A Powering Unit for an OpenGL Lighting Engine

David_Harris@hmc.edu
Harvey Mudd College

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 a powering unit based on a logarithm lookup table, a multiplier, and an inverse log table. The log lookup table is partitioned into subintervals to reduce table size. A synthesized design uses 84k gates to achieve 10-bit accuracy with a latency of 9.62 ns in a 180 nm process. Although the system is tailored for the OpenGL application, the same principles can be applied to the design of other powering units.

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. Therefore, the transformation and lighting calculations are moving from the host processor to a hardware transform and lighting (T&L) engine [3].

The T&L 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 non-integer) 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 [4].

Accurately computing A^B is considered a difficult floating-point operation [5]. Approximations for specific cases can be much more efficient. For example, Tang [6,7] describes an algorithm for exp(B) using range reduction, a polynomial approximation, and reconstruction. Lookup tables are used to assist the reconstruction. Efficient approaches involving table lookup and interpolation exist when B is a constant [8,9]. Software math libraries [10] often rely on many multiplications. If B were an integer, a limited number of multiplications would suffice. Unfortunately, none of these approaches are well suited to a low-cost direct hardware implementation supporting floating-point values for A and B but requiring only modest accuracy. Indeed, the author is unaware of any published work on powering units optimized for such criteria.

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. The algorithm uses the identity

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

where lookup tables are used for the logarithm and exponential. For OpenGL, b is 7 and p is typically 8 or 10, depending on 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 size grows as $O(2^{b+p})$ and is impractically large. A key idea of this paper is to partition the logarithm table into multiple tables over subintervals, as done by Coleman *et al*. [11]. This leads to $O(b)$ tables that grow as $O(2^p)$ for an area of $O(b2^p)$.

This paper presents the algorithm and error analysis. Based on the error analysis, we determine the size of the lookup tables. The design was coded in Verilog and synthesized to produce area and timing results. The same principles apply to the design of hardware powering units for other applications.

2: Algorithm

We wish to compute $P = A^B$ where A and B are floating point numbers in the ranges [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 powering unit because it is very expensive to calculate the result to the high level of precision required to round exactly [5].

We begin with the identity $A^B = 2^{B \log_2 A}$. The integer portion of $\log_2 A$ is the exponent field of A because A is provided as a floating point number. The fractional portion of the logarithm is looked up from a table given the significand field of A. A multiplier computes $X = B \log_2 A$ and the result is expressed in fixed point format. We finally determine the result 2^X by table lookup on the fractional part of X followed by a shift by the integer portion of X. For convenience, this fraction may be expressed as a floating point number for use later in the pipeline.

To express the algorithm more precisely, we must define some notation. Let the bit vector $x[m{:}n]$ represent $\sum_{i=n}^{m} x[i]2^i$. Let $p' = \lceil \log_2(p+1) \rceil$; this 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 truncated to n_1 fractional bits and $\hat{B}[b:-n_2]$ be 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 truncation and shift of the signficands. Let $\hat{P}[0:-p]$ be the final result faithfully rounded to p fractional bits.

Figure 1 shows the powering 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 for each intermediate result required to achieve a particular accuracy will be explored in the next section.

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.

```
if  Â = 1  then  P̂ = 1
else if  Â < 2^-(p+1)  then  P̂ = 0
else begin
   lookup  L[p'-1 : -n3] = -log2(Â + 2^-(n1+1))
   multiply  X[p'-1 : -n4] = L • B̂
   if  X ≥ p+1  then  P̂ = 0
   else begin
      lookup  E[-1 : -p] = 2^-(X[-1:-n4] + 2^-(n4+1))
      right shift  P̂[0 : -p] = E >> X[p'-1 : 0]
   end
end
```

Figure 1: Powering algorithm

The logarithm table should look up the logarithm of A, but instead looks up the logarithm of $A + \varepsilon_1$ = $\hat{A} + 2^{-(n_1+1)}$ where the truncation error is

$$|\varepsilon_1| \le 2^{-(n_1+1)} \tag{2}$$

Observe the benefit of programming the logarithm table for entry $\hat{A}$ with $-\log_2\left(\hat{A} + 2^{-(n_1+1)}\right)$: if we had programmed the table with $-\log_2\left(\hat{A}\right)$, the error ε_1 caused by truncation of A could be twice as great.

The logarithm table produces a result 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.

$$|\varepsilon_3| \le 2^{-(n_3+1)} \tag{3}$$

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

$$|\varepsilon_2| = B - \hat{B} < 2^{-n_2} \tag{4}$$

The product is truncated to n_4 bits before being used in the exponent lookup table. As in the logarithm table

$$|\varepsilon_4| \le 2^{-(n_4+1)} \tag{5}$$

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

$$|\varepsilon_5| \le 2^{-(p+1)} \tag{6}$$

Considering all these errors, we actually compute

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

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

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

Because the errors are small, we use first-order Taylor series approximations for $\log_2 x$ and 2^x

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

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

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

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

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. 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. In both cases, this maximum occurs at

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

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

$$\left|A^B\log_2 A\ln 2\right|\le\frac{1}{e} \quad (14)$$

Using (13) and (14) and taking an upper bound of 1 for A^B, we reduce (12) to

$$\left.\begin{matrix}\left|\varepsilon_1+\frac{\varepsilon_2}{e}+\varepsilon_3 A\ln 2+\varepsilon_4\ln 2+\varepsilon_5\right|\\ \left|\frac{-\varepsilon_1}{eA\ln A}+\frac{\varepsilon_2}{e}+\frac{-\varepsilon_3\ln 2}{e\ln A}+\varepsilon_4\ln 2+\varepsilon_5\right|\\ \left|\varepsilon_1 2^b+\frac{\varepsilon_2}{e}+\varepsilon_3 2^b\ln 2+\varepsilon_4\ln 2+\varepsilon_5\right|\end{matrix}\right\}<2^{-p}\ \begin{matrix}A<e^{-1}\\ \text{otherwise}\\ A>e^{-2^{-b}}\end{matrix} \quad (15)$$

We already bounded ε_5 in (6). Reducing the errors ε_1 and ε_4 is costly because these determine the sizes of the logarithm and exponent lookup tables. To minimize table sizes while obtaining sufficient accuracy, we will choose n_1, n_2, n_3, and n_4 to satisfy (15).

$$\left.\begin{matrix}|\varepsilon_1|\\ \frac{-1}{eA\ln A}|\varepsilon_1|\\ 2^b|\varepsilon_1|\end{matrix}\right\}<2^{-(p+2)}\ \begin{matrix}A<e^{-1}\\ \text{otherwise}\\ A>e^{-2^{-b}}\end{matrix} \quad (16)$$

$$\frac{|\varepsilon_2|}{e}<2^{-(p+4)} \quad (17)$$

$$\left.\begin{matrix}A\ln 2|\varepsilon_3|\\ \frac{-\ln 2}{e\ln A}|\varepsilon_3|\\ 2^b\ln 2|\varepsilon_3|\end{matrix}\right\}<2^{-(p+4)}\ \begin{matrix}A<e^{-1}\\ \text{otherwise}\\ A>e^{-2^{-b}}\end{matrix} \quad (18)$$

$$\ln 2|\varepsilon_4|<2^{-(p+3)} \quad (19)$$

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: Implementation

This section describes a Verilog implementation of the powering algorithm. It presents the table designs, 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. The key for an efficient design is a logarithm table partitioned into multiple subintervals.

4.1: Logarithm table design

If a single logarithm lookup table indexed with n_1 bits were used to cover all inputs A across the interval [0,1], (16) implies it would have to be large enough that $2^b\varepsilon_1<2^{-(p+2)}$ or $\varepsilon_1<2^{-(p+b+2)}$. (2) requires $n_1=p+b+1$, so the lookup table would have 2^{p+b+1} entries. This is costly for $b=7$, $p=10$.

Notice that the weight on the ε_1 error in (16) 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 table T_{b+1} covers the subinterval [1-$2^{-(b+1)}$, 1). Each table is indexed with only $\tilde{n}_1$ bits of 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 a fixed point representation of *A*. Values of *A* in $T_i\ (0 \le i \le b)$ are binary fractions with *i* leading 1's. We treat the subsequent $\tilde{n}_1$ bits as the index *j* into the table and truncate the remaining less significant bits. Therefore, the *j*th entry of T_i holds $L[p'-1:-n_3^i] = -\log_2\left(\hat{A} + 2^{-(\tilde{n}_1+i+2)}\right)$ for $\hat{A} = 1 - 2^{-i} + j2^{-(i+\tilde{n}_1+1)}$. Hence, $|\varepsilon_1| \le 2^{-(\tilde{n}_1+i+2)}$. 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 $|w_i\varepsilon_1| \le 1.07 \bullet 2^{-(\tilde{n}_1+2)}$.

Proof: The breakpoint $A = e^{-2^{-b}}$ occurs in table T_b, as seen from the Taylor series approximation $e^{-2^{-b}} \approx 1 - 2^{-b} + \frac{1}{2}\left(2^{-b}\right)^2$. 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}, $|w_{b+1}\varepsilon_1| \le 2^b \bullet 2^{-(\tilde{n}_1+b+2)} = 2^{-(\tilde{n}_1+2)}$.

For tables $T_0 \ldots T_b$, $|w_i| < \dfrac{-1}{e\left(1-2^{-(i+1)}\right)\ln\left(1-2^{-(i+1)}\right)}$.

Evaluating numerically, we find $|w_i| < 1.07 \bullet 2^i$, so $|w_i\varepsilon_1| < 1.07 \bullet 2^i \bullet 2^{-(\tilde{n}_1+i+2)} = 1.07 \bullet 2^{-(\tilde{n}_1+2)}$. 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 $|v_i\varepsilon_3| \le 2^{-\left(n_3^i+1-i\right)}$.

Proof: From (3), we know $|\varepsilon_3| \le 2^{-\left(n_3^i+1\right)}$. 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}, $|v_{b+1}\varepsilon_3| \le 2^b \ln 2 \bullet 2^{-\left(n_3^{b+1}+1\right)} < 2^{-\left(n_3^{b+1}+1-(b+1)\right)}$.

For tables $T_0 \ldots T_b$, $|v_i| < \dfrac{-\ln 2}{e\left(1-2^{-(i+1)}\right)\ln\left(1-2^{-(i+1)}\right)}$.

Evaluating numerically, we find $|v_i| < 2^i$, so $|v_i\varepsilon_3| < 2^i \bullet 2^{-\left(n_3^i+1\right)} = 2^{-\left(n_3^i+1-i\right)}$.

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 0's, 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 0's. In general, the total number of bits in the logarithm tables is $2^p\left[(b+2)(p+4) + p' - 1\right]$.

In summary, our design with $b = 7$, $p = 10$ requires nine logarithm tables of 1024 entries each. T_0 has four integer bits and thirteen fractional bits. The other tables have fourteen fractional bits. The total table size is about 16KB, nontrivial, yet still modest in comparison to the sea of multipliers used by a geometry engine. 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: 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.

4.3: Block diagram

Figure 2 shows a block diagram of the powering unit. Portions of the significand of *A* are presented to the log tables to cover each subinterval of [0,1]. The result from the appropriate subinterval is selected and multiplied by *B* with an ordinary unsigned multiplier. The product *X* indexes an exponent table. The result multiplexer selects 1 in the special cases of *A* = 1, 0 if *A* is tiny or *X* is too large, or A^B otherwise. Not shown are the small shifters and adders required to convert between floating point and the short fixed point formats.

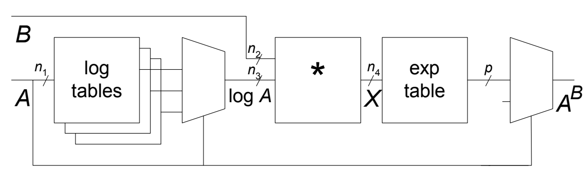

Figure 2: Block diagram of powering unit

4.4: Verification

VCS simulations verified the Verilog implementation against a C reference model. The C model 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. The maximum error found in simulation for $p = 8$ was 0.0029. This is better than our bound of $2^{-p} = 0.0039$ because the worst case errors in the log and exponent lookups do not occur simultaneously on any of the cases tested. The maximum error found for $p = 10$ was $0.00076 < 2^{-p} = 0.00098$.

4.5: Synthesis results

The powering unit was synthesized with Synopsys tools and mapped to the LSI Logic G12-p 180 nm cell library [12] 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. 4 LSI cell units. No ROM generator was available so lookup tables were synthesized into gates instead. If a ROM generator were available, the number of ROM bits required in place of the table gates is also shown in the table with an estimated conversion of one bit to about 2 μm^2.

Module	$p = 8$		$p = 10$	
	Bits	**Gates**	**Bits**	**Gates**
Log Tables	28416	14867	132096	72734
Exponent Table	8192	1823	40960	6181
Multiplier		2317		3035
Random Logic		1176		1669
Total	36608	20183	173056	83619

Table 1: Powering unit size

Adding a factor of two to account for estimated interconnect, the overall synthesized areas are 1 mm^2 for $p = 8$ and 4 mm^2 for $p = 10$. Using ROMs could reduce the area to 0.6 mm^2 for $p = 10$. The latencies are 7.87 and 9.62 ns, respectively. The powering unit could be partitioned into a 3-stage pipeline to improve cycle time.

5: Conclusion

This paper described a hardware implementation of a powering 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 ranges of A to minimize the overall table size. This implementation, good to 10 bits of accuracy, uses nine 1024-entry log lookup tables, a 2048-entry exponent lookup table, and a multiplier. Synthesized in a 180 nm process, it has an area of 4 mm^2 and a latency of 9.62 ns. Using ROMs for table storage could reduce the area significantly. Another design with 8-bit accuracy has an area of 1 mm^2 and a latency of 7.87 ns. The designs are freely available through the Harvey Mudd Open Source Floating Point Project [13].

Acknowledgments

The author thanks Alan Scott at Evans and Sutherland for practical advice on OpenGL applications. Stuart Oberman has been a consistent source of expert advice on floating point arithmetic.

References

[1] M. Segal and K. Akeley, *The OpenGL 1.2.1 Graphics System: A Specification (Version 1.2.1),* Silicon Graphics, 1999, `http://www.opengl.org`

[2] M. Woo *et al., OpenGL Programming Guide, 3rd Edition*, Addison Wesley: Reading, MA, 1999.

[3] nVIDIA Technical Brief, *Transformation and Lighting*, NVIDIA Corporation, 1999.

[4] 754-1985, *IEEE Standard for Binary Floating Point Arithmetic*, IEEE, 1995.

[5] J. Muller, *Elementary Functions*, Birkhauser: Boston, MA, 1997.

[6] P. Tang, "Table-Driven Implementation of the Exponential Function in IEEE Floating-Point Arithmetic," *ACM Transactions on Mathematical Software*, Vol. 15, No. 2, June 1989, pp. 144-157.

[7] P. Tang, "Table-Lookup Algorithms for Elementary Functions and Their Error Analysis," *Proc. 10th Symp. Computer. Arithmetic*, pp. 232-236, 1991.

[8] 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.

[9] J. Pineiro, J. Bruguera, and J. Muller, "Faithful powering computation using table look-up and a fused accumulation tree," *Proc. 15th Symp. Comp. Arithmetic*, pp. 40-47, 2001.

[10] W. Cody and W. Wait, *Software Manual for the Elementary Functions*, Prentice-Hall: Englewood Cliffs, NJ, 1980.

[11] T. 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.

[12] LSI Logic, *G12-p Cell-Based ASIC Products*, 1999.

[13] Harvey Mudd College Open Source Floating Point Project, `http://www.hmc.edu/chips`