RC 20357 (01/29/96)
Computer Science

IBM Research Report

Efficient Evaluation of $1/x$ for Texture Mapping

Chandrasekhar Narayanaswami
William Luken

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Abstract

Accounting for the effect of perspective when performing a texture mapped area fill operation requires at least one division for each pixel in the filled area. This division can be costly in terms of the number of processor cycles required as well as in the number of gates required to provide a hardware divider. This paper considers the number of operations and the relationship between table size and accuracy obtainable using a novel combination of several table look up methods to approximate 1/x.

1 Introduction

Texture mapping [1] provides a method for projecting an image on to a set of triangles or polygons. This technique is widely used to render three dimensional (3D) models with greater realism than is obtained with simple shaded area fills. This makes it possible to represent 3D models with complex surface characteristics that would otherwise require very large numbers of small polygons.

In its simplest form, texture mapping involves assigning pairs of texture coordinates (u, v) to each vertex of a triangle and determining texture coordinates $(\tilde{u}, \tilde{v})$ for each pixel comprising the triangle by means of linear interpolation between the vertices of the triangle. The values of $\tilde{u}$ and $\tilde{v}$ are then used to determine the address of a pixel in a texture image. Polygons with more than three vertices may be handled by decomposition into a set of triangles. This simple approach neglects the effects of perspective projection resulting in undesirable artifacts such as those shown in Figure 15 and 18.

In order to correct the texture mapping for the effects of perspective, it is necessary to replace the values of the texture coordinates u and v at each vertex with $u^{'} = u/z$ and $v^{'} = v/z$ and $q^{'} = 1/z$, where z is the depth value at each vertex. The values of $u^{'}$, $v^{'}$, $q^{'}$ are then interpolated between the vertices of each triangle to determine the values of $\tilde{u}^{'}$, $\tilde{v}^{'}$, and $\tilde{q}^{'}$ at each pixel comprising the triangle. The perspective corrected texture coordinates at each pixel are then determined by $\tilde{u} = \tilde{u}^{'}/\tilde{q}^{'}$ and $\tilde{v} = \tilde{v}^{'}/\tilde{q}^{'}$. This results in a perspective corrected image such as that shown in Figures 17 and 20.

The number of divisions required at each pixel can be reduced from two to one by calculating $\hat{q} = 1/\tilde{q}^{'}$, followed by $\tilde{u} = \tilde{u}^{'}\hat{q}$, and $\tilde{v} = \tilde{v}^{'}\hat{q}$. In the case of integer arithmetic, this may be accomplished by calculating $\hat{q} = K/\tilde{q}^{'}$ and $\tilde{u} = \tilde{u}^{'}\hat{q}/K$, and $\tilde{v} = \tilde{v}^{'}\hat{q}/K$ where $K = 2^k$ is a power of 2 and division by K is accomplished by discarding the k least significant bits of the product $\tilde{u}^{'}\hat{q}$.

One common approach used to speed up evaluation of complex functions such as x^n, $\cos(x)$, $\log x$, etc., is to look up values in precomputed tables. This is a classic example of a time-space trade-off and the question of the optimal balance point arises naturally. One such example is the approximation of x^n for lighting calculations [2] in the geometry pipeline.

Several algorithms for calculating approximations to K/q based on table look-up are presented and compared in this paper. These algorithms differ in the number of operations and the sizes of the tables required to achieve sufficient accuracy. The resulting accuracy should be adequate to select the correct element of a texture map in most cases. Since more texture maps have dimensions which are smaller than 1000×1000 pixels, an accuracy of one part in 1000 is usually adequate. Moreover,

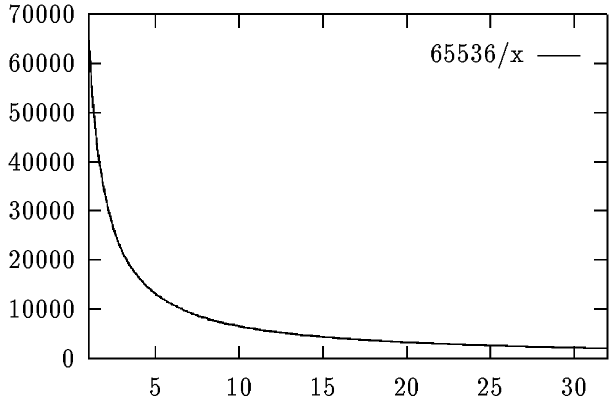

Figure 1: Plot of the $65536/x$ for x in [1,32]

when bilinear or other filtering is performed as part of the texture mapping, the effects of small errors in the texture coordinates is even less noticeable.
Before introducing the algorithms for obtaining values of K/x, it is instructive to consider the shape of this function as shown in Figure 1. For large values of x, the portion of this function is bounded by a unit step in x, i.e., $f(x)$ to $f(x+1)$ is well represented by linear interpolation. For small values of x, however, the deviations between $f(x)$ and the linear approximation become substantial over the range of $x_0 < x \leq x_0 + 1$ and these deviations become unbounded as x approaches zero.

2 Simple look-up

Consider a table composed of N entries consisting of values of K/x corresponding to values of $x = x_0, x_1, \cdots, x_{N-1}$, where K is any positive number. The ith entry in this table may be expressed as $f[i] = K/x_i$. Let $\Delta_{xi} \equiv x_{i+1} - x_i$. With a uniform subdivision of the input range $[x_0, x_{N-1}]$, $\Delta_{xi} = K/(N-1)$.
A simple direct look-up approximation to $f(x)$ may be defined by $f_0(x) = f[j]$ where j is determined by the condition, $x_j \leq x < x_{j+1}$. If a uniform subdivision is used, j may be determined by $(x - x_0)/\Delta_x$. The errors in the resulting approximation are illustrated in Figure 2.
Due to the limited number of samples of $f(x)$ stored in the look-up table, this simple direct look-up algorithm incurs a quantization error which depends on the value of x and the difference δ between x and the value of x_j corresponding to the table entry. The resulting relative error may be defined as

$$e_0 \equiv \frac{f_0(x) - f(x)}{f(x)} = \frac{f[j] - f(x_j + \delta)}{f(x)} = \frac{\delta}{x_j}$$

This error will be largest when $x_j = x_0$ (see Figure 2 also) and $\delta = \Delta_x$, or

$$e_{max} = \frac{\Delta_x}{x_0}$$

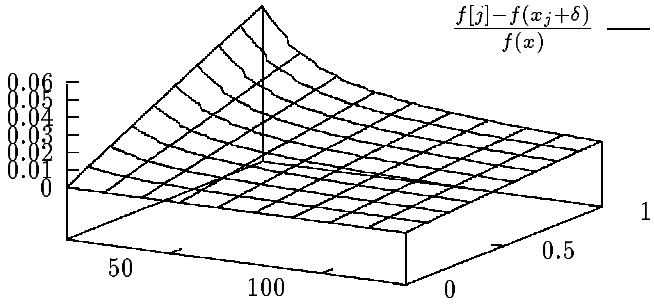

Figure 2: Plot of family of relative errors with simple look-up. x_i is plotted on the x axis and δ, the difference between the actual value of x and x_j the value of x for which a look-up table entry exists, is plotted on the y axis. It is assumed that there is an table entry for each integer value of x in the range [16,128].

Therefore, in order to represent the value of $f(x)$ to an accuracy of 1 part in 1000 for $x = 1$ to $M = 64k$ for example, the interval Δx must satisfy the bound

$$\frac{\Delta_x}{1} < \frac{1}{1000}$$

The number of entries in the resulting table is given by $N = (M-1)/\Delta_x$ or approximately 64 million, which is clearly not practical. As shown in the next section, the size of the table required to provide a specified accuracy is much smaller when using linear interpolation instead of a simple look-up.

3 Linear interpolation

The quantization errors associated with small look-up tables may be reduced through the use of linear interpolation between successive entries in the look-up table. The table stores the values of the function $f(x)$ at $x = x_i$, $i = 0, 1, \cdots, N-1$. Let $\Delta_{xi} \equiv x_{i+1} - x_i$, $\forall i = 0, 1, \cdots, N-2$. Then, by linear interpolation,

$$\begin{aligned} f(x_i + \delta) \approx f_l(x) &= (1 - \frac{\delta}{\Delta_{xi}})f(x_i) + \frac{\delta}{\Delta_{xi}} f(x_{i+1}) \\ &= f[i] + \frac{\delta}{\Delta_{xi}}(f[i+1] - f[i]) \end{aligned}$$

Figure 3 shows a geometric interpretation of the linear interpolation formula. With the above form there is a need to determine δ, followed by a divide, a subtract, a multiply, and an add operation. The division can be reduced to a shift by choosing $\Delta_{xi} = 2^p$. This should be the preferred mode of operation.
If it is not possible to make $\Delta_{xi} = 2^p$, a more convenient representation of linear interpolation, which has no divisions and fewer additions but uses twice as much

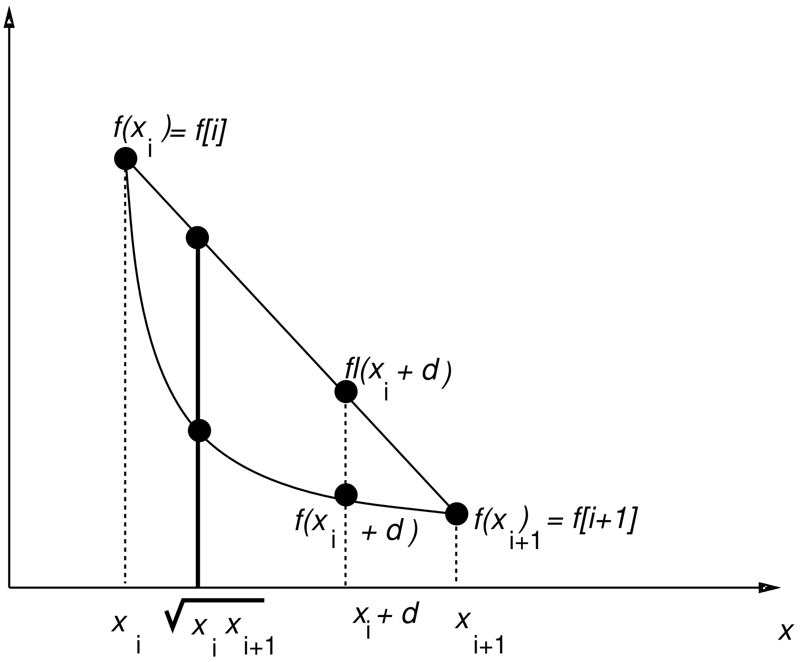

Figure 3: Geometric interpretation of the linear interpolation formula. The value at an intermediate point $x_i + d$ is given as $f_l(x_i + d) = (df[i+1] + (x_{i+1} - x_i - d)f[i])/(x_{i+1} - x_i)$

storage is to use two tables $g[]$ and $h[]$ as follows:

$$f(x_i + \delta) \approx f_l(x) = xg[i] + h[i] \quad = \quad (x_i + \delta)g[i] + h[i]$$

where $g[i]$ and $h[i]$ are defined by:

$$\begin{aligned} g[i] &= \frac{f[i+1] - f[i]}{\Delta_{xi}} \\ h[i] &= \frac{x_{i+1}f[i] - x_i f[i+1]}{\Delta_{xi}} \end{aligned}$$

The error between the linear approximation and the actual function is

$$e_l(\delta) \equiv f_l(x) - f(x) \quad = \quad f[i] + \frac{\delta}{\Delta_{xi}}(f[i+1] - f[i]) - f(x_i + \delta)$$

The value of δ_m where the error is a maximum is determined by the condition

$$\begin{aligned} \left.\frac{de_l(\delta)}{d\delta}\right|_{\delta_m} &= \frac{f[i+1] - f[i]}{\Delta_{xi}} + \frac{K}{(x_i + \delta_m)^2} = 0 \\ \Rightarrow \quad & x_i + \delta_m = \sqrt{x_i x_{i+1}} \end{aligned}$$

The maximum error is thus

$$\begin{aligned} e_{lmax} &\equiv e_l(\delta_m) \\ &= \frac{K(x_i + x_{i+1} - 2\sqrt{x_i x_{i+1}})}{x_i x_{i+1}} \end{aligned}$$

If $x_{i+1} = rx_i$

$$e_{lmax} = \frac{K(1 + r - 2\sqrt{r})}{rx_i}$$

When the error is maximum, the fractional error is

$$e_{lmax\_frac} \equiv \frac{e_{lmax}}{f(x)} = \sqrt{\frac{x_i}{x_{i+1}}} + \sqrt{\frac{x_{i+1}}{x_i}} - 2$$

If $x_{i+1} = rx_i$

$$e_{lmax\_frac} = \sqrt{\frac{1}{r}} + \sqrt{r} - 2 = \frac{(\sqrt{r}-1)^2}{\sqrt{r}}$$

When $x_{i+1} = 2x_i$ the fractional error at the point of maximum error is 1.2132×10^{-1}. When $x_{i+1} = 1.25x_i$ the fractional error at the point of maximum error is 1.246×10^{-2}. The maximum value of the relative errors occurs at the value of $\delta = \delta_{rel}$ where

$$\left.\frac{d}{d\delta}\frac{e_l(\delta)}{f(x)}\right|_{\delta_{rel}} = 0$$

$$\Rightarrow \quad x_i + \delta_{rel} = \frac{x_i + x_{i+1}}{2}$$

Thus the maximum relative error occurs at the center of the interval. The maximum relative error at this point is

$$e_{lmax\_rel} \equiv \frac{e_l(\delta_{rel})}{f(x_i + \delta_{rel})} = \frac{1}{4}[\frac{x_i}{x_{i+1}} + \frac{x_{i+1}}{x_i} - 2]$$

If $x_{i+1} = rx_i$

$$e_{lmax\_rel} = \frac{1 + r^2 - 2r}{4r} = \frac{(r-1)^2}{4r}$$

When $x_{i+1} = 2x_i$ the maximum relative error is 0.125 or 12.5%. When $x_{i+1} = 1.25x_i$ the maximum relative error is 0.0125 or 1.25%.
Table 1 shows values of e_{lmax}, e_{lmax_frac}, and e_{lmax_rel} for several values of x_i and x_{i+1}. Figure 4 shows $f(x)$ and $f_l(x)$, the linear approximation to $f(x)$. Figure 5 shows the actual and relative error in the linear approximation to $f(x)$. The errors are plotted in the interval [32,33] because this is where the maximum errors occur in the range [32,64] as shown in Figure 6.
Figures 6 and 7 show the maximum actual and the maximum relative error in the linear approximation to $f(x)$ as a function of $r = x_{i+1}/x_i$. The range for x is chosen to be [32,64] in these plots because that is the range used for analyzing the errors in the method presented in Section 5. The errors are less than 1/1000 and therefore satisfies accuracy requirements listed in the beginning of the paper.

4 Other approaches

As an alternative to linear interpolation, $f(x)$ may be approximated by quadratic interpolation. In this case $f(x)$ may be determined by

$$f_q(x) = f_q(x_i + \delta) = A[i]\delta^2 + B[i]\delta + C[i]$$

x_i	x_{i+1}	Max Error $e_{lmax}(K=1)$	Frac Max Error e_{lmax_frac}	Max Relative Error e_{lmax_rel}
16	17	5.5717×10^{-5}	9.1891×10^{-4}	9.1912×10^{-4}
32	33	7.2848×10^{-6}	2.3673×10^{-4}	2.3674×10^{-4}
64	65	9.4630×10^{-7}	6.0096×10^{-5}	6.0096×10^{-5}
128	129	1.1875×10^{-7}	1.5141×10^{-5}	1.5141×10^{-5}
256	257	1.4872×10^{-8}	3.7999×10^{-6}	3.7999×10^{-6}
512	513	1.8609×10^{-9}	9.5183×10^{-7}	9.5182×10^{-7}

Table 1: Table of errors for linear interpolation. e_{lmax}, e_{lmax_frac}, e_{lmax_rel} are shown.

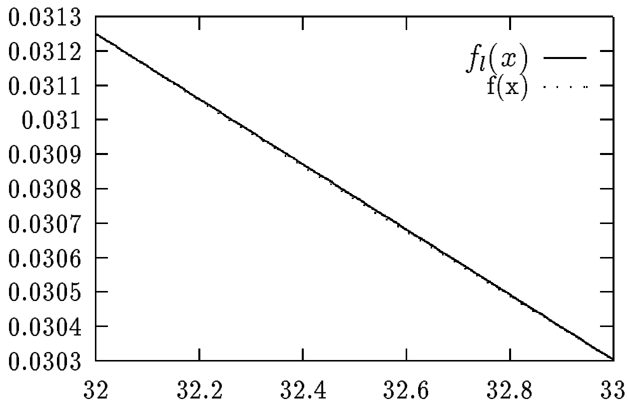

Figure 4: Plot of linear approximation to $1/x$. $f_l(x)$ is the approximation to $f(x)$.

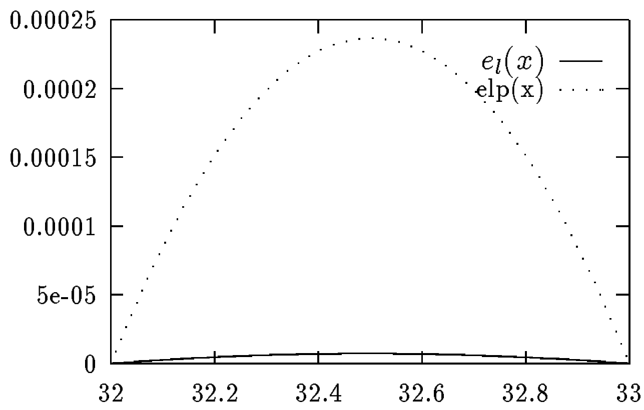

Figure 5: Plot of errors due to linear approximations to $1/x$. $e_l(x) = f_l(x) - f(x)$ is the error and $e_{lp}(x) = e_l(x)/f(x)$ is the relative error.

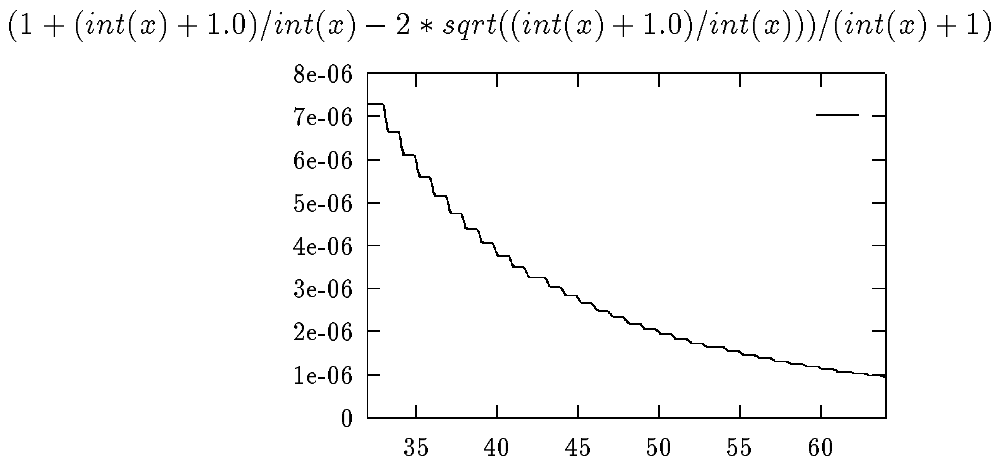

Figure 6: Maximum actual errors with linear interpolation when $x_i \leq x < x_{i+1}$ are plotted in the range [32,64]. For example when $32 \leq x < 33$ the maximum error is 7.2848×10^{-6}. There is a table entry for each integer x in the range [32,64]. $r = x_{i+1}/x_i$

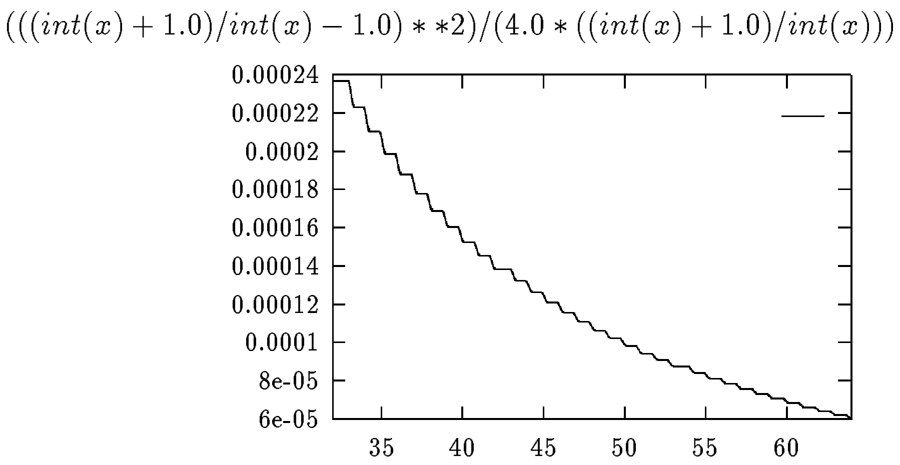

Figure 7: Maximum relative errors with linear interpolation when $x_i \leq x < x_{i+1}$ are plotted in the range [32,64]. For example when $32 \leq x < 33$ the maximum error is 2.3674×10^{-4}. There is a table entry for each integer x in the range [32,64]. $r = x_{i+1}/x_i$

where $x_i \leq x = x_i + \delta < x_{i+1}$ and $A[i]$, $B[i]$, and $C[i]$ are stored in precomputed tables. Clearly, $f_q(x_i) = C[i] = f[i]$ to match values $f_q(x)$ and $f(x)$ at $x = x_i$. $A[i]$ and $B[i]$ may be chosen such that $f_q(x_{i+1}) = f(x_{i+1}) = f[i+1]$ and one additional constraint such as matching function values or the first derivative of $f_q(x)$ and $f(x)$ at some point x in the interval $[x_i, x_{i+1})$.

Quadratic interpolation will yield smaller errors than linear interpolation but will require additional multiplications and additions which will slow down computation. The speed penalty can be avoided by using extra adders and multipliers but this will cost gates. Moreover, tables for $A[i]$, $B[i]$, and $C[i]$ may have to be stored, thus increasing the storage. If for example the first derivative of $f_q(x)$ is matched with the first derivative of $f(x)$ at the left end point, x_i, then $B[i] = -A[i]x_{i+1}$, allowing the number of tables to be reduced from three to two.

The Newton-Raphson method can also be used. Let x be the input, let $R_0(x) = 1/x_0$ be the initial estimate, and let $f(x) = 1/x$. Then by Taylor's theorem

$$\begin{aligned} 1/x &= 1/x_0 + f'(x_0)(x - x_0) + \frac{1}{2!}f''(x_0)(x - x_0)^2 + \ldots.. \\ &= R_0(x) + f'(1/R_0(x))(x - 1/R_0(x)) + \frac{1}{2!}f''(1/R_0(x))(x - 1/R_0(x))^2 + \ldots.. \end{aligned}$$

So using only the first two terms

$$\begin{aligned} 1/x &\approx R_0(x) - R_0(x)^2(x - 1/R_0(x)) \\ R_1(x) &= 2R_0(x) - R_0(x)^2 x = R_0(x)(2 - R_0(x)x) \\ R_2(x) &= 2R_1(x) - R_1(x)^2 x = R_1(x)(2 - R_1(x)x) \end{aligned}$$

This method takes 2 *fmas* (floating point multiply adds) per iteration and the time for the initial look-up. The drawback is that each iteration required additional cycles and/or gates. Several processors already use this algorithm as part of floating point division calculations.

Figures 8 and 9 show the relative errors with Newton's method. Figure 8 shows the relative error when a single iteration is used. Figure 9 shows the relative error when a two iterations are used. With two iterations the error is quite small.

With the first three terms from the Taylor series, the following results:

$$\begin{aligned} R_1(x) &= 3R_0(x) - 3R_0(x)^2 x + R_0(x)^3 x = R_0(x)(3(1 - R_0(x)x) + R_0(x)^2 x^2) \\ R_2(x) &= 3R_1(x) - 3R_1(x)^2 x + R_1(x)^3 x = R_1(x)(3(1 - R_1(x)x) + R_1(x)^2 x^2) \end{aligned}$$

The above can be implemented with 5 *fmas* per iteration and the time for the initial look-up.

Signal processing chips from Weitek, the PowerPC 604, and the Intel i860, etc., expose the reciprocal seed to software which can then be used to compute a more accurate result.

Newton's method is especially simple for $1/x$, but it still requires two multiplies. This works well, but is more costly than linear interpolation (which needs only one multiply). Newton's method is best suited for getting high precision. For the relatively low level of precision required here (3 to 4 digits), linear interpolation is more economical. The ultimate choice depends on the accuracy requirements and the technology in which the algorithm is implemented and the ratio of memory units to computation units. Thus different algorithms may also be used for different ranges of x, as we show in the following section.

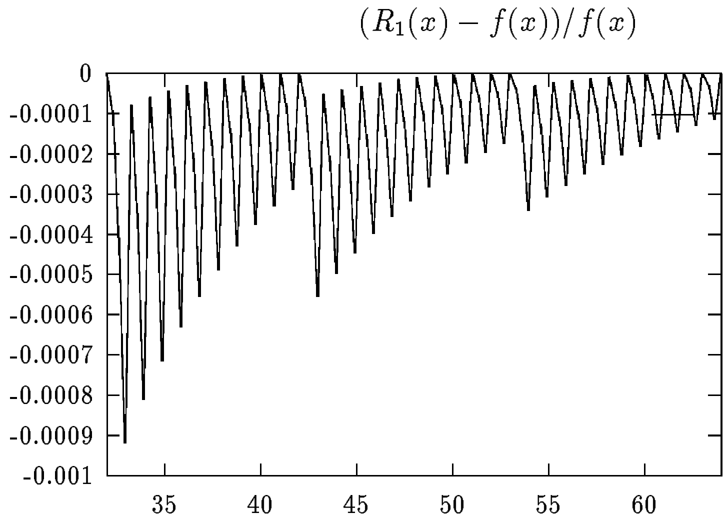

Figure 8: Maximum relative errors with one iteration of Newton's method in the range [32,64]. There is a table entry for each x in the range [32,64].

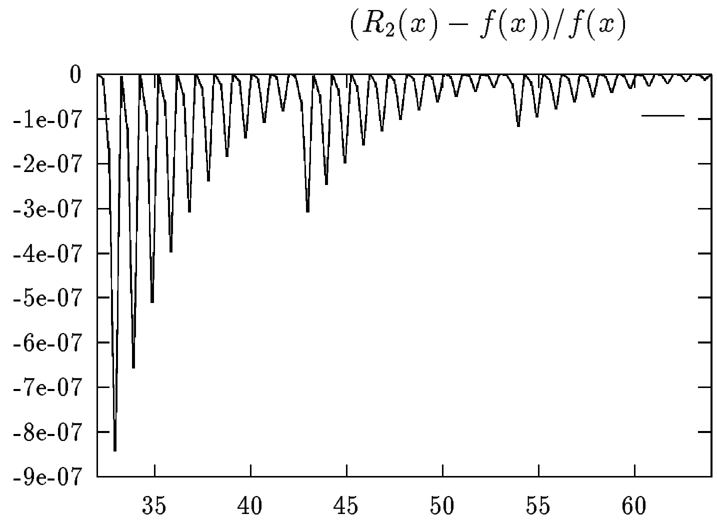

Figure 9: Maximum relative errors with two iterations of Newton's method in the range [32,64]. There is a table entry for each x in the range [32,64].

5 Combining algorithms

In this section we consider in detail a technique that is a combination of simple look-up, linear approximation, and remapping input x ranges. In addition the number of bits in each entry in the table and the number of entries in the table required for correct texture mapping are considered.

The formulae derived in Section 3 show that the maximum error can be kept small by using an r close to 1. The maximum error itself drops off rapidly when x is large. Thus a table with an r close to one should do for x in mid to high ranges. For values of x close to zero, the table can store the actual value of the quotient so that a simple look-up can be performed.

Therefore it is clear that for large values of x it should be possible to use the result for a lower value of x and scale the result with a small loss in overall relative accuracy. Let us say that we store table values for input values of $x = x_L, x_L+1, x_L+2, \cdots, 2x_L-1 = x_H - 1$.

We will use the following definitions in the rest of the section. We will assume that x_L and x_H are powers of two.

$$
\begin{aligned}
X_M &\equiv \text{maximum value of x} \\
nbits_T &\equiv \text{number bits in each table entry} \\
nbits_L &\equiv \log_2(x_L) \\
nbits_H &\equiv \log_2(x_H) = nbits_L + 1 \\
nbits_{X_M} &= \lfloor 1 + \log_2(X_M) \rfloor \\
nbits_K &= \lfloor 1 + \log_2(K) \rfloor \\
nbits_R &= \text{number bits in result} K/x \\
nbits_E &= \log_2(K/X_M) = nbits_K - nbits_{X_M} \\
nbits_x &= \lfloor 1 + \log_2(x) \rfloor
\end{aligned}
$$

For integer values of x such that $x_L \leq x \leq x_H - 1$, a simple look-up can be used. These results will be error free since corresponding table entries for these divisors store exact quotients.

For integer values of x such that $1 \leq x < x_L$ the following will be used.

$$
\begin{aligned}
kbits &= nbits_H - nbits_x \\
x_{norm} &= x \times 2^{kbits} \\
f_{out} &= \left(\frac{K}{x_{norm}} \right) \times 2^{kbits}
\end{aligned}
$$

In the case where $x > x_H - 1$:

$$
\begin{aligned}
kbits &= nbits_x - nbits_H \\
x_{norm} &= \left\lfloor \frac{x}{2^{kbits}} \right\rfloor \\
\delta &= x - (x_{norm} \times 2^{kbits}) \qquad\qquad (0 \leq \delta < 2^{kbits}) \\
f1 &= \frac{K}{x_{norm}}
\end{aligned}
$$

$$
\begin{aligned}
f2 &= \frac{K}{x_{norm}+1} \\
f_{out} &= \frac{f1 - \frac{\delta(f1-f2)}{2^{kbits}}}{2^{kbits}} \\
&= \frac{2^{kbits} \times f1 - \delta(f1-f2)}{2^{2\times kbits}}
\end{aligned}
$$

The following examples are based on a 32 entry table, (i.e., $x_L = 32$, $x_H = 64$, $nbits_L = 5$, $nbits_H = 6$) with values of K/x. The ith entry in the table stores $K/(i+32)$, where i ranges from 0 to 31. Then, given an integer x, it is normalized to lie in the range [32,63] by either selecting the most significant 6 bits of x if $x > 63$ or by left shifting x by the right number of bits if $x < 32$. $x = 0$ is not a valid input for the division algorithm and is treated as a special case. Then linear interpolation is used on the normalized value and the result is de-normalized. Using this method, for example $K/15$ is evaluated as $(K/60) \times 4$. $K/33$ is evaluated as $(K/33)$. $K/69$ is evaluated as $((K/35 - K/34) \times 1/2 + K/34)/2$. $K/130$ is evaluated as $((K/33 - K/32) \times 2/4 + K/32)/4$.
Consider the size of the look-up table. There are two things to examine:

1. the number of entries in the table, and
2. the number of bits in each entry in the table.

Table 1 shows what the worst-case error can be limited to by the linear interpolation technique. This essentially determines the number of entries in the table.
Let us now determine $nbits_T$, the number of bits required for each table entry to achieve the desired relative accuracy of say one part in 2^A. Clearly, for a given $nbits_T$, the maximum value of K should be such that $2^{nbits_T} - 1 \geq K/x_L + 0.5$, i.e., $K \leq x_L \times (2^{nbits_T} - 1.5)$. Therefore the maximum value for $nbits_K$ is $nbits_T + nbits_L$. There are two cases to consider, $x > x_H - 1$ and $x < x_L$. Clearly, $nbits_T$ should satisfy both cases.

Case 1, $\mathbf{x > x_H - 1}$: The smallest number of bits in the result of K/x will be (when $x = X_M$) $nbits_K - nbits_{X_M} \equiv nbits_E = nbits_T + nbits_L - nbits_{X_M}$. In this case, $kbits$ of precision are lost in the computation of f_{out}.

Case a, $\mathbf{nbits_R < A}$: In this case, the number of bits in the result is smaller than the desired accuracy, and so if there is any erroneous bit in the result the error will be greater than the maximum allowed error. There is no way to guarantee that the result has no erroneous bits using the said computations. In order to prevent this condition the minimum result should have $A+1$ bits, i.e., $K/X_M \geq 2^A$ or $nbits_K > A + nbits_{X_M}$. Therefore, the number of bits in each table entry should satisfy, $nbits_T > A + nbits_{X_M} - nbits_L$.

Case b, $\mathbf{nbits_R \geq A}$: In this case errors due to linear approximation and also the round-off error in a table with a finite number of bits will exist. As $nbits_T$ increases the round-off error in the table becomes smaller and therefore the errors become smaller in general.

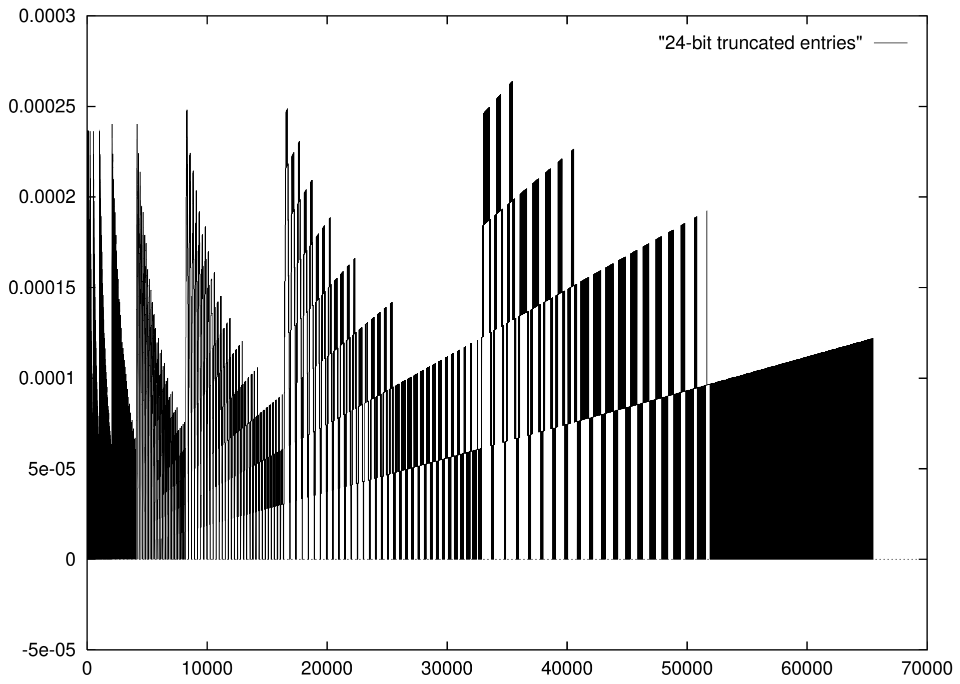

Figure 10: Relative errors with adaptive subdivision using 24-bit table entries. The table entries and the final results are truncated.

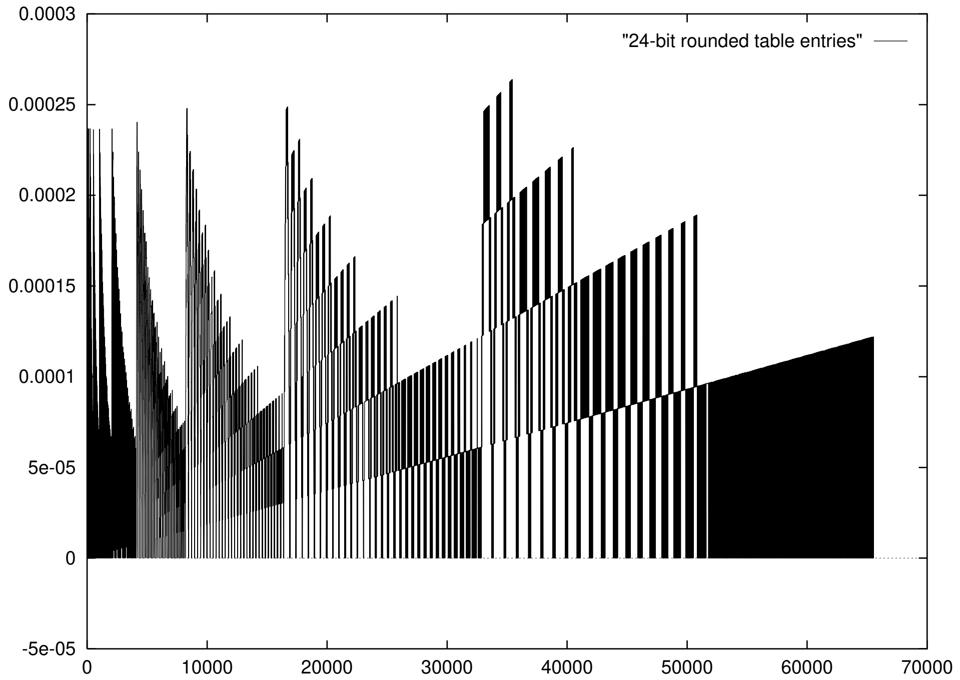

Figure 11: Relative errors with adaptive subdivision using 24-bit table entries. The table entries and the final results are rounded.

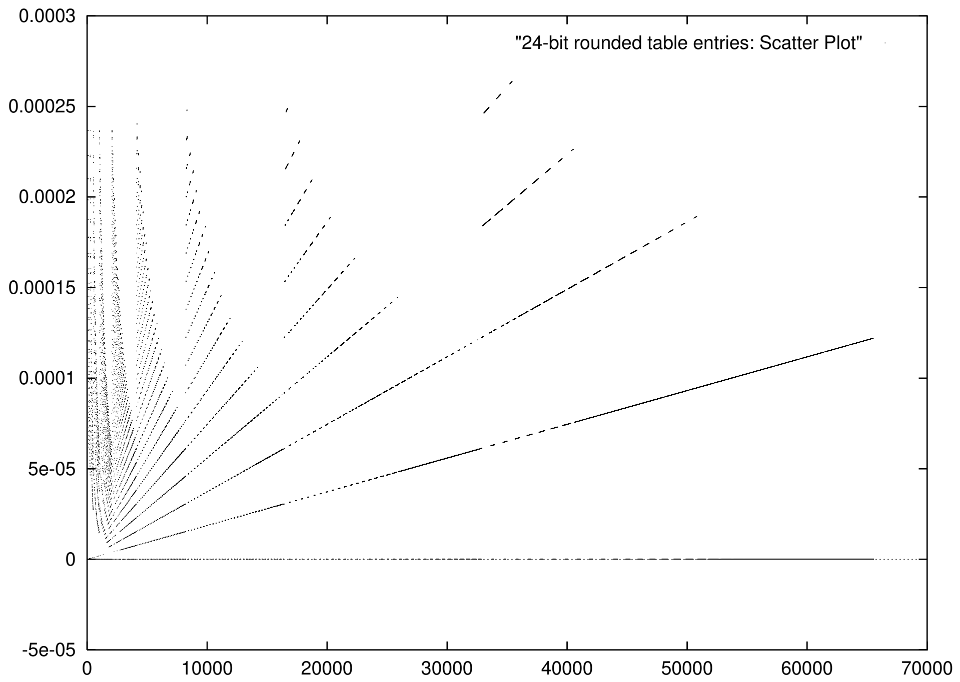

Figure 12: Relative errors with 24-bit table entries. The table entries and the final results are rounded.

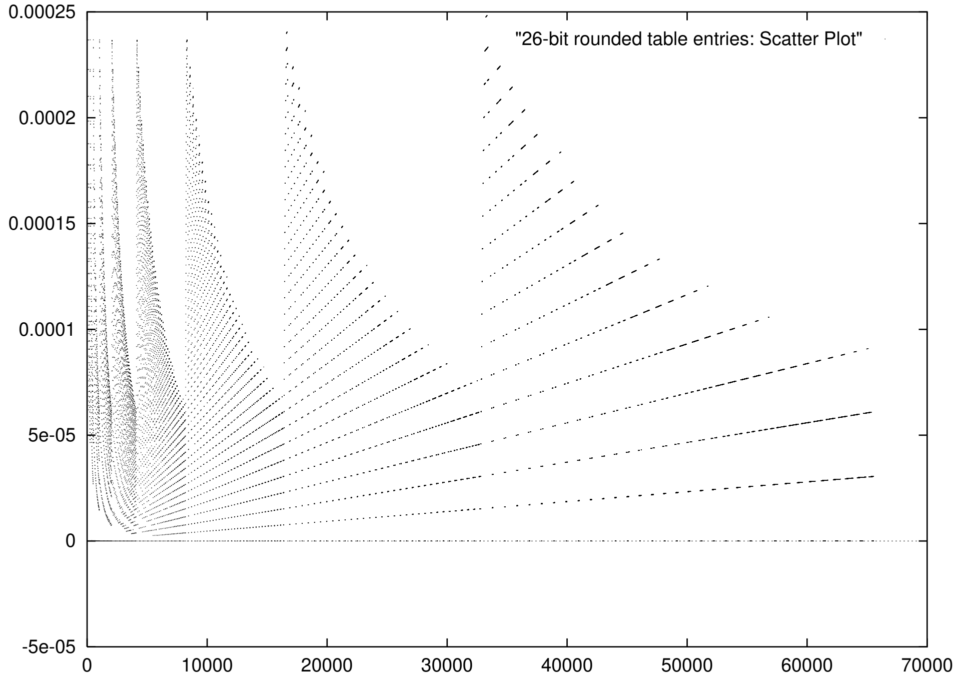

Figure 13: Relative errors with 26 bit table entries. The table entries and the final results are rounded.

Increasing $nbits_T$ beyond the required minimum reduces the frequency of higher errors as can be seen by comparing Figures 12 and 13. This is because there are more bits in the quantities used to linearly interpolate between. However, it cannot reduce the maximum error produced because that is fixed by the number of entries in the table.

Case 2, $\mathbf{x < x_L}$: In this case the errors in the table get amplified by 2^{kbits}, i.e., $kbits$ bits are wrong. To have an eventual relative precision of one part in 2^A, $nbits_R - A$ bits can be thrown away without noticing it. There are now two cases to consider:

Case a, $\mathbf{nbits_R < A}$: In this case no bits can be thrown away and so there is no way to extract the result with the desired minimum relative accuracy. For this case to occur $x > K/2^A$, i.e., $x > (2^{nbits_T + nbits_L} - 1)/2^A$ and $x < 2^{nbits_L}$. This happens only if $nbits_T < A$. Therefore to avoid this case completely, $nbits_T$ should be chosen such that $nbits_T \geq A$.

Case b, $\mathbf{nbits_R \geq A}$: The number of bits in the maximum result of $K/x = K/1$ is $nbits_R = nbits_K = nbits_T + nbits_L$. The maximum value of $kbits$ is $nbits_L$. To satisfy the minimum relative accuracy requirement, $nbits_T + nbits +_L - A > nbits_L$, i.e., $nbits_T > A$.

Increasing $nbits_T$ beyond the required minimum decreases the maximum error in the range $1 \leq x < x_L$. Since the requirement on $nbits_T$, for the range $x_H \leq x \leq X_M$ is more stringent than on the range $1 \leq x < x_L$ we get better than one part in 2^A relative precision in the result in the range $1 \leq x < x_L$.

To satisfy all the above conditions $nbits_T > A + nbits_{X_M} - nbits_L$. For $A = 12$, $nbits_{X_M} = 16$ and $nbits_L = 5$, $nbits_T \geq 24$. The total number of bits in the table will be $32 \times 24 = 768$.

Figures 10-13 show the relative errors using the above technique when a table with 32 entries is used. Figure 11 shows results when table entries are rounded and Figure 10 shows results when table entries are truncated. Figures 12 and 13 show that as the number of bits in each table entry is increased, the frequency with which high errors occurs reduces, thus reducing the average error.

It is instructive to compare the relative errors with this method as shown in Figures 12 and 13 and that due to Newton's method with one iteration as shown in Figure 8. This comparison is shown more clearly in Figure 14. The maximum relative error is only *one quarter* of that due to Newton's method with one iteration.

If a higher level of accuracy is needed then a similar table having 64 entries can be used. Table 1 shows the best upper bounds on errors with a 64 entry table. In this case the maximum relative error is 6.0096×10^{-5}, which is quite small for purposes of texture mapping. The ith entry in the table stores $K/(i + 64)$, where i ranges from 0 to 63. For $A = 14$, $nbits_{X_M} = 16$ and $nbits_L = 6$, $nbits_T \geq 25$. The total number of bits in the table will be $64 \times 25 = 1600$.

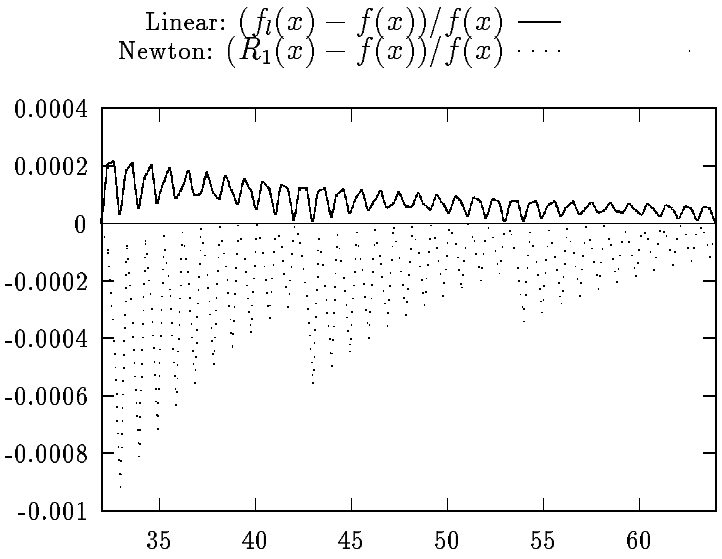

Figure 14: Comparison of maximum relative errors with Newton's method and linear interpolation in the range [32,64]. There is a table entry for each x in the range [32,64].

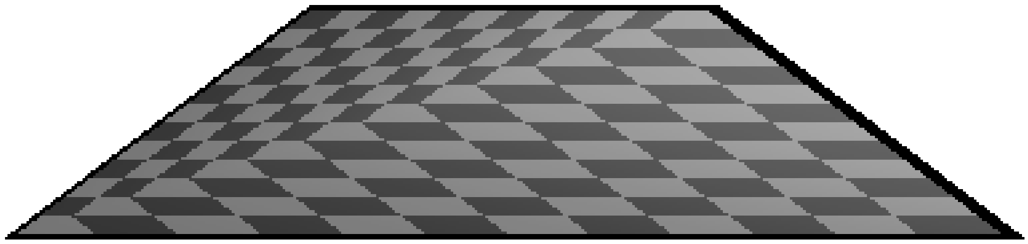

Figure 15: Texture mapped image with no perspective correction.

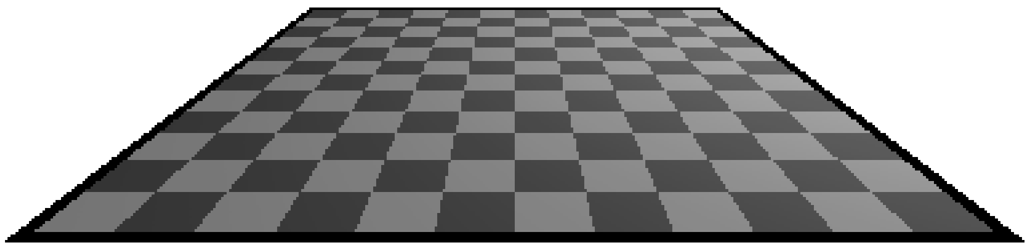

Figure 16: Texture mapped image with approximate division

6 Look-up table implementation

The amount of memory required to store a look-up table can be further reduced through the use of logic minimization and compression techniques. While this may not be crucial for a 32 entry table where each entry is 24 bits, it may be considered when larger tables are used.
For example, a hexadecimal representation of a look-up table for $1/x$ composed of 32 entries of 24 bits each is shown below.

```
a[0] = ffffff     a[8] = cccccc    a[24] = 924924    a[16] = aaaaaa
a[1] = f83e0f     a[9] = c7ce0c    a[25] = 8fb823    a[17] = a72f05
a[2] = f0f0f0    a[10] = c30c30    a[26] = 8d3dcb    a[18] = a3d70a
a[3] = ea0ea0    a[11] = be82fa    a[27] = 8ad8f2    a[19] = a0a0a0
a[4] = e38e38    a[12] = ba2e8b    a[28] = 888888    a[20] = 9d89d8
a[5] = dd67c8    a[13] = b60b60    a[29] = 864b8a    a[21] = 9a90e7
a[6] = d79435    a[14] = b21642    a[30] = 842108    a[22] = 97b425
a[7] = d20d20    a[15] = ae4c41    a[31] = 820820    a[23] = 94f209
```

A logic minimizer such as *espresso* can automatically identify redundancies in the above table and minimize the number of gates required. A look at the above table show redundancies of the following form:

- All digits in some entries are the same such as in a[0], a[8], a[16], and a[28]. These 4 entries can be encoded more compactly.
- Repeating digits. The 3 LSBs and the 3 MSBs of each of the entries, a[2], a[3], a[4], a[7], a[10], a[13], a[19], a[20], a[24], a[31] are identical. These 10 entries can be compacted by a factor of two.

As usual, whenever compression is considered, the cost of logic for decompression should always be weighed against the savings in storage.

7 Experimental Verification

The above algorithm has been implemented and tested. In the following figures a checkerboard texture has been mapped onto a square composed of two triangles.

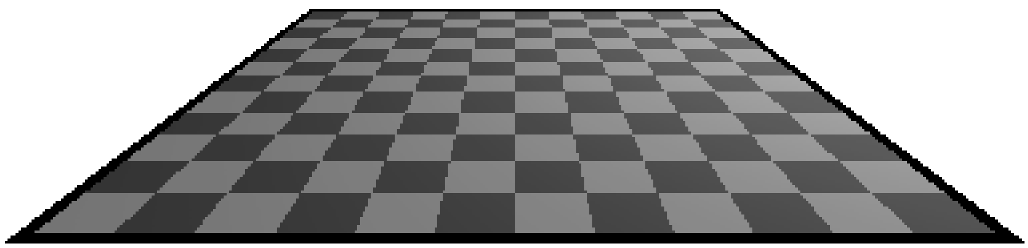

Figure 17: Texture mapped image with exact division

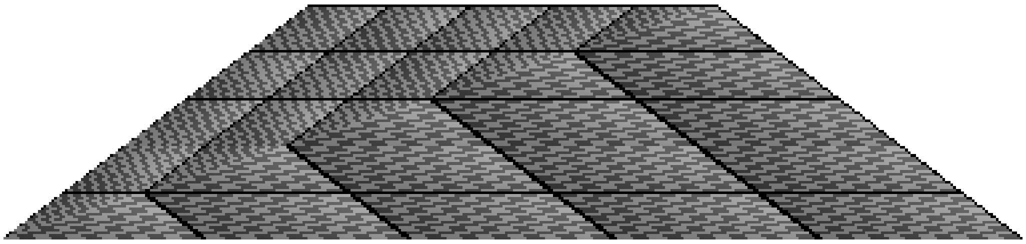

Figure 18: Texture mapped image with no perspective correction.

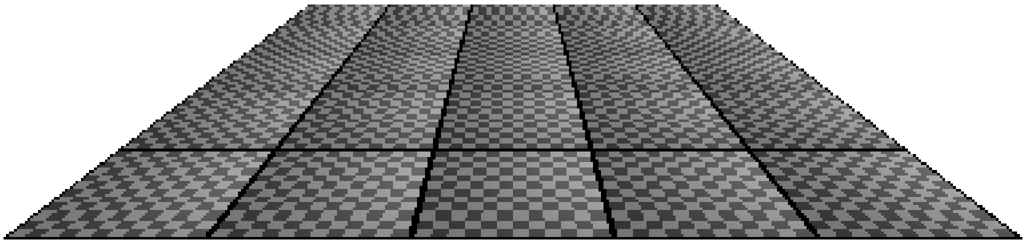

Figure 19: Texture mapped image with approximate division

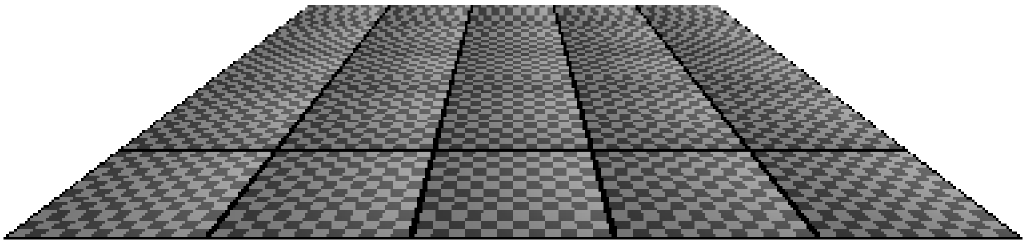

Figure 20: Texture mapped image with exact division

Figures 15-17 are based on a 512×512 image of a 12×12 checkerboard with a black border. The same texture is repeated in a 5×5 grid in Figures 18-20. Only point sampling has been used so any artifacts have not been smoothed out as would have been the case if some filtering had been performed. Figures 15 and 18 show the results when no perspective correction is applied. Figures 16 and 19 show the results when perspective correction is applied with approximate division using the look-up table. Figures 17 and 20 show the results when perspective correction is applied with exact division using floating point arithmetic. It is clear from the above figures that the approximate division works very well and is very close to exact division and a very major improvement over the case where no perspective correction is applied.

8 Conclusion

This report has shown that piecewise linear approximation to the function $1/x$ can be quite practical. In particular, a table with 32 24-bit entries can yield results which have a relative error of less than 2.5×10^{-4}. This error was acceptable for the texture mapping application. Higher accuracies may be obtained by using either quadratic interpolation or the Newton's method.

Most of the hardware required for the algorithm is simple adders, subtracters, shifters and a multiplier. The most complex parts are the multiplier and the circuit that determines the leading 1 in x. Consequently this algorithm is easily implementable in hardware.

References

[1] Heckbert, P. S. Survey of texture mapping. *IEEE Computer Graphics and Applications 6*, 11 (Nov. 1986), 56–67.

[2] Narayanaswami, C., and Luken, W. L. Approximating x^n Efficiently. *Information Processing Letters 50*, 4 (April 1994), 205–210.