Direct Calculation of MIP - Map Level for Faster Texture Mapping

Kevin Wu
Computer Systems Laboratory
HPL-98-112
June, 1998

E-mail: kevinwu@hpl.hp.com

computer graphics, texture mapping, MIP mapping, trilinear filtering, floating-point arithmetic

This paper describes a method applicable to texture mapping for calculating the MIP-map level directly from the IEEE-standard floating-point representation of the compression value. This calculation ordinarily requires execution of a base-2 logarithm, and it needs to be performed once per pixel. However, the full logarithm need not be evaluated because only the integer part of the result is required. The MIP-map level is essentially the base-2 exponent of the MIP map compression value, and this exponent is immediately available from the IEEE-standard floating-point representation.

Internal Accession Date Only

1. Introduction

In texture mapping implementations, MIP mapping [1] is a common anti-aliasing technique. A MIP map is a multi-resolution image pyramid. The highest resolution image is level 0 at the base of the pyramid. The next higher level is formed by convolving the current level with a low-pass filter kernel and subsampling each direction by a factor of two. By recursively applying this process, all levels of the pyramid are defined with the highest level consisting of a single texture element (a.k.a. texel). The MIP map saves computation during rendering because the appropriate texture resolution is available in prefiltered form whether a textured surface is near or far from the viewer. For decimation or so-called "minification" where a lower resolution is required, a continuous value *d* quantitatively describes the spatial compression of texels from texture space to screen space. The MIP-map level *k* selected during rendering is the largest integer less than or equal to the base-2 logarithm of *d*. The calculation is performed at each pixel so avoiding evaluation of the logarithm is desirable.

Direct calculation of the MIP-map level from the compression value *d* is possible with only a simple mask-shift-subtraction sequence when *d* is in IEEE floating-point format. The MIP-map level *k* is essentially the base-2 exponent of *d*.

2. IEEE Floating-Point Format

A number in IEEE floating-point format [2, 3] has three fields: a sign bit *s*, an exponent *e*, and a fraction *f*. For single-precision numbers, *e* is 8 bits and *f* is 23 bits in size. For double-precision numbers, *e* is 11 bits and *f* is 52 bits in size. The sign bit is the most significant bit, followed by the exponent in the next most significant bits, followed by the fraction in the least significant bits. All of the following discussion applies to single-precision numbers; the extension to double-precision numbers is straightforward.

The following rules determine the value of a single-precision floating-point number:

1. If $e = 255$ and $f \neq 0$, then the value is NaN (not a number) regardless of *s*.
2. If $e = 255$ and $f = 0$, then the value is $(-1)^s \infty$.
3. If $0 < e < 255$, then the value is $(-1)^s 2^{e-127} (1.f)$.
4. If $e = 0$ and $f \neq 0$, then the value is $(-1)^s 2^{-126} (0.f)$.
5. If $e = 0$ and $f = 0$, then the value is $(-1)^s 0$.

Rule 3 (normalized form) applies to most numbers that can be represented with this format. Rule 4 (denormalized form) extends the range for numbers that are very small in magnitude.

3. MIP Map Filter Size

A MIP map stores multiple resolutions of a texture in a precalculated image pyramid. During rendering, the scan converter interpolates texture coordinates across a surface. At each pixel, the scan converter maps a color from the texture to the pixel. This color should be a blend of all texels in the pixel's projection onto texture space where the filter profile determines the weights applied to the texels. For square pixels, the projection is an arbitrary quadrilateral for perspective projections when the surface is a triangle. A local affine approximation [4] of the pixel's footprint in texture space is a parallelogram with the Jacobian basis vectors forming the edges as shown in Figure 1. The Jacobian basis vectors are the columns of the Jacobian matrix **J** from screen to texture space. The elements of the Jacobian matrix are the partial derivatives of the texture coordinates with respect to the screen coordinates.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial s}{\partial x} & \frac{\partial s}{\partial y} \\ \frac{\partial t}{\partial x} & \frac{\partial t}{\partial y} \end{bmatrix} = \begin{bmatrix} s_x & s_y \\ t_x & t_y \end{bmatrix} \tag{1}$$

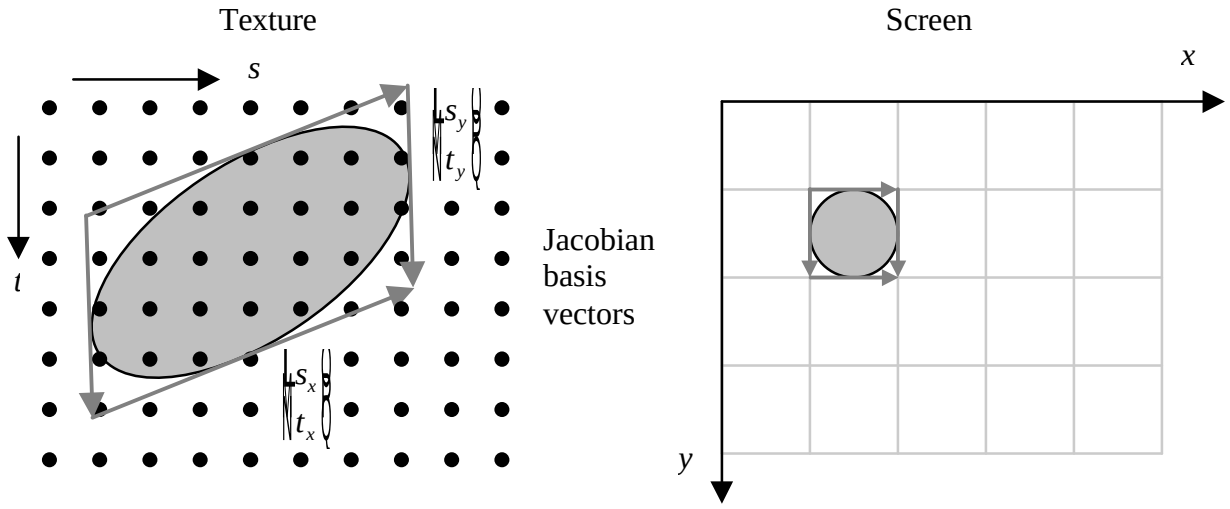

Figure 1: Pixel's footprint in texture space.

MIP-map levels are formed by applying a square filter. By implication, MIP mapping accomplishes anti-aliasing by approximating the parallelogram of the pixel's footprint in texture space with a square filter shape. Williams [1] uses Heckbert's suggestion [5] for computing the texture compression value *d*. The approximate size of the square filter shape is the length of the longer of the two Jacobian basis vectors.

$$d = \max\left(\left\|\begin{matrix} s_x \\ t_x \end{matrix}\right\|, \left\|\begin{matrix} s_y \\ t_y \end{matrix}\right\|\right) \tag{2}$$

With this selection of d, the textured surface is excessively blurred when the pixel footprint is long and narrow, particularly when blending texels with bilinear or trilinear interpolation [6]. However, these texel blending operations are simpler to implement in hardware than higher quality anisotropic texture filtering methods [4, 7, 8, 9, 10, 11].

4. Calculation of MIP-Map Level

In a MIP map, the relationship between the compression value d and the MIP-map level k is established by two conditions. Let the vertical and horizontal distance between adjacent texels be 1 in level 0, and consider a Jacobian matrix equal to the identity matrix. A unit change in x or y produces a unit change in s or t, respectively. For this case, the length of either Jacobian basis vector is 1 so $d = 1$. Thus, the first condition is that when $d = 1$, the appropriate choice of MIP-map level is $k = 0$. The second condition is that the resolution of MIP-map level $k + 1$ is half that of level k. In other words, the compression value d is twice as large at level $k + 1$ than at level k. With these two constraints, we seek the MIP-map level k for an arbitrary compression value d such that

$$2^k \le d < 2^{k+1} \tag{3}$$

or equivalently

$$k = \lfloor \log_2 d \rfloor \tag{4}$$

where the floor function $\lfloor a \rfloor$ is the largest integer less than or equal a. As a practical consideration, this computed value of k needs to be clamped between 0 and the highest level of the MIP map before attempting to read texels.

The straightforward approach to evaluating Equation (4) begins with computing $\log_2 d$ with the natural logarithm function of a standard math library.

$$\log_2 d = \frac{\ln d}{\ln 2} \tag{5}$$

This approach computes $\ln d$ and multiplies by the precomputed value of $1/\ln 2$. Finally, this product is the argument of the floor function of a standard math library. However, two calls to library functions per pixel are time consuming.

A faster approach begins with the observation that $\lfloor \log_2 d \rfloor$ of a binary number d is essentially the number of bits from the binary point to the most significant digit. If the binary representation of d is

$$d = 1\underbrace{xxx\cdots x}_{n}.xx\cdots x \tag{6}$$

then the MIP-map level is $k = n$. This approach computes the MIP-map level by counting the number of bits from the binary point to the most significant digit. A binary search ($O(\log n)$) would be even faster than a linear one ($O(n)$).

The approach taken in this paper depends on representing d in IEEE floating-point format. In particular, d is most likely to be in the form given by Rule 3 of Section 2.

$$d = (-1)^s 2^{e-127}(1.f) \tag{7}$$

Since d is the length of a Jacobian basis vector, it is non-negative so $s = 0$. Applying $\log_2$ yields an intermediate result.

$$\log_2 d = \log_2\left(2^{e-127}(1.f)\right) = \underbrace{e-127}_{\text{integer}} + \underbrace{\log_2(1.f)}_{\text{fraction}} \tag{8}$$

Note that $1.f$ is in the range $[1, 2)$ so $\log_2(1.f)$ is in the range $[0, 1)$. Applying the floor function yields k.

$$k = \lfloor \log_2 d \rfloor = e - 127 \tag{9}$$

The algorithm for computing the MIP-map level k from the compression value d in IEEE floating-point format is as follows:

1. Apply a mask to d to set the sign bit s and the fraction f to 0 without changing the exponent e.
2. Shift the result of Step 1 to the right by the width of f. This gives e as an integer.
3. Subtract 127 from the result of Step 2.
4. Clamp the result of Step 3 to the range of 0 and the highest level of the MIP map.
5. The result of Step 4 is k.

As long as the value of d is not NaN, this algorithm returns the correct result because of the clamping step. This approach is simpler and likely faster than the preceding two approaches.

5. Interlevel Interpolation Value

Interlevel interpolation uses the compression value *d* to interpolate texture colors between MIP-map levels. Linear and trilinear texture filtering use interlevel interpolation to create smooth transitions between MIP-map levels. Starting with Range (3), we apply linear interpolation between the two levels *k* and *k* + 1.

$$d = 2^k + l\left(2^{k+1} - 2^k\right) \qquad 0 \le l < 1 \tag{10}$$

Solving for l gives the interlevel interpolation value.

$$l = 2^{-k} d - 1 \qquad 0 \le l < 1 \tag{11}$$

A slightly faster way to evaluate this equation is available by substituting Equation (9) into Equation (7) and noting that *s* is 0.

$$d = 2^k (1.f) \tag{12}$$

Then we substitute Equation (12) into Equation (11).

$$l = 2^{-k}\left(2^k (1.f)\right) - 1 = 1.f - 1 = 0.f \tag{13}$$

To calculate l in 9.23 fixed-point format, we set *s* and *e* to 0 while leaving *f* unchanged. Alternatively, to calculate l in floating-point format, we calculate $1.f$ in floating-point format by setting *e* in *d* to 127 (because we want 2^0 and the offset in IEEE floating-point format is 127) and subtracting 1.0.

6. Conclusions

A fast method for calculating the MIP-map level is possible via direct processing of the IEEE floating-point form of the texture compression value. The mask-shift-subtract sequence is faster than the straightforward approach of computing the base-2 logarithm followed by the floor function. In addition, a slightly faster calculation of the interlevel interpolation value is possible via direct processing.

1. References

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

[2] *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Standard 754-1985, Institute of Electrical and Electronics Engineers, New York, N. Y., 1985.

[3] Walt Donovan and Tim Van Hook, Direct Outcode Calculation for Faster Clip Testing, in *Graphics Gems IV*, Paul S. Heckbert, ed., AP Professional, Cambridge, Mass., 1994.

[4] 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.

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

[6] Alan Watt and Mark Watt, *Advanced Animation and Rendering Techniques: Theory and Practice*, Addison-Wesley, Wokingham, England, 1992.

[7] Elliot A. Feibush, Marc Levoy, Robert L. Cook, Synthetic Texturing Using Digital Filters, *Computer Graphics*, SIGGRAPH 1980 Proceedings, 14(3):294–301, July 1980.

[8] M. Gangnet, D. Perny, and P. Coueignoux, Perspective Mapping of Planar Textures, *Eurographics 1982*, pp. 57–71, September 1982.

[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.