An Exponentiation Unit for an OpenGL Lighting Engine

David Harris, *Member*, *IEEE*

Abstract—The OpenGL geometry pipeline lighting stage requires raising a number in the range $[0, 1]$ to a power between $[1, 128]$ to compute specular reflections and spotlights. The result need only be accurate to a number of bits related to the color depth of the output device. This paper describes a hardware implementation of such an exponentiation unit based on a logarithm lookup table, a multiplier, and an inverse log table. The inputs arrive in IEEE single-precision floating-point format and the output is a floating-point color component in the range $[0, 1]$ with 8-10 bits of accuracy. The log lookup table is partitioned into subintervals to reduce table size and each subinterval is computed from a bipartite table to further reduce size. A synthesized design uses 32k gates to achieve 10-bit accuracy with a latency of 9.4 ns in a 180 nm process. Although the system is tailored to the OpenGL application, the same principles can be applied to the design of other exponentiation units.

Index Terms—Powering, exponentiation, computer arithmetic, OpenGL hardware acceleration, table lookups, table complexity, bipartite tables.

1 Introduction

OPENGL is a standard for professional 3D graphics [1], [2]. The OpenGL pipeline consists of floating-point intensive transformation and lighting followed by short integer computations for rasterization. Hardware graphics accelerators have traditionally focused on the rasterization stages, but have become so fast that transformation and lighting are now a bottleneck [3]. Therefore, the transformation and lighting calculations have moved from the host processor to a hardware transform and lighting engine on a mainstream PC-graphics accelerator, such as the NVIDIA GeForce4 and ATI RADEON 9700.

The transform and lighting engine accepts vertices and normal vectors and performs matrix multiplies for coordinate system transformations. It then calculates ambient, emissive, diffuse, and specular lighting. Specular lighting results in highlights when light comes from a particular direction and reflects off the surface. Lights may be specified as spotlights that also favor a particular direction. Both specular lighting and directed spotlight calculations involve raising the cosine of an angle to a power to determine the light intensity reaching the viewer. Specifically, the OpenGL pipeline must raise a number in the range $[0, 1]$ to a (possibly noninteger) power in the range $[0, 128]$. In practice the power is usually in the range $[1, 128]$ and this design is restricted to that range for ease of hardware implementation. Such a limit is consistent with the philosophy of accelerating the common OpenGL modes and trapping to software for other modes. The inputs and outputs of the pipeline are commonly represented as single-precision IEEE floating-point numbers [1], [4]. The exponentiation operation is required for each vertex for Gouraud shading and for every pixel for the more realistic but computationally expensive Phong shading [5]. Ideally, all computations would be performed with as high a precision as possible until a final rounding stage that determines the vertex color. In practice, the precision may be reduced to save hardware without introducing perceptible artifacts.

Accurately computing A^B is considered a difficult floating-point operation [6] because the function is ill conditioned, especially for large values of B [7]. Approximations for specific cases can be efficient. For example, Tang [8], [9] describes an algorithm for $\exp(B)$ using range reduction, a polynomial approximation, and reconstruction. Efficient approaches involving table lookup and interpolation exist when B is a constant [10], [11]. Software math libraries [12] often rely on multiplications, particularly when B is an integer.

The hardware implementations of commercial graphics accelerators such as the SGI Infinite Reality Engine or the NVIDIA or ATI chips have been closely guarded trade secrets. However, a number of academic designs have been published. These designs use the fact that the absolute accuracy requirements are set by the color depth of the output device and the acuity of human vision. Shin et al. [13] use a direct ROM lookup based on A and B. While a full ROM would be very expensive (e.g., 8 Mbits), they use nonuniform quantization of the inputs to maintain visual fidelity with only a 128kb ROM. However, their approach does not handle arbitrary noninteger values of B, is prone to banding from the quantization, and offers little error analysis. Another technique is to use the identity

$$A^B = 2^{B \log_2 A} \tag{1}$$

to compute the result using a logarithm, multiply, and exponent. Chen and Lee [14] perform the logarithm and exponent with table lookups, while Kwon et al. [15] use clever piecewise linear approximations that avoid the need

• *The author is with Harvey Mudd College, 301 E. Twelfth St., Claremont, CA 91711. E-mail: David_Harris@hmc.edu.*

Manuscript received 14 Mar. 2003; revised 13 aug. 2003; accepted 30 Sept. 2003.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 118453.

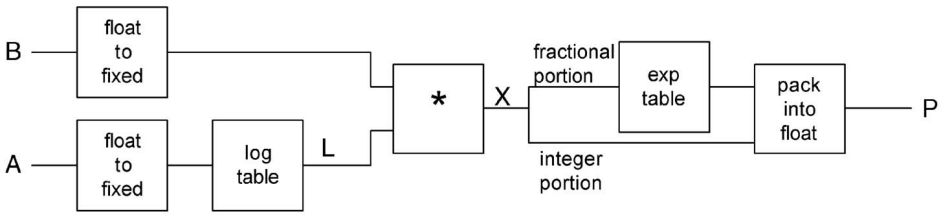

Fig. 1. Block diagram of exponentiation unit.

for tables and are accurate over most but not all of the input domain.

This paper describes an algorithm and hardware implementation for calculating $P = A^B$. A and B are provided at arbitrarily high precision with $A \in [0, 1]$, $B \in [1, 2^b]$. $P \in [0, 1]$ is expressed as a fixed-point number faithfully rounded to p fractional bits. In other words, the result has an absolute error of less than 2^{-p} [18]. As in [14], the algorithm is based on (1) with logarithm and exponential lookup tables. For OpenGL, b is 7 and p is typically 8 or 10, corresponding to the number of bits used to represent each red, green, blue, and alpha color component.

The hardware cost depends on the size of the lookup tables required to produce a p-bit result. If a single logarithm lookup table were used, we will find the number of entries grows as $O(2^{b+p})$ and can be impractically large. Alternatively, Chen and Lee [14] use a medium-sized $(4,096 \times 9)$ table and accept significant errors for large values of B. A key idea of this paper is to partition the logarithm table into multiple tables over subintervals, as done by Coleman et al. [16]. This leads to $O(b)$ tables that grow as $O(2^p)$ for an area of $O(b2^p)$. The logarithm table size can be reduced even further using table-lookup-and-addition methods [17], [18], [19], [20], [21] in place of a conventional table lookup.

This paper presents the algorithm and error analysis that determines the size of the lookup tables. The design was coded in Verilog and synthesized to produce area and timing results.

2 Algorithm

We wish to compute $P = A^B$, where A and B are floating-point numbers in the domains $[0, 1]$ and $[1, 2^b]$, respectively. The result is faithfully rounded to p fractional bits. This means that the returned result is guaranteed to be one of the two fixed-point numbers that surround the exact result. Faithful rounding is more practical than exact rounding for a exponentiation unit because it is expensive to calculate the result to the high level of precision required to round exactly [6].

Fig. 1 shows the steps of the computation. We begin with the identity $A^B = 2^{B \log_2 A}$. The logarithm, L, is selected from a table using a fixed-point representation of A, then multiplied by a fixed-point representation of B to obtain the product $X \approx B \log_2 A$. We finally determine the result $P = 2^X$ as a floating-point number. The exponent is the integer portion of X and the significand is looked up from a table based on the fractional part of X.

In this paper's notation, let the bit vector $x[m : n]$ represent $\sum_{i=n}^{m} x[i]2^i$. Let $p' = \lceil \log_2(p + 1) \rceil$, which is the number of bits required to represent the integer portion of the logarithm of the smallest $A = 2^{-p}$ that will not generate a result of 0. Let $\hat{A}[-1 : -n_1]$ be a fixed-point representation of A truncated to n_1 fractional bits. $\hat{B}[b : -n_2]$ is the corresponding fixed-point representation of B with $b + 1$ integer bits and n_2 fractional bits. One can readily convert from floating-point inputs into these fixed-point representations by a shift and truncation of the signficands. The result P will be a floating-point number in the range $[0, 1]$ faithfully rounded to within one part in 2^{-p}.

Fig. 2 lists the exponentiation algorithm. It first handles special cases of large and small inputs. Note that only p' bits of integer part must be maintained in the log lookup and multiplication because, if the integer portion exceeds this range, the final result will be 0. The number of bits n_1, n_2, n_3, and n_4 for each intermediate result determines the sizes of the tables and the multiplier. In the next section, we will perform an error analysis to find the smallest numbers of bits required to achieve a particular accuracy.

As a concrete illustration of the algorithm, consider a small exponent unit handling values of B in the range of $[0, 4]$ returning a result faithfully rounded to four bits. In this case $b = 2$, $p = 4$, and $p' = 3$. The subsequent error analysis will show the intermediate computations must maintain $n_1 = 7$, $n_2 = 7$, $n_3 = 9$, and $n_4 = 6$ bits if a straightforward logarithm lookup table is employed. Now, consider the steps of computing $0.97^{3.5} = 0.898879$. In binary, $A = 0.11111000010100\ldots_2$, $B = 11.10_2$, and P = $A^B = 0.11100110000111\ldots_2$. The intermediate results are:

$$\hat{A} = 0.1111100_2$$
$$\hat{B} = 11.1000000_2$$
$$L = -\log_2(0.11111001_2) = 0.04_{10} = 0.000010100_2$$
$$X = L\hat{B} = 000.001000_2$$
$$E = 2^{-(0.0010001)_2} = 0.912052\ldots_{10} = 0.1110_2$$
$$P = 0.1110_2 \times 2^{-000_2} = 0.1110_2 = 0.875_{10}.$$

Thus, the answer is faithfully rounded to four bits.

3 Error Analysis

To guarantee a faithfully rounded result, we must consider the sources of error introduced by the finite precision lookup tables and multiplier. Given these sources, we determine the necessary table sizes.

The logarithm table ideally returns the logarithm of A, but, because it is indexed with only n_1 bits in $\hat{A}[-1 : -n_1]$,

```
if A = 1.0 then P = 1.0                                  # (1.0)^b = 1.0
else if Â[-1:-n1] < 2^-(p+1) then P = 0.0                # input is tiny
else begin
   L[p'-1:-n3] = -log2(Â[-1:-n1] + 2^-(n1+1))            # 2^n1 -entry log lookup table
   X[p'-1:-n4] = L[p'-1:-n3] • B̂[b:-n2]                  # (p'+n3) x (b+n2+1) bit multiplier
   if X[p'-1:0] > p then P = 0.0                         # result will be negligible
   else begin
      E[-1:-p] = 2^-(X[-1:-n4]+2^-(n4+1))                # 2^n4 -entry exponent lookup table
      P_exponent = -X[p'-1:0]
      P_significand = E[-1:-p]                           # conversion to floating point
   end
end
```

Fig. 2. Exponentiation algorithm.

some round-off error is introduced. Let the table entry for $\hat{A}$ contain $-\log_2(\hat{A}[-1:-n_1]+2^{-(n_1+1)})$. The round-off error ε_1 is thus at most half of a unit in the least significant position.

$$|\varepsilon_1| = \left|A - \left(\hat{A}[-1:-n_1] + 2^{-(n_1+1)}\right)\right| \le 2^{-(n_1+1)}. \quad (2)$$

Fig. 3 illustrates the variation of the round-off error with A for $n_1 = 3$.

The logarithm table produces a result L rounded to the nearest fixed-point number with n_3 fractional bits. This introduces another error representing the difference between the exact logarithm and the table contents.

$$|\varepsilon_3| = \left|-\log_2\left(\hat{A} + 2^{-(n_1+1)}\right) - L[p'-1:-n_3]\right| \le 2^{-(n_3+1)}. \quad (3)$$

The B input to the multiplier is truncated to $\hat{B}[b:-n_2]$ with n_2 fractional bits, so

$$|\varepsilon_2| = \left|B - \hat{B}[b:-n_2]\right| < 2^{-n_2}. \quad (4)$$

The product $X[p'-1:-n_4]$ is truncated to n_4 bits before being used in the exponent lookup table. As in the logarithm table,

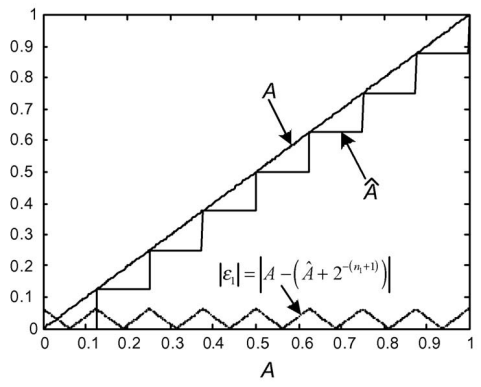

Fig. 3. Round-off error in index to logarithm table.

$$|\varepsilon_4| = \left|L \cdot \hat{B} - \left(X[p'-1:-n_4] + 2^{-(n_4+1)}\right)\right| \le 2^{-(n_4+1)}. \quad (5)$$

Finally, the exponent table produces a result rounded to the nearest fixed-point number with p fractional bits, introducing a further error.

$$|\varepsilon_5| = \left|2^{-\left(X[-1:-n_4]+2^{-(n_4+1)}\right)} - E[-1:-p]\right| \le 2^{-(p+1)}. \quad (6)$$

Considering all these errors, we actually compute

$$\hat{P} = 2^{(B+\varepsilon_2)(\log_2(A+\varepsilon_1)+\varepsilon_3)+\varepsilon_4} + \varepsilon_5. \quad (7)$$

For faithful rounding, we must choose tables large enough that the error is small enough: $|\hat{P} - P| < 2^{-p}$.

$$\left|2^{(B+\varepsilon_2)(\log_2(A+\varepsilon_1)+\varepsilon_3)+\varepsilon_4} + \varepsilon_5 - 2^{B\log_2 A}\right| < 2^{-p}. \quad (8)$$

Because the errors are small, we will analyze them using first-order Taylor series approximations for $log_2 x$ and 2^x

$$\log_2(x+\varepsilon) \approx \log_2 x + \frac{\varepsilon}{x \ln 2} \quad (9)$$

$$2^{x+\varepsilon} \approx 2^x(1 + \varepsilon \ln 2). \quad (10)$$

Substituting (9) into (8) and eliminating quadratic error terms, we find

$$\left|2^{B\left(\log_2 A + \frac{\varepsilon_1}{A\ln 2} + \varepsilon_3\right) + \varepsilon_2 \log_2 A + \varepsilon_4} + \varepsilon_5 - 2^{B\log_2 A}\right| < 2^{-p}. \quad (11)$$

Then, substituting (10) into (11) and simplifying, we find our error bound

$$\left|A^B\left(\varepsilon_1 \frac{B}{A} + \varepsilon_2 \log_2 A \ln 2 + \varepsilon_3 B \ln 2 + \varepsilon_4 \ln 2\right) + \varepsilon_5\right| < 2^{-p}. \quad (12)$$

This bound depends on the input A; for small values of A, we can place looser constraints on the errors than when A is close to 1. This suggests that we could benefit from partitioning the logarithm table into subintervals with greater precision for inputs close to unity. We will choose upper bounds on ε_1, ε_2, ε_3, ε_4, and ε_5 to ensure that (12) is satisfied for a desired accuracy p. The bounds on the

logarithm table errors ε_1 and ε_3 will depend on the value of A. The other bounds will be independent of A and B.

To find bounds on ε_1 and ε_3 we take a derivative of their terms with respect to B to find the maximum value each term can take on for a given value of A. For both errors, this maximum occurs at

$$B = \begin{cases} 1 & A < e^{-1} \\ -1/\ln A & \text{otherwise} \\ 2^b & A > e^{-2^{-b}}. \end{cases} \tag{13}$$

Also observe that the weight on the ε_2 term takes on a maximum value at $A = 1/e$, $B = 1$ of

$$\left|A^B \log_2 A \ln 2\right| \le \frac{1}{e}. \tag{14}$$

We can now eliminate B from (12) by substituting (13) and (14) and taking an upper bound of 1 for A^B. The error bound depends on the interval containing A.

$$\left.\begin{array}{l} \left|\varepsilon_1 + \frac{\varepsilon_2}{e} + \varepsilon_3 A \ln 2 + \varepsilon_4 \ln 2 + \varepsilon_5\right| \\ \left|\frac{-\varepsilon_1}{eA\ln A} + \frac{\varepsilon_2}{e} + \frac{-\varepsilon_3 \ln 2}{e \ln A} + \varepsilon_4 \ln 2 + \varepsilon_5\right| \\ \left|\varepsilon_1 2^b + \frac{\varepsilon_2}{e} + \varepsilon_3 2^b \ln 2 + \varepsilon_4 \ln 2 + \varepsilon_5\right| \end{array}\right\} < 2^{-p} \begin{array}{ll} & A < e^{-1} \\ \text{otherwise} & \\ & A > e^{-2^{-b}}. \end{array} \tag{15}$$

We already bounded ε_5 in (6). Reducing the errors ε_1 and ε_4 is costly because these determine the number of entries in the logarithm and exponent lookup tables. We will allocate the total allowable error of 2^{-p} among the five terms to satisfy (15) while giving as much slop as possible for ε_1 and ε_4 to minimize table sizes.

$$\left.\begin{array}{l} |\varepsilon_1| \\ \frac{-1}{eA\ln A}|\varepsilon_1| \\ 2^b|\varepsilon_1| \end{array}\right\} < 2^{-(p+2)} \begin{array}{ll} & A < e^{-1} \\ \text{otherwise} & \\ & A > e^{-2^{-b}} \end{array}$$

$$\frac{|\varepsilon_2|}{e} < 2^{-(p+4)}$$

$$\left.\begin{array}{l} A \ln 2|\varepsilon_3| \\ \frac{-\ln 2}{e \ln A}|\varepsilon_3| \\ 2^b \ln 2|\varepsilon_3| \end{array}\right\} < 2^{-(p+4)} \begin{array}{ll} & A < e^{-1} \\ \text{otherwise} & \\ & A > e^{-2^{-b}} \end{array} \tag{16)-(19}$$

$$\ln 2|\varepsilon_4| < 2^{-(p+3)}.$$

Given these bounds, we can determine the number of bits required at each step in the computation. Taking $\frac{1}{e} < 2^{-1}$ and $\ln 2 < 2^0$, we solve (4) and (17) for $n_2 = p + 3$ and (5) and (19) for $n_4 = p + 2$. We will choose n_1 and n_3 in the next section based on our logarithm table design.

4 Table Design

Fig. 4 plots the weight w multiplying the ε_1 term in (16) for $b = 7$, showing that it increases from 1 for small inputs A to $2^b = 128$ as A gets very close to 1. As the magnitude of this term determines the size of the logarithm, we observe that the table must be huge to provide the required accuracy for values of A near 1. Specifically, if a single logarithm lookup table indexed with n_1 bits were used to cover all inputs $\hat{A}[-1 : -n_1]$ across the interval $[0, 1]$, (16) implies it would have to be large enough that $2^b\varepsilon_1 < 2^{-(p+2)}$ or $\varepsilon_1 < 2^{-(p+b+2)}$. Equation (2) requires $n_1 = p + b + 1$, so the lookup table would have 2^{p+b+1} entries. This is unacceptably costly (256K entries) for $b = 7$, $p = 10$. This section shows how to use multiple tables spanning subintervals of the domain to reduce the overall table size. It then examines bipartite tables to further reduce size. Finally, it defines the exponent table size.

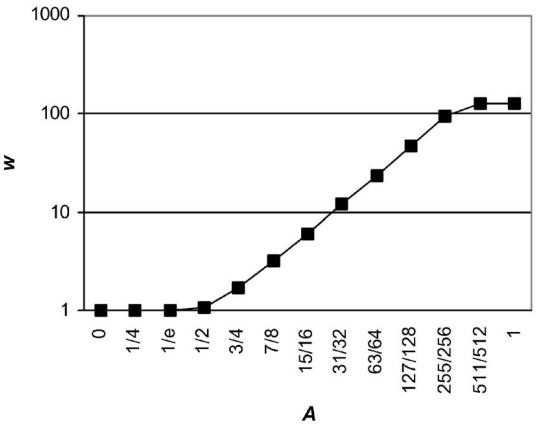

Fig. 4. Weight on ε_1 error.

4.1 Logarithm Table Design

Notice that the weight on the ε_1 error in (16) introduced by indexing the logarithm table with a finite number of bits increases as A approaches 1. Hence, we use multiple logarithm lookup tables valid over different subintervals with greater precision for inputs close to unity. Specifically, we use $b + 2$ tables: $T_0 \ldots T_{b+1}$. Table T_i covers the subinterval $[1 - 2^{-i}, 1 - 2^{-(i+1)})$ except the last table T_{b+1} covers the subinterval $[1 - 2^{-(b+1)}, 1)$. Each table is indexed with only $\tilde{n}_1$ bits of $\hat{A}$ and returns an approximation to the logarithm $L[p' - 1 : -n_3^i]$, where the number of fractional bits n_3^i increases with the table number i.

We index the table using $\hat{A}$, the fixed point representation of A. Values of A in T_i $(0 \le i \le b)$ are binary fractions with i leading 1s. We treat the subsequent $\tilde{n}_1$ bits as the index j into the table and truncate the remaining least significant bits. Therefore, the jth entry of T_i holds $L[p' - 1 : -n_3^i] = -\log_2(\hat{A} + 2^{-(\tilde{n}_1+i+2)})$ for $\hat{A} = 1 - 2^{-i} + j2^{-(i+\tilde{n}_1+1)}$. Hence, the round-off error is $|\varepsilon_1| \le 2^{-(\tilde{n}_1+i+2)}$. Table T_{b+1} covers the same size subinterval as table T_b, so, for values of A in this table, $|\varepsilon_1| \le 2^{-(\tilde{n}_1+b+2)}$. Now, we can find a bound on the error introduced in the result by the finite sized logarithm lookup tables.

Theorem 1. *The maximum weighted magnitude of the ε_1 error term in (16) introduced by table T_i is $|w_i\varepsilon_1| \le 1.07 \cdot 2^{-(\tilde{n}_1+2)}$.*

Proof. The breakpoint $A = e^{-2^{-b}}$ occurs in table T_b, as seen from the Taylor series approximation $e^{-2^{-b}} \approx 1 - 2^{-b} + \frac{1}{2}(2^{-b})^2$. Therefore, we divide the proof into two parts, one for table T_{b+1} and the other for tables $T_0 \ldots T_b$.

For table T_{b+1}, $|w_{b+1}\varepsilon_1| \le 2^b 2^{-(\tilde{n}_1+b+2)} = 2^{-(\tilde{n}_1+2)}$.

For tables $T_0 \ldots T_b$, $|w_i| < \frac{-1}{e(1-2^{-(i+1)})\ln(1-2^{-(i+1)})}$. Evaluating numerically, we find $|w_i| < 1.07 \cdot 2^i$, so $|w_i\varepsilon_1| < 1.07 \cdot 2^i \cdot 2^{-(\tilde{n}_1+i+2)} = 1.07 \cdot 2^{-(\tilde{n}_1+2)}$. For large i, a first order Taylor series approximation shows $w_i \to \frac{2}{e}2^i$, so the bound becomes loose for $i > 0$. □

Theorem 2. *The maximum weighted magnitude of the ε_3 error term in (18) introduced by table T_i is $|v_i\varepsilon_3| \le 2^{-(n_3^i+1-i)}$.*

Proof. From (3), we know $|\varepsilon_3| \le 2^{-(n_3^i+1)}$. Again, we divide the proof into two parts, one for table T_{b+1} and the other for tables $T_0 \ldots T_b$.

For table T_{b+1},

$$|v_{b+1}\varepsilon_3| \le 2^b \ln 2 \cdot 2^{-\left(n_3^{b+1}+1\right)} < 2^{-\left(n_3^{b+1}+1-(b+1)\right)}.$$

For tables $T_0 \ldots T_b$,

$$|v_i| < \frac{-\ln 2}{e(1-2^{-(i+1)})\ln(1-2^{-(i+1)})}.$$

Evaluating numerically, we find $|v_i| < 2^i$, so

$$|v_i\varepsilon_3| < 2^i \cdot 2^{-\left(n_3^i+1\right)} = 2^{-\left(n_3^i+1-i\right)}.$$

□

Given Theorem 1 and (16), we choose $\tilde{n}_1 = p$ bits to index the logarithm tables. Note that we violate the bound in (16) by the factor of 1.07. This is compensated for by the slack in (19). Similarly, given Theorem 2 and (18), we choose $n_3^i = i + p + 3$.

Note that T_0 contains p' integer bits and $p+3$ fractional bits. For $i > 0$, one can determine numerically that the integer bits and $i-1$ most significant fractional bits in T_i are all 0s, so the tables contain only $p+4$ nontrivial bits in each entry. The multiplier may thus be optimized to accept only $p'+p+3$ bits of L if followed by a right shift by i to compensate for the leading 0s. In general, the total number of bits in the logarithm tables is $2^p[(b+2)(p+4)+p'-1]$.

In summary, our design with $b = 7$, $p = 10$ requires nine logarithm tables of 1,024 entries each. T_0 has 4 integer bits and 13 fractional bits. The other tables have 14 fractional bits. The total table size is about 16KB. The table size reduces significantly if less accuracy is necessary; for example, a table for $p = 8$-bit color depth requires only 3.5KB of storage.

4.2 Bipartite Logarithm Tables

The logarithm table sizes can be reduced further using symmetric bipartite table approximations [17], [18], [19]. Consider evaluating $f(x) = -\log_2(x + 2^{-(p+1)})$, where x is the input A truncated to p bits. In the previous section, we looked up $f(x)$ in a 2^p-entry table. An alternative method is to divide x into three bit strings, denoted as x_0, x_1, and x_2, of lengths p_0, p_1, and p_2, respectively. The value of the input operand is $x = x_0 + x_1 + x_2$ and the length is $p = p_0 + p_1 + p_2$. Using the first part of a Taylor series

$$\begin{aligned} f(x) &= f(x_0 + x_1 + x_2) \\ &\approx f(x_0 + x_1) + x_2 \cdot f'(x_0 + x_1) \\ &\approx f(x_0 + x_1) + x_2 \cdot f'(x_0) \\ &\approx g(x_0, x_1) + h(x_0, x_2), \end{aligned} \tag{20}$$

where $g(x_0, x_1)$ is a table of initial values with $2^{p_0+p_1}$ entries containing $f(x_0 + x_1)$ and $h(x_0, x_2)$ is a table of offsets with $2^{p_0+p_2}$ entries containing $x_2 \cdot f'(x_0)$.

One can show that the lengths p_0, p_1, and p_2 depend on the desired accuracy of the result and the second derivative of f [19]. If the second derivative is relatively small, a Taylor series analysis shows choosing $p_0 \approx p_1 \approx p_2$ gives acceptable error [20] and requires approximately $2^{(2/3)p}$ table entries rather than the 2^p entries required in a conventional table.

A direct implementation of the logarithm lookup using a single bipartite table covering the range $[0, 1)$ is not feasible. The derivative of the function goes to infinity at $x = 0$, so the error in the lookup becomes unbounded. Even over the range $[0.5, 1)$, a single bipartite table would require more entries to produce acceptable error than do the multiple tables over subintervals developed in the previous section. However, we can reduce storage requirements by replacing each of the tables $T_1 \ldots T_b$ with a bipartite table.

Exploiting symmetry in the table of offsets $h(x_0, x_2)$ allows the table to be reduced in size by a factor of two to $2^{p_0+p_2-1}$ at the expense of a few XORs [19]. The rounded contents of the g and h tables and the exact error bounds are determined using the algorithm presented in [21]. Based on the error bounds, we choose $p_0 = 4$, $p_1 = 3$, $p_2 = 3$ for $p = 10$ and $p_0 = 3$, $p_1 = 2$, $p_2 = 3$ for $p = 8$. Thus, for $p = 10$, we replace each 1,024-entry lookup table $T_1 \ldots T_8$ with a 128-entry table of initial values, a 64-entry table of offsets, and a 14-bit carry-propagate adder. Similarly, for $p = 8$, we replace each 256-entry lookup table with a pair of 32-entry tables and a 12-bit adder. In both cases, we must keep the full unipartite T_0 to cover the subinterval $[0, 0.5)$. The combined logarithm table sizes are 5 KB or 1.3KB for $p = 10$ or 8, respectively. No guard bits are used because a tedious analysis shows that the extra error from an unfaithfully rounded result is smaller than the slack in the bounds on ε_3 and ε_2.

4.3 Exponent Table Design

As shown earlier, the exponent table is indexed with the upper $n_4 = p + 2$ fractional bits of the product computed by the multiplier and produces an answer rounded to p fractional bits. Thus, the table requires $p2^{p+2}$ bits of storage. The table for $p = 10$ has 5 KB of storage and a table for $p = 8$ has 1 KB of storage.

5 Implementation

This section describes Verilog implementations of the exponentation algorithm. The design using a logarithm table with multiple subintervals described in Section 4.1 is called the unipartite design. The design with multiple subintervals and bipartite tables described in Section 4.2 is called the bipartite design. It presents a block diagram of the unit, the verification methodology, and the synthesis results. The inputs A and B are IEEE single-precision floating-point numbers. The output P is calculated as a fixed-point number with p bits of fraction and is converted to floating-point format for later use. The Verilog model is parameterized by b and p.

Fig. 5 shows a block diagram of the exponentiation unit. The floating-point inputs A and B are converted to fixed-point format $\hat{A}$ and $\hat{B}$. $\hat{A}$ is presented to $b+2$ log tables covering the subintervals of $[0, 1]$ and the logarithm L is selected from the appropriate table based on the number of leading 1s, i. To reduce the size of the multiplier, the right shift of L by i is delayed until after the multiplication. The exponent table looks up E based on the fractional bits of X.

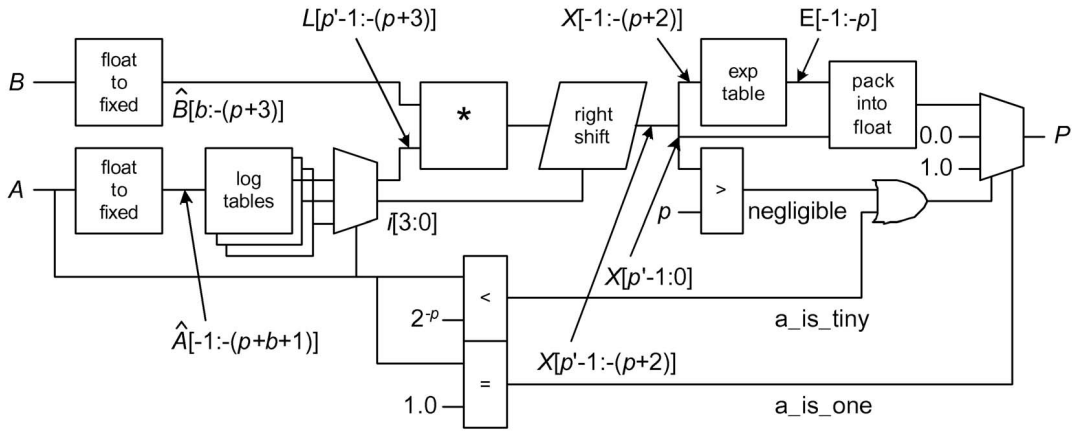

Fig. 5. Refined block diagram of exponentation unit.

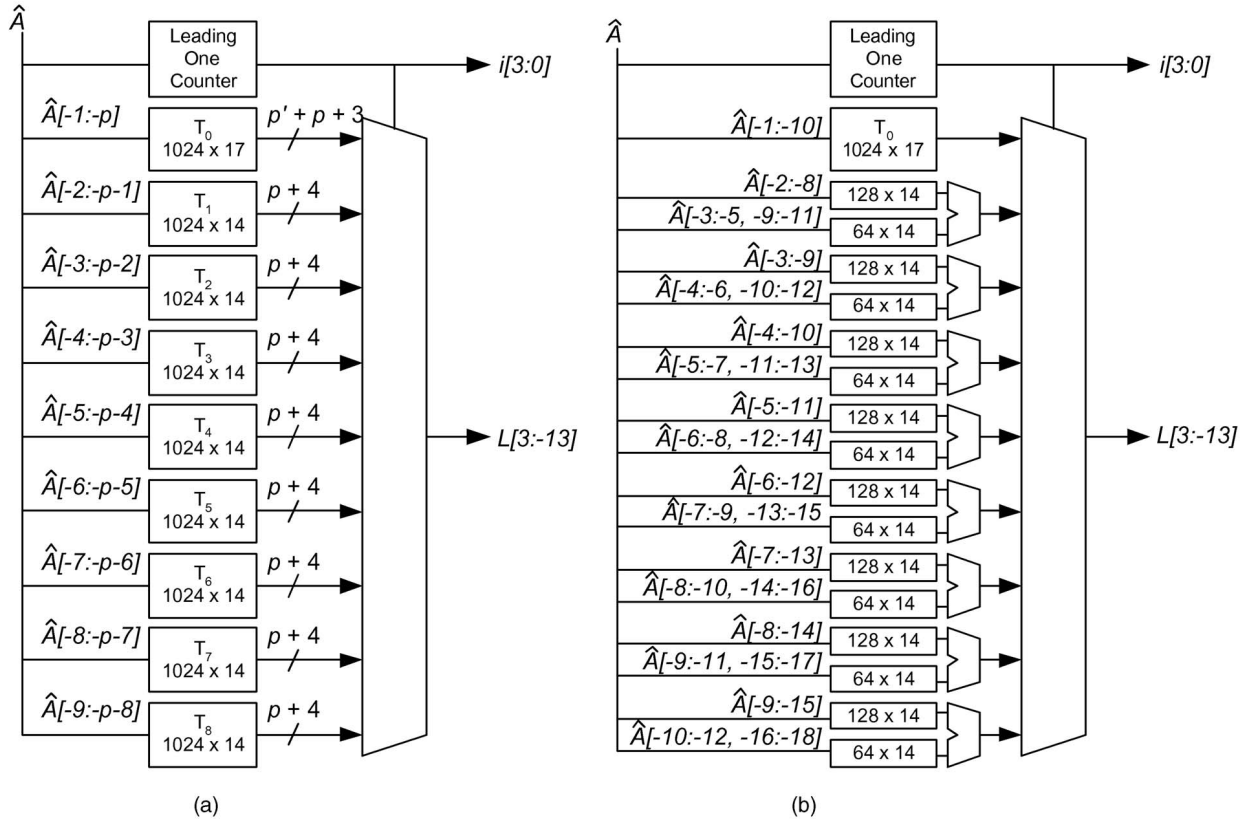

Fig. 6. Log table designs for $p = 10$.

In the normal case, the floating-point result P has a significand equal to E and an exponent equal to the integer part of X. The result multiplexer selects 1.0 in the special cases of $A = 1.0$, 0.0 if A is tiny or X is too large, or A^B otherwise.

In the unipartite design, each of the log tables has 2^p entries. In the bipartite design, each of the tables consists of a smaller bipartite table and an adder. Fig. 6 illustrates these tables for $p = 10$, $b = 7$. Each also contains a leading-one counter and multiplexer to select the appropriate table T_i.

5.1 Verification

VCS simulations verified the Verilog implementation against a C reference model. The C model generates the tables and determines both the expected Verilog result and the true value of A^B to ensure the algorithm rounds faithfully.

The test vectors include both directed and random tests for $b = 7$ and $p = 8$ and 10. Six million random vectors were applied. For faithful rounding, the maximum error must be less than 2^{-p} (0.0039 for $p = 8$ or 0.00098 for $p = 10$). In the unipartite design, the maximum errors found were 0.0030 and 0.00080, respectively. In the bipartite design, the maximum errors were 0.0031 and 0.00082. The small extra

TABLE 1
Exponentiation Unit Size

Module	$p=8$				$p=10$			
	Unipartite		Bipartite		Unipartite		Bipartite	
	Gates	Bits	Gates	Bits	Gates	Bits	Gates	Bits
Log Tables	14,867	28,416	5,491	11,008	72,734	132,096	20,353	38,912
Exponent Table	1,823	8,192	1,866	8,192	6,181	40,960	6,187	40,960
Multiplier	2,317		2,579		3,035		3,171	
Random Logic	1,176		1,438		1,669		2,050	
Total	36,608		11,374		83,619		31,761	

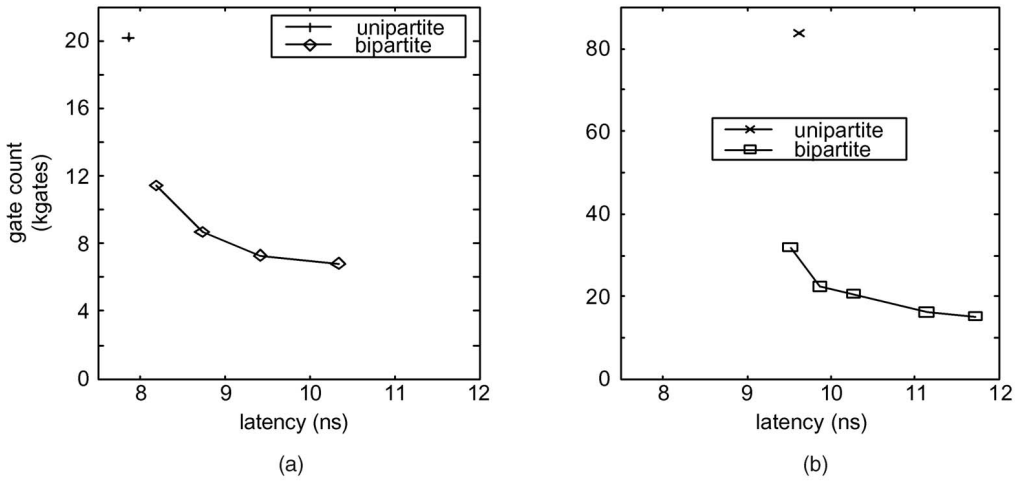

Fig. 7. Gate count versus latency. (a) p = 8. (b) p = 10.

error in the bipartite design comes from round-off in the log tables that could be avoided with an additional guard bit [21].

5.2 Synthesis Results

The exponentiation unit was synthesized with Synopsys tools and mapped to the LSI Logic G12-p 180 nm cell library [22] using worst-case models. The gate count of each component is listed in Table 1 with a conversion of one gate to 24 μm^2, i.e., four LSI cell units. No ROM generator was available, so lookup tables were synthesized onto the cell library. If a ROM generator were available, the number of ROM bits required for each table is also shown in the table with an estimated conversion of one bit to about 2 μm^2. In comparison, the design of [13] requires 128 Kb of ROM for < 8-bit accuracy, [14] uses 73 Kb of ROM for < 6-bit accuracy when $B = 128$, and [15] uses a few adders and a floating-point multiplier for < 6-bit accuracy.

Fig. 7 plots the gate count versus cycle time for the exponentiation units. The bipartite designs pay an extra adder delay to sum the table results, but, in the case of $p = 10$, the smaller bipartite tables are faster by more than enough to offset the adder delay. If cycle time can be relaxed further, a significant area savings is achievable.

Adding a factor of two to account for estimated interconnect, the overall synthesized areas of the fastest bipartite designs are 0.6 mm^2 for $p = 8$ and 1.5 mm^2 for $p = 10$. Building the tables from ROMs rather than combinational logic would reduce the areas to 0.3 and 0.6 mm^2, respectively. The exponentiation unit can be partitioned before and after the multiplier into a 3-stage pipeline to match the cycle time (typically 3-4 ns) of 180 nm graphics accelerators.

As the OpenGL standard requires up to two exponentiations per vertex per light source (if the source has specular and spotlight attributes), the pipelined design can process at least 125 megavertices/second for a single light source. In comparison, the AGP 8x memory interface has a bandwidth of 2.1 GB/s, which is sufficient to deliver at most 88 megavertices/second to the geometry engine if each vertex consists of three single-precision floating-point XYZ coordinates and three color (RGB) or normal $(N_x N_y N_z)$ components. Multiple light sources may be handled either at reduced throughput or with multiple exponentiation units operating in parallel. This hardware exponent unit may also be applicable to Phong shading, which requires lighting calculations on a per pixel rather than a per vertex basis.

6 Conclusion

This paper described a hardware implementation of a exponentiation unit suitable for OpenGL lighting computations or other applications with similar accuracy requirements. The unit calculates $P = A^B$, where A and B are IEEE single-precision floating-point numbers in the range $[0, 1]$ and $[1, 2^b]$, respectively, and P is faithfully rounded to p fractional bits. The unit uses a logarithm lookup, a multiplier, and an exponent lookup. Error analysis shows that the logarithm lookup table accuracy requirements depend on the value of A, so the unit uses multiple tables

over different subintervals of A to minimize the overall table size. Bipartite tables are used for most subintervals to further reduce table sizes. This implementation, good to 10 bits of accuracy, uses nine log lookup tables, a 2,048-entry exponent lookup table, and a multiplier. Synthesized in a 180 nm process, it has an area of 1.5 mm^2 and a latency of 9.4 ns. Another design with 8-bit accuracy has an area of 0.6 mm^2 and a latency of 8.2 ns. In comparison, the design of [13] requires a larger lookup table than the 10-bit bipartite design yet is accurate to less than 8 bits, is susceptible to banding artifacts, and does not support noninteger powers. The designs are freely available through the Harvey Mudd Open Source Floating Point Project [23]. Future areas of potential work to further reduce the logarithm table size include multipartite tables [24], [15], nonuniform intervals, or first or second-order interpolation.

ACKNOWLEDGMENTS

The author thanks Alan Scott at Evans and Sutherland for practical advice on OpenGL applications. Michael Schulte suggested using multipartite tables to reduce the table size. The anonymous reviewers contributed numerous helpful suggestions.

REFERENCES

[1] M. Segal and K. Akeley, *The OpenGL Graphics System: A Specification (Version 1.4),* Silicon Graphics, 2002, http://www.opengl.org.
[2] M. Woo et al., *OpenGL Programming Guide,* third ed. Reading, Mass.: Addison Wesley, 1999.
[3] "nVIDIA Technical Brief, Transformation and Lighting," NVIDIA Corp., 1999.
[4] *IEEE Standard for Binary Floating Point Arithmetic,* ANSI/IEEE Std. 754-1985, 1985.
[5] J. Foley, A. Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice.* Reading, Mass.: Addison Wesley, 1996.
[6] J. Muller, *Elementary Functions.* Boston: Birkhauser, 1997.
[7] M. Overton, *Numerical Computing with IEEE Floating Point Arithmetic.* Philadelphia: SIAM, 2001.
[8] P. Tang, "Table-Driven Implementation of the Exponential Function in IEEE Floating Point Arithmetic," *ACM Trans. Math. Software,* vol. 15, no. 2, pp. 144-157, June 1989.
[9] P. Tang, "Table-Lookup Algorithms for Elementary Functions and Their Error Analysis," *Proc. 10th Symp. Computer Arithmetic,* pp. 232-236, 1991.
[10] N. Takagi, "Powering by a Table Look-Up and a Multiplication with Operand Modification," *IEEE Trans. Computers,* vol. 47, no. 11, pp. 1216-1222, Nov. 1998.
[11] J. Pineiro, J. Bruguera, and J. Muller, "Faithful Powering Computation Using Table Look-Up and a Fused Accumulation Tree," *Proc. 15th Symp. Computer Arithmetic,* pp. 40-47, 2001.
[12] W. Cody and W. Wait, *Software Manual for the Elementary Functions.* Englewood Cliffs, N.J.: Prentice-Hall, 1980.
[13] H. Shin, J. Lee, and L. Kim, "A Hardware Cost Minimized Fast Phong Shader," *IEEE Trans. VLSI Systems,* vol. 9, no. 2, pp. 297-304, Apr. 2001.
[14] C. Chen and C. Lee, "A Cost Effective Lighting Processor for 3D Graphics Applications," *Proc. IEEE Int'l Conf. Image Processing,* vol. 2, pp. 792-796, 1999.
[15] Y. Kwon, I. Park, and C. Kyung, "A Hardware Accelerator for the Specular Intensity of Phong Illumination Model in 3-Dimensional Graphics," *Proc. ACM Asia/South Pacific Design Automation Conf.,* pp. 559-546, 2000.
[16] J. Coleman, E. Chester, C. Softley, and J. Kadlec, "Arithmetic on the European Logarithmic Microprocessor," *IEEE Trans. Computers,* vol. 49, no. 7, pp. 702-715, July 2000.
[17] H. Hassler and N. Takagi, "Function Evaluation by Table Look-Up and Addition," *Proc. 12th Symp. Computer Arithmetic,* pp. 10-16, 1995.
[18] D. Das Sarma and D. Matula, "Faithful Bipartite ROM Reciprocal Tables," *Proc. 12th Symp. Computer Arithmetic,* pp. 17-29, 1995.
[19] M. Schulte and J. Stine, "Approximating Elementary Functions with Symmetric Bipartite Tables," *IEEE Trans. Computers,* vol. 48, no. 8, pp. 842-847, Aug. 1999.
[20] J. Muller, "A Few Results on Table-Based Methods," *Reliable Computing,* vol. 5, no. 3, pp. 279-288, 1999.
[21] F. Dinechin and A. Tisserand, "Some Improvements on Multipartite Table Methods," *Proc. 15th Symp. Computer Arithmetic,* pp. 128-135, 2001.
[22] LSI Logic, *G12-p Cell-Based ASIC Products,* 1999.
[23] "Harvey Mudd College Open Source Floating Point Project," http://www.hmc.edu/chips, 2000.
[24] J. Stine and M. Schulte, "The Symmetric Table Addition Method for Accurate Function Approximation," *J. VLSI Signal Processing,* vol. 21, no. 2, pp. 167-177, 1999.

David Harris (M'94) received the PhD degree from Stanford University in 1999 and the MEng and SB degrees from the Massachusetts Institute of Technology in 1994. Since 1999, he has been an assistant professor of engineering at Harvey Mudd College. He has also designed chips at Sun Microsystems, Intel, Hewlett-Packard, Evans and Sutherland, and elsewhere. His primary research and teaching interests are in the area of high-speed integrated circuits and digital design. He is the author of two books and holds seven patents. When not teaching or designing chips, he may be found rock climbing or flying a Cessna. He is a member of the IEEE.