects. These costs include computations used in z-buffering, mip-mapping, fog, and haze. If the z value of the ideal scanline is accurate enough, the z-dependent portions of these algorithms can be factored out of the inner loop and performed only once per scanline. If a more accurate z value than the z value of the ideal scanline is required, a forward differencing algorithm for z can be derived in a similar manner to the one given for r.

Diagonal scanlines do have drawbacks. It is harder to clip a scanpath against the edge of a polygon and, more ominously, a scanpath can cross a polygon edge multiple times because it is not a straight line. This problem is solvable by doing some extra work at the beginning and end of each scanline to ensure that only the correct pixels are rendered, but requires additional code complexity. Another drawback of diagonal scanlines is that the precomputation for each scanline is somewhat more involved than for traditional horizontal scanlines. Also, the scanline drawing algorithm from Figure 5 consumes a lot of machine registers, which can be a problem on Pentium-class processors with few available registers. All of these difficulties, however, can be overcome to create an accurate constant-z texture mapper.

A more insidious problem arising from using diagonal scanlines is that modern cache architectures are not necessarily optimized to handle the resulting memory access pattern.[5] With horizontal scanlines, the same cache line of screen memory is typically accessed several times before the next one is

[4] The additional code required to compute the *texture* function is not included in this count.

[5] In fact, video memory is sometimes not cached at all.

accessed. With diagonal scanlines, however, a worst case can occur where each reference accesses a new cache line. In this case, a given cache line is not referenced again until the next scanline is processed, so the number of memory references separating two references to the same cache line is equal to the length of the scanline. Typical sizes of first-level caches in modern processors are on the order of 512 cache lines, so drawing scanlines with lengths longer than 512 can cause every memory reference to miss in the cache. For certain values of k and N, the problem can be even worse as cache conflicts due to a lack of associativity can occur. Experimental experience with this problem on a PowerPC 604 processor (16KB data cache, 512 lines, 4-way set associative) and a PowerPC 750 processor (32KB data cache, 1024 lines, 8-way set associative) suggests that its effects can be limited by restricting the length of scanlines to be comfortably smaller than the number of lines in the cache.

The algorithm introduced in this paper can be improved and extended in several ways. Although our algorithm is presented for texture-mapping, it applies equally to other forms of perspective three-dimensional rendering of flat surfaces, such as phong and gouraud shading. If greater accuracy is needed, higher-order approximations in Δ can be computed in a similar manner to the first-order approximation outlined here. Antialiasing techniques [4] can be combined with our algorithm in a straightforward fashion. It would be interesting to extend our algorithm to curved-surface texture mapping, where a curve of constant z replaces the ideal scanline.

Acknowledgements

Thanks to Andrew Kirmse for getting me started on texture mapping.

References

[1] Franklin C. Crow. Summed-area tables for texture mapping. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):207–212, July 1984.

[2] James D. Foley and Andries van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, Reading, Massachusetts, 1982.

[3] Andrew S. Glassner, editor. *Graphics Gems*. Academic Press Professional, Boston, 1990.

[4] Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, pages 56–67, May 1986.

[5] Chris Hecker. Perspective texture mapping, part 4: Approximations. *Game Developer Magazine*, December 1995.

[6] Rod Salmon and Mel Slater. *Computer Graphics: Systems and Concepts*. Addison-Wesley, Reading, Massachusetts, 1987.

[7] Alvy Ray Smith. Incremental rendering of textures in perspective. *SIGGRAPH '80 Animation Graphics seminar notes*, July 1980.

[8] Rojer W. Swanson and Larry J. Thayer. A fast shaded-polygon renderer. *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(4):95–101, August 1986.

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