A VLSI-Design for fast Vector Normalization

Günter Knittel
Wilhelm-Schickard-Institut für Informatik - Graphisch-Interaktive Systeme (WSI/GRIS)
Universität Tübingen
Auf der Morgenstelle 10, D-72076 Tübingen
email: knittel@goya.gris.informatik.uni-tuebingen.de

Abstract

The design of a vector normalizer is described. It is an integral part of our graphics subsystem for scientific visualization, but will be of great use for speeding up any computer graphics architecture.

In the actual design, the circuitry handles 3D-vectors with 33 bit two's complement components. The components of the normalized vectors are computed as 16 bit two's complement fixed-point numbers. Due to the overall pipeline architecture, the chip accepts one 3D-vector and produces one normalized vector each clock.

To normalize a 3D-vector, three square operations, two additions, one square root operation and three divisions must be performed. The target clock frequency is 50 MHz, by which the performance of the chip rates at 450 MOPS.

A single-chip VLSI implementation is currently in work, simulation results will be available by the end of the third quarter '93. We use Mentor 8.2 tools on HP 700 workstations and Toshiba's TC160G Gate Array technology.

Keywords: graphics hardware, arithmetic accelerator, real-time Phong shading

1 Introduction

Most computer graphics algorithms require fast and frequent vector normalizations. For example, the well-known Phong illumination model [BuiT75]

$$I = I_A k_a C + I_L \left(k_d C (\vec{G}_N \vec{L}_N) + k_s (\vec{G}_N \vec{H}_N)^n \right) \quad \text{(simplified)}^* \qquad (1)$$

calculates the light intensity I of a point on an object surface according to four unit vectors:

- ❑ the surface normal $\vec{G}_N$,
- ❑ the normalized vector $\vec{L}_N$ in direction of the light source and
- ❑ the so-called halfway vector $\vec{H}_N$, which in turn is the normalized sum of $\vec{L}_N$ and the normalized vector $\vec{V}_N$ in direction of the observer.

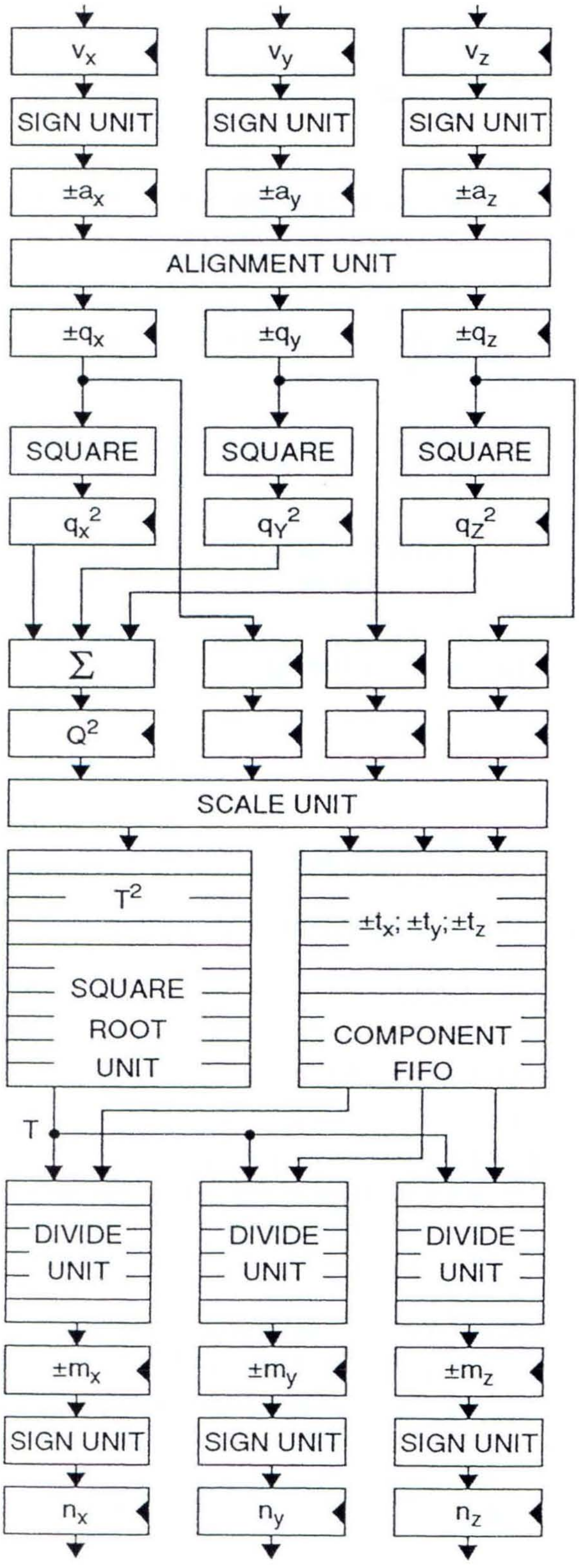
Applications aiming at virtual reality, e.g., the graphics subsystem for volume rendering developed at WSI/GRIS [Knit93], must provide perspective projection and non-parallel light, that is, none of the vectors is constant. Unfortunately, normalizing a vector presents a great computational expense (especially the square root operation) and, moreover, (1) has to be evaluated several millions of times each second.

This was the motivation to develop a high-speed single-chip vector normalizer. The large number of vectors to be processed sequentially permits the use of a moderately deep pipeline structure without any performance penalty. For the square root function, an algorithm was adapted which computes one result bit per stage and uses only a small circuitry within each stage. The architecture of the chip is scalable with respect to speed and required chip space (by placing more or less functional units into a single pipeline stage) or precision (by adding the appropriate number of stages and operand bits).

* I_A: ambient light, I_L: light coming from the light source, k_a,k_d,k_s: ambient, diffuse and specular reflection coefficients, C: color of the object, n: specular reflection exponent

2 Architectural Overview

The circuitry described on the following pages accepts the components of a 3D-vector $\vec{V} = \{v_x;v_y;v_z\}$ and produces its associated normalized vector $\vec{N} = \{n_x;n_y;n_z\}$.



The block diagram shows the deep but regular pipeline structure of the chip. The boxes with the small filled triangle represent registers. The register structure within the pipelined units (square root unit and divide unit) has been ommitted for clarity, but will be explained in later sections.

Operands which skip certain functional units must travel through pipeline registers (FIFOs) to maintain synchronization. Thus, FIFO memories must also be placed onto the chip.

There are no feedbacks or functional units for exception handling required, by which the control structure becomes extremely simple. There is an additional valid flag which travels along with each vector and a small circuitry to mask the clock. Besides that, the chip has just to be clocked.

The excessive pipeline structure relies on a great number of vectors to be processed sequentially, as is the case in most computer graphics applications and especially in the algorithms used in our voxel subsystem. Thus, the pipeline will always be filled and so operate at maximum efficiency.

We assume a global space of 32 bit extent in each direction, that is

$$-2^{31} \le x, y, z \le 2^{31} - 1 \ . \qquad (2)$$

Therefore, the input operands are expected to be 33 bit two's complement integers. Smaller operands must be sign-extended to 33 bits. The components of the normalized vector are computed as 16 bit two's complement fixed-point numbers

$$n = -n_0 \times 2^0 + \sum_{j=-1}^{-15} n_j \times 2^j \ . \qquad (3)$$

Thus, the chip has 147 I/O - pins (excluding control-, test- and clock-terminals).

We will now describe all functional units in dataflow order in details, e.g. by Boolean equations or by schematic drawings. For each functional unit, a coarse gate count estimation will be given.

3 Naming Conventions

A vector is denoted by an uppercase letter with an arrow. The components are designated by the lowercase letter with the indeces x, y and z, e.g. $\vec{U} = \{u_x;u_y;u_z\}$. If an operation is applied to any component, the index is omitted. The particular bits of a component or a magnitude are identified by subscript numbers, e.g. $u = \{u_{15};u_{14};u_{13}\ldots u_0\}$. The vector length is represented by the uppercase letter without any diacritical marks. The bits of squared variables are quoted, e.g. $U^2 = \{U'_{31};U'_{30}\ldots U'_0\}$.

4 The Sign Unit (Input)

The sign unit at the inputs converts a 33 bit two's complement number *v* into a 32 bit unsigned integer *a* preceded by a sign flag *S*. Thus, the range is restricted to

$$-2^{32}+1 \le v \le 2^{32}-1 . \tag{4}$$

The sign flag is 1 if the number is negative. All sign flags are propagated through the whole circuit and passed to the sign units at the outputs.

The arithmetic operation is to invert all bits and add 1 if the highest bit is set, otherwise to leave everything unchanged. Thus:

$$a_0 = v_0 ; \tag{5}$$

$$a_1 = \bar{v}_{32}v_1 \vee v_{32}(\bar{v}_1 v_0 \vee v_1 \bar{v}_0) = \bar{v}_{32}v_1 \vee v_{32}(v_1 \oplus v_0) ; \tag{6}$$

$$a_2 = \bar{v}_{32}v_2 \vee v_{32}(\bar{v}_2(v_1 \vee v_0) \vee v_2\bar{v}_1\bar{v}_0) = \bar{v}_{32}v_2 \vee v_{32}(v_2 \oplus (v_1 \vee v_0)) ; \tag{7}$$

$$a_3 = \bar{v}_{32}v_3 \vee v_{32}(\bar{v}_3(v_2 \vee v_1 \vee v_0) \vee v_3\bar{v}_2\bar{v}_1\bar{v}_0) \tag{8}$$

$$= \bar{v}_{32}v_3 \vee v_{32}(v_3 \oplus (v_2 \vee v_1 \vee v_0)) ; \tag{9}$$

In general:

$$a_P = \bar{v}_{32}v_P \vee v_{32}(v_P \oplus (v_{P-1} \vee v_{P-2} \vee \ldots \vee v_1 \vee v_0)) ; \qquad S = v_{32} . \tag{10}$$

Gate count: 3.000

5 The Alignment Unit

In order to reduce the width of the arithmetic units, the components of the vector are uniformly scaled up or down until no component is greater than 2^{15}-1 and at least one component is greater than or equal to 2^{14}. Theoretically, no error emerges from this operation since

$$n = \frac{v \times 2^n}{\sqrt{(v_x \times 2^n)^2 + (v_y \times 2^n)^2 + (v_z \times 2^n)^2}} = \frac{v}{\sqrt{v_x^2 + v_y^2 + v_z^2}} . \tag{11}$$

However, due to the possible truncation of large vectors, a rounding error might arise. See Section 14 Error Estimation.

To describe the function of this unit we use the following abbreviations:

$$SHR17 = (a_{x_{31}} \vee a_{y_{31}} \vee a_{z_{31}}) ; \tag{12}$$

$$SHR16 = (a_{x_{30}} \vee a_{y_{30}} \vee a_{z_{30}}) \wedge \bar{a}_{x_{31}} \wedge \bar{a}_{y_{31}} \wedge \bar{a}_{z_{31}} ; \tag{13}$$

$$SHR15 = (a_{x_{29}} \vee a_{y_{29}} \vee a_{z_{29}}) \wedge \bar{a}_{x_{31}} \wedge \bar{a}_{x_{30}} \wedge \bar{a}_{y_{31}} \wedge \bar{a}_{y_{30}} \wedge \bar{a}_{z_{31}} \wedge \bar{a}_{z_{30}} ; \tag{14}$$

$$SHR14 = (a_{x_{28}} \vee a_{y_{28}} \vee a_{z_{28}}) \wedge \bar{a}_{x_{31}} \wedge \ldots \wedge \bar{a}_{x_{29}} \wedge \bar{a}_{y_{31}} \wedge \ldots \wedge \bar{a}_{y_{29}} \wedge \bar{a}_{z_{31}} \wedge \ldots \wedge \bar{a}_{z_{29}} ; \tag{15}$$

...

$$SH0 = (a_{x_{14}} \vee a_{y_{14}} \vee a_{z_{14}}) \wedge \bar{a}_{x_{31}} \wedge \ldots \wedge \bar{a}_{x_{15}} \wedge \bar{a}_{y_{31}} \wedge \ldots \wedge \bar{a}_{y_{15}} \wedge \bar{a}_{z_{31}} \wedge \ldots \wedge \bar{a}_{z_{15}} ; \quad (16)$$

$$SHL1 = (a_{x_{13}} \vee a_{y_{13}} \vee a_{z_{13}}) \wedge \bar{a}_{x_{31}} \wedge \ldots \wedge \bar{a}_{x_{14}} \wedge \bar{a}_{y_{31}} \wedge \ldots \wedge \bar{a}_{y_{14}} \wedge \bar{a}_{z_{31}} \wedge \ldots \wedge \bar{a}_{z_{14}} ; \quad (17)$$

$$\ldots$$

$$SHL14 = (a_{x_0} \vee a_{y_0} \vee a_{z_0}) \wedge \bar{a}_{x_{31}} \wedge \ldots \wedge \bar{a}_{x_1} \wedge \bar{a}_{y_{31}} \wedge \ldots \wedge \bar{a}_{y_1} \wedge \bar{a}_{z_{31}} \wedge \ldots \wedge \bar{a}_{z_1} ; \quad (18)$$

The function of the alignment circuitry is then defined by:

$$q_0 = a_0 \wedge \mathrm{SH0} \vee a_1 \wedge \mathrm{SHR1} \vee a_2 \wedge \mathrm{SHR2} \vee a_3 \wedge \mathrm{SHR3} \vee \ldots \vee a_{17} \wedge \mathrm{SHR17} ; \quad (19)$$

$$q_1 = a_0 \wedge \mathrm{SHL1} \vee a_1 \wedge \mathrm{SH0} \vee a_2 \wedge \mathrm{SHR1} \vee a_3 \wedge \mathrm{SHR2} \vee \ldots \vee a_{18} \wedge \mathrm{SHR17} ; \quad (20)$$

$$q_2 = a_0 \wedge \mathrm{SHL2} \vee a_1 \wedge \mathrm{SHL1} \vee a_2 \wedge \mathrm{SH0} \vee a_3 \wedge \mathrm{SHR1} \vee \ldots \vee a_{19} \wedge \mathrm{SHR17} ; \quad (21)$$

$$\ldots$$

$$q_{14} = a_0 \wedge \mathrm{SHL14} \vee \ldots \vee a_{13} \wedge \mathrm{SHL1} \vee a_{14} \wedge \mathrm{SH0} \vee a_{15} \wedge \mathrm{SHR1} \vee \ldots \quad (22)$$

$$\ldots \vee a_{31} \wedge \mathrm{SHR17} . \quad (23)$$

Gate count: 3.000

6 The Square Units

Since the input operands are 15 bit integers, the results are 30 bit positive numbers. We use standard multiplier networks. However, the computing pattern shows some redundancy which can be exploited to cut the required chip space by one half.
We will demonstrate the scheme for a 6 bit number.

$$q^2 = (q_5 2^5 + q_4 2^4 + q_3 2^3 + q_2 2^2 + q_1 2^1 + q_0 2^0)^2 = (q'_{11} 2^{11} + q'_{10} 2^{10} + \ldots + q'_0 2^0) . \quad (24)$$

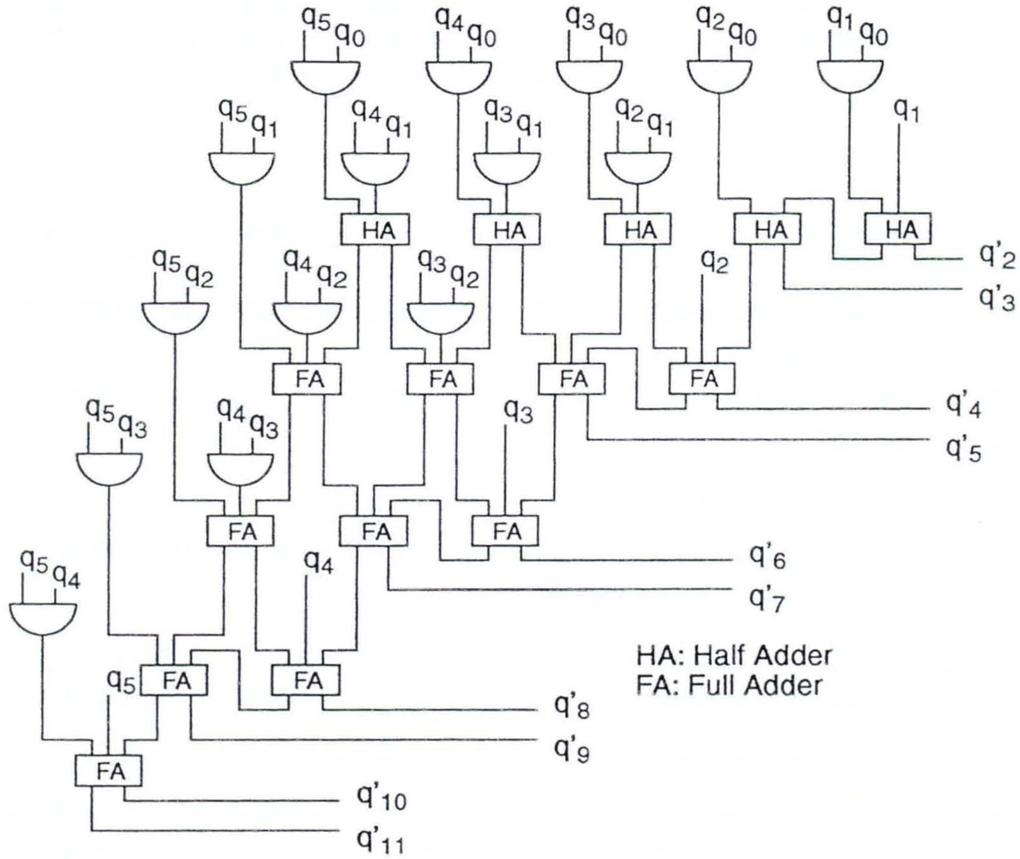
The computing pattern is shown in the following table:

q'_{11}	q'_{10}	q'_9	q'_8	q'_7	q'_6	q'_5	q'_4	q'_3	q'_2	q'_1	q'_0
						q_5q_0	q_4q_0	q_3q_0	q_2q_0	q_1q_0	q_0
					q_5q_1	q_4q_1	q_3q_1	q_2q_1	q_1	q_1q_0	
				q_5q_2	q_4q_2	q_3q_2	q_2	q_2q_1	q_2q_0		
			q_5q_3	q_4q_3	q_3	q_3q_2	q_3q_1	q_3q_0			
		q_5q_4	q_4	q_4q_3	q_4q_2	q_4q_1	q_4q_0				
	q_5	q_5q_4	q_5q_3	q_5q_2	q_5q_1	q_5q_0					

Most elements occur twice and so the table can be reorganized:

q'_{11}	q'_{10}	q'_9	q'_8	q'_7	q'_6	q'_5	q'_4	q'_3	q'_2	q'_1	q'_0
					q_5q_0	q_4q_0	q_3q_0	q_2q_0	q_1q_0		q_0
				q_5q_1	q_4q_1	q_3q_1	q_2q_1		q_1		
			q_5q_2	q_4q_2	q_3q_2		q_2				
		q_5q_3	q_4q_3		q_3						
	q_5q_4		q_4								
	q_5										

The following circuitry performs this function ($q'_0 = q_0$; $q'_1 = 0$):



Gate count: 3.500

7 The Σ - Unit

This is a standard triple input integer adder. The results are 32 bit integers. The alignment unit assures that

$$10000000H \le Q^2 \le BFFD0003H \ . \tag{25}$$

Gate count: 3.000

8 The Scale Unit

The components of the vector and the vector length are scaled so that

$$10000000H \le T^2 \le 3FFFFFFFH \qquad \text{or} \qquad 2^{14} \le T \le 2^{15} - 1 \ . \tag{26}$$

That is, the squared vector length is shifted right two places and each of the components is shifted right one digit if one of the most significant bits of the squared vector length is set. The function is then described as follows:

$$T_j = Q'_j \wedge \overline{Q}'_{31} \wedge \overline{Q}'_{30} \vee Q'_{j+2} \wedge (Q'_{31} \vee Q'_{30}) \qquad \text{for} \qquad 0 \le j \le 29, \tag{27}$$

$$t_j = q_j \wedge \overline{Q}'_{31} \wedge \overline{Q}'_{30} \vee q_{j+1} \wedge (Q'_{31} \vee Q'_{30}) \qquad \text{for} \qquad 0 \le j \le 13 \qquad \text{and} \tag{28}$$

$$t_{14} = q_{14} \wedge \overline{Q}'_{31} \wedge \overline{Q}'_{30} \ . \tag{29}$$

Again, truncation errors are subject of consideration (see Section 14 Error Estimation).
Gate count: 1.500

9 The Square Root Unit

The algorithm is first explained for decimal numbers [GKHK86]. For every two digits of an integer, the integer part of the square root has one digit. If the number of digits is odd, a zero must be placed in front of the integer. The same holds true for decimals, except that a zero must be appended if necessary. Let us consider the following example.

$$\begin{array}{ccccccc} 9\,7 & | & 2\,6 & | & 4\,1 & . & 7\,0 \\ A & & B & & C & . & D \end{array}$$

We start the calculation with the most significant digit. *A* is the integer part of the square root of 97, no matter what follows. Thus, $A = 9$. The remainder $R_A = 16$.

Now let's consider the next two digits. $10 \times A$ is still an approximation of the square root of 9726. The new remainder $R_A^* = 100 \times R_A + 26 = 1626$.

If we add *B*, the new square root is $(10 \times A + B)$. By doing so, we increase the square by $20 \times A \times B + B^2$. So we can formulate:

$$R_A^* \geq 20 \times A \times B + B^2 ; \tag{30}$$

Since $20 \times A \times B > B^2$, we will for the moment neglect B^2. A rough estimation gives:

$$B = \lfloor R_A^* / 20 \times A \rfloor = \lfloor 1626/180 \rfloor = 9 ; \tag{31}$$

If this value satisfies (30), we have found *B*. In this example, however, this is not the case and therefore we have to decrement *B* by one. Thus, $B = 8$. The remainder R_B is given by:

$$R_B = R_A^* - (20 \times A \times B + B^2) = 1626 - (1440 + 64) = 122 ; \tag{32}$$

Now we are ready for the next two digits. Again, $10 \times AB$ is still an approximation of the square root of 972641 (Don't get confused with the notation: $AB = 98$ in this example, whereas $A \times B = 72$). The new remainder $R_B^* = 100 \times R_B + 41 = 12241$.

We add C to increase the square by $20 \times AB \times C + C^2$. So we find:

$$R_B^* \geq 20 \times AB \times C + C^2 ; \tag{33}$$

Another guesstimate gives:

$$C = \lfloor R_B^* / 20 \times AB \rfloor = \lfloor 12241/1960 \rfloor = 6 ; \tag{34}$$

This time (33) is satisfied. The integer part of the root therefore is 986. For the decimals we can proceed in just the same way:

$$R_C = R_B^* - (20 \times AB \times C + C^2) = 12241 - (20 \times 98 \times 6 + 36) = 455 ; \tag{35}$$

$$R_C^* = 100 \times R_C + 70 = 44570 ; \tag{36}$$

$$R_C^* \geq 20 \times ABC \times D + D^2 ; \tag{37}$$

$$D = \lfloor R_C^* / 20 \times ABC \rfloor = \lfloor 44570/19720 \rfloor = 2 ; \tag{38}$$

The final result:

$$\sqrt{972641.7} \approx 986.2 ; \tag{39}$$

This procedure can be continued by appending pairs of zeros to the decimals until the required precision is reached.

The advantages of this method will become clear if we consider binary numbers. Again the integer part of the square root has one bit for every pair of bits of the operand. For a better readability we will denote the square bits as D_n, that is $T^2 = \{D_{29};D_{28}...D_0\}$.

Input operand	D_{29}	D_{28}	D_{27}	D_{26}	D_{25}	D_{24}	D_{23}	D_{22}	D_{21}	D_{20}	D_{19}	D_{18}	
Square root	T_{14}		T_{13}		T_{12}		T_{11}		T_{10}		T_9		

Naturally, the most significant bit T_{14} can only be 0 or 1 (accordingly, there are only two square numbers which fit into two bits: 00 and 01). Thus:

$$T_{14} = D_{29} \vee D_{28}; \qquad (40)$$

The remainder $R^{14}_{[29..28]}$ is calculated according to

$$R^{14}_{29} = D_{29} \wedge D_{28} \qquad \text{and} \qquad R^{14}_{28} = D_{29} \wedge \overline{D}_{28}; \qquad (41)$$

The square root of the binary number $D_{29}D_{28}D_{27}D_{26}$ is still approximated by $2 \times T_{14}$, the remainder $R^*_{[29..26]}$ is represented by $R^{14}_{29}R^{14}_{28}D_{27}D_{26}$.

If we add T_{13}, the square is increased by $4 \times T_{14} \times T_{13} + T^2_{13}$. So the following relation must be satisfied:

$$R^*_{[29...26]} \geq 4 \times T_{14} \times T_{13} + T^2_{13}; \qquad (42)$$

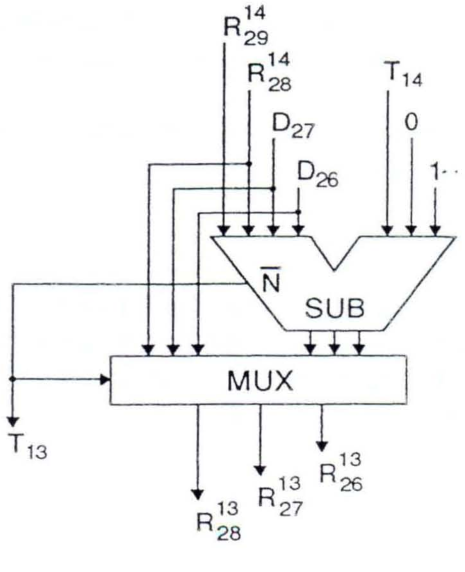
If we assume $T_{13} = 1$, then $R^*_{[29...26]} \geq 4 \times T_{14} + 1$. (43)

In other words: $T_{13} = 1$ if the above test passes, else $T_{13} = 0$. The new remainder $R^{13}_{[29..26]}$ is computed by:

$$R^{13}_{[29...26]} = R^*_{[29...26]} - 4 \times T_{14} \times T_{13} - T^2_{13}; \qquad (44)$$

That is, the remainder is left unchanged in the case $T_{13} = 0$.

This function is performed by the simple circuitry shown below. The subtractor generates the flag $\overline{\text{N}}$(egative, active low) as result bit which also controls the Multiplexer.



For clarity, we will demonstrate the computation of the next result bit, T_{12}. The new intermediate remainder $R^*_{[28...24]}$ is constructed from $R^{13}_{28}R^{13}_{27}R^{13}_{26}D_{25}D_{24}$.

If we add T_{12}, the square is increased by $4 \times T_{14}T_{13} \times T_{12} + T^2_{12}$.

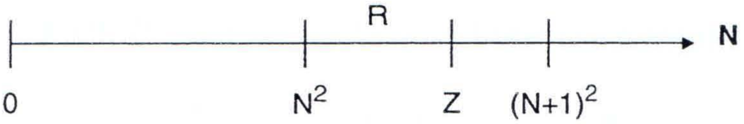
$$T_{12} = 1 \text{ if } R^*_{[28...24]} \ge 4 \times T_{14}T_{13} + 1 . \tag{45}$$

Depending on the result of this compare operation, the new remainder is either left unchanged or

$$R^{12}_{[27...24]} = R^*_{[28...24]} - 4 \times T_{14}T_{13} - 1 . \tag{46}$$

Just as we did for decimal numbers, we can repeat this calculation until the required precision is obtained.

Note that R^{13}_{29} and R^{12}_{28} have been dismissed. They are always 0. In general, the remainder has at most one digit more than the root. This can be shown as follows. Consider the integer numbers Z and N, where N is the integer part of the square root of Z.



For the remainder *R* we can formulate:

$$R \le (N+1)^2 - N^2 - 1 ; \tag{47}$$

$$R \le N^2 + 2N + 1 - N^2 - 1 ; \tag{48}$$

$$R \le 2N ; \tag{49}$$

The block diagram on the next page shows the circuitry for 30 bit integers $D_{[29..0]}$. Except for the first one a register is inserted after each stage. The multiplexers and registers at the outputs of the subtractors have been merged into a single symbol.

In general, for the calculation of the integer part of a square root of a number with *N* bits (where *N* is assumed even), we need *N/2 - 1* subtractors, starting with a 4-bit and ending with a (*N/2 + 2*) - bit subtractor. *N/2 - 2* multiplexers are also required, starting with a 3-bit, ending with a *N/2* - bit multiplexer.

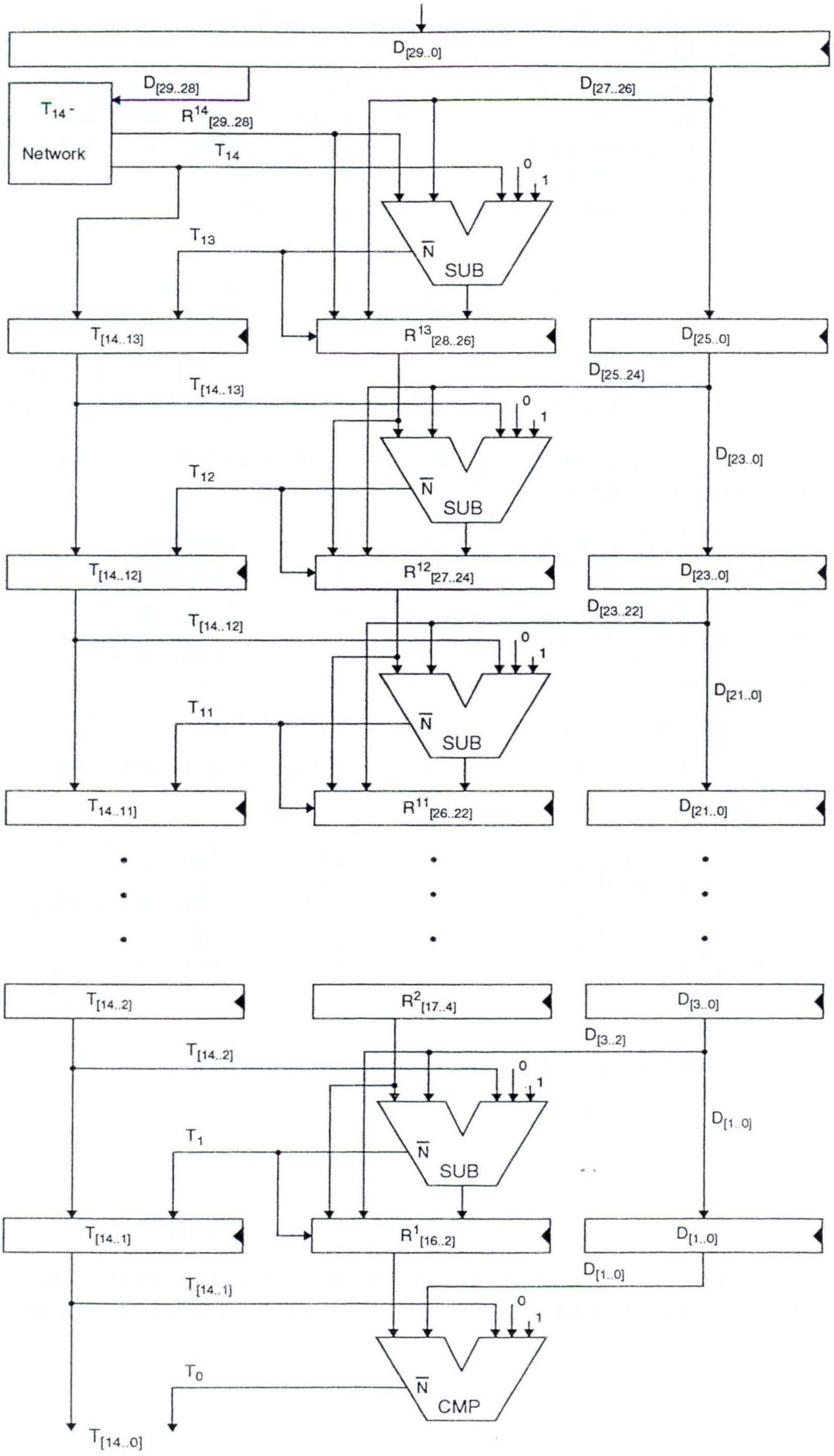
In this particular case, the Square Root Unit has 14 stages, 14 subtractors from 4 to 17 bits, 13 multiplexers from 3 to 15 bits and 433 register bits.

Gate count: 6.000.

10 Component FIFO

This is a 14 x 49 bit memory including one valid bit per vector. The components FIFO should be realized as a register pipeline (as opposed to the usual "fall-through" - architecture of FIFOs), so that the components and the vector length arrive at the same time at the inputs of the divide units without special control circuitry.

Gate count: 5.000



Square Root Pipeline

11 The Division Unit

The components *t* are taken from the component FIFO as 15 bit unsigned integers preceded by a sign bit. The vector length *T* arrives as a 15 bit unsigned integer as well. Thus, there are three unsigned division pipelines, as for example explained in [Hoff82], to be constructed. The results *{m_x;m_y;m_z}* shall be computed as 15 bit fixed point numbers.

Since $0 \leq a \leq V$, $0 \leq m \leq 1$. In the first instance we assume that

$$m = \sum_{j=0}^{-14} m_j 2^j \qquad \text{where} \qquad m = |n|. \tag{50}$$

The algorithm shall be explained by an example where $t = 011100111010011$ and $T = 100001111010111$. The quotient is computed bitwise using 16 bit two's complement arithmetic.

$m_0 = 1$ if $t - T \geq 0$. The light grey cells contain the sign extensions of the operands. The dark cell holds the inverted result bit m_0.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Bit-Position
	0	0	1	1	1	0	0	1	1	1	0	1	0	0	1	1	Component *t*
+	1	0	1	1	1	1	0	0	0	0	1	0	1	0	0	1	2's Compl. of *T*
=	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0	0	Remainder $R^0_{[15..0]}$

If $m_0 = 0$, the remainder *R⁰* must be corrected by adding *T*. Then, $R^{-1} = R^0 - T/2$, and $m_{-1} = 1$ if $R^{-1} \geq 0$. However, the same can be achieved by adding *T/2* to *R⁰* if $m_0 = 0$ and subtracting *T/2* from *R⁰* if $m_0 = 1$.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-1	Bit-Position
↓	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0	0	0	Remainder $R^0_{[15..-1]}$
+		0	1	0	0	0	0	1	1	1	1	0	1	0	1	1	1	*T/2*
=		0	0	1	0	1	1	1	1	1	1	0	0	1	1	1	1	Remainder $R^{-1}_{[14..-1]}$

Note that the result bits can be excluded from further calculation since

$$|R^0| = |t - T| \leq T; \tag{51}$$

$$|R^{-1}| = ||R^0| - T/2| \leq T/2; \tag{52}$$

$$|R^{-2}| = ||R^{-1}| - T/4| \leq T/4 \text{ and so forth.} \tag{53}$$

Thus, the width of the required ALUs remains constant throughout the complete pipeline. The computation is continued in this way until the required precision is reached. The last remainder is discarded.

However, this scheme makes no good use of the available precision. m_0 is set only in the case $t = T$. To increase the precision, we use the following format instead:

$$m = \sum_{j=-1}^{-15} m_j 2^j . \tag{54}$$

The maximum error is then reduced to 2^{-15}.
For $t = T$, m is expressed as 0.111111111111111. This is achieved by assuming $m_0 \equiv 0$ and starting the computation with the operation t - $T/2$.
The first step is given below:

	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-1	Bit-Position
	0	1	1	1	0	0	1	1	1	0	1	0	0	1	1	0	Component t
+	1	0	1	1	1	1	0	0	0	0	1	0	1	0	0	1	2's Compl. of $T/2$
=	0	0	1	0	1	1	1	1	1	1	0	0	1	1	1	1	Remainder $R^{-1}{}_{[14..-1]}$

The circuitry shown on the next side performs this function. Each pipeline stage computes one result bit. The three division pipelines consume approximately 40.000 gates.

12 The Sign Unit (Outputs)

The sign units at the outputs perform the inverse function as the sign units at the inputs, however, the arithmetic operation is the same. The 15 bit positive components m, which are preceded by a sign flag S, are converted into16 bit two's complement components n.
Again we formulate:

$$n_{-15} = m_{-15} ; \tag{55}$$

$$n_{-14} = \bar{S} m_{-14} \vee S (\bar{m}_{-14} m_{-15} \vee m_{-14} \bar{m}_{-15}) = \bar{S} m_{-14} \vee S (m_{-14} \oplus m_{-15}) ; \tag{56}$$

$$n_{-13} = \bar{S} m_{-13} \vee S (\bar{m}_{-13} (m_{-14} \vee m_{-15}) \vee m_{-13} \bar{m}_{-14} \bar{m}_{-15}) \tag{57}$$

$$= \bar{S} m_{-13} \vee S (m_{-13} \oplus (m_{-14} \vee m_{-15})) ; \tag{58}$$

...

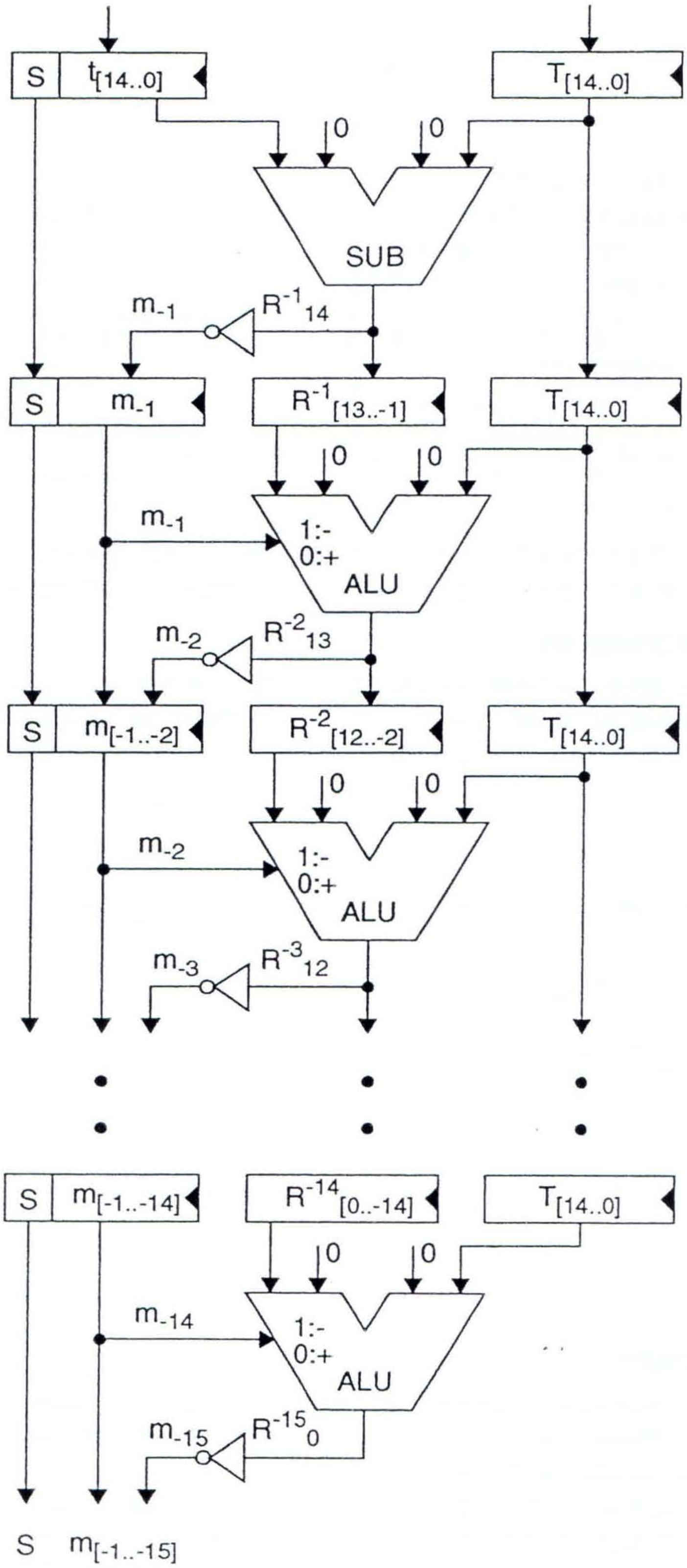
$$n_{-1} = \bar{S} m_{-1} \vee S (m_{-1} \oplus (m_{-2} \vee m_{-3} \vee \ldots \vee m_{-14} \vee m_{-15})) . \tag{59}$$

$$n_0 = S . \tag{60}$$

Gate count: 1.500

13 Control Structure

If the component FIFO is realized as a register pipeline (as opposed to the usual "fall-through"-architecture of FIFOs), there is no internal control structure required. All operands travel the same distance and so the chip just has to be clocked.
Provisions are made to freeze the pipeline. The activation of an external signal masks the clock. This circuitry is designed very carefully to avoid spikes on the internal clock lines.
The valid flags, one for each stage, must be reset during initialization. The valid flag must be held active at the inputs whenever a vector is clocked in. Normalized vectors are available as long as the valid vector output maintains an active state.
"Design-for-Testability" features are also taken into account. We use scan-path flipflops for all registers to construct one or more scan chains.



One of three Division Pipelines

14 Error Estimation

Incoming components *v* are considered to be “true values”. The normalization of $\vec{V}$ without rounding errors will give the exact unit vector $\vec{N_E}$. We will derive an error vector $\Delta\vec{N}$, so that $\vec{N} = \vec{N_E} + \Delta\vec{N}$.

The sign units at the inputs operate precision conserving, that is

$$a = |v| \ . \tag{61}$$

Depending on their size, the operands are possibly right shifted and truncated by the alignment unit and the scale unit. For simplicity, let's assume that the error Δt is defined by

$$-1 \le \Delta t \le 0 \ . \tag{62}$$

Due to this discretization of the components, a change in direction of the normalized vector might occur. Instead of the vector $\vec{V} = \{v_x;v_y;v_z\}$, the vector $\vec{T} = \{t_x;t_y;t_z\}$ is normalized. Provided this computation is carried out accurately, the maximum deviation occurs for

$$\vec{T} = \{V_{min};0;0\} \qquad \text{and} \qquad \Delta t_y = \Delta t_z = -1 \qquad \text{where} \qquad V_{min} = 2^{14} \ . \tag{63}$$

The error vector $\vec{M_D}$ is then defined by:

$$\vec{M_D} \approx \{0;-2^{-14};-2^{-14}\} \qquad \text{where} \qquad M_D = 8,63 \times 10^{-5} \ . \tag{64}$$

Any other permutation of the components in (63) and (64) yields the same result for M_D.

However, there might be an error in *T*, so that $\vec{T}$ is not scaled properly. We have to distinguish two cases:

1.) There was no shift operation in the scale unit.

T^2 is the true squared length of $\vec{T}$. However, the limited precision of the square root unit causes a truncation error ΔT. For simplicity, we assume that

$$-1 \le \Delta T \le 0 \ . \tag{65}$$

2.) The scale unit performed a right shift operation.

The squared vector length is divided by 4 and the two LSBs are discarded. After this operation, the range of T^2 is given by

$$10000000H \le T^2 \le 2FFF4000H \tag{66}$$

and therefore, the truncation error of T^2 can be neglected. So it can be said that

$$T^2 = \left(\frac{q_x}{2}\right)^2 + \left(\frac{q_y}{2}\right)^2 + \left(\frac{q_z}{2}\right)^2 \ . \tag{67}$$

On the other hand,

$$\vec{T} = \{t_x;t_y;t_z\} = \{\left\lfloor\frac{q_x}{2}\right\rfloor;\left\lfloor\frac{q_y}{2}\right\rfloor;\left\lfloor\frac{q_z}{2}\right\rfloor\} \ . \tag{68}$$

Taking the truncation error of the square root unit into account, the resulting error ΔT is then given by

$$-1 \le \Delta T \le 0.5 \times \sqrt{3} \ . \tag{69}$$

For a given ΔT, the resulting vector length *M* is given by:

$$M = \sqrt{\left(\frac{t_x}{T+\Delta T}\right)^2 + \left(\frac{t_y}{T+\Delta T}\right)^2 + \left(\frac{t_z}{T+\Delta T}\right)^2} = \sqrt{\frac{t_x^2+t_y^2+t_z^2}{(T+\Delta T)^2}} = \frac{T}{T+\Delta T} \approx 1 - \frac{\Delta T}{T} \ . \tag{70}$$

For the moment we assume that the divide units operate at infinite precision. Then the error vector $\vec{M}_S$ is given by:

$$\vec{M}_S = s \times \vec{M} \qquad \text{where} \qquad -\sqrt{3} \times 2^{-15} \le s \le 2^{-14} . \tag{71}$$

The maximum truncation error of the divide units is -2^{-15} for each component. This produces an additional error vector $\vec{M}_T$, given by:

$$\vec{M}_T = \{-2^{-15}\ldots 0; -2^{-15}\ldots 0; -2^{-15}\ldots 0\} . \tag{72}$$

The error vector $\Delta\vec{N}$ is then defined by:

$$\Delta\vec{N} = \vec{M}_D + \vec{M}_S + \vec{M}_T . \tag{73}$$

We assume further that $\vec{M}_D \perp \vec{M}_S$.

The error vector of maximum magnitude is finally given by:

$$\Delta\vec{N} = \{\pm 2^{-14}; -3 \times 2^{-15}; -3 \times 2^{-15}\} \qquad \text{and} \qquad \Delta N = 1.43 \times 10^{-4} , \tag{74}$$

or any permutation of the components. The sign units at the outputs again operate precision conserving.

15 Design Complexity

The total number of gates needed for the functional units is approximately 70.000. Assuming a 50% array utilization, which should be achievable in consideration of the regular structure of the chip, a 140.000 gates master is needed.

16 Conclusion

We presented a single-chip VLSI solution to one of the essential tasks in computer graphics, the normalization of vectors. This approach is superior over other hardware solutions such as look-up tables or micro-programmed ALUs, because it achieves maximum speed at minimum costs. Advances in VLSI technology can directly be exploited to increase clock frequency and to place multiple vector normalizers along with additional functional units onto a single chip, so that a complete Phong shader with a generation rate of 100M pixel/s will be feasible as a single-chip device in the near future.

17 Acknowledgments

This work is supervised by Prof. Strasser and is part of the advanced graphics accelerator project at WSI/GRIS, University of Tuebingen, supported partially by the CEC's ESPRIT programme. Claus Dreischer, who is currently implementing the design, gave valuable suggestions and provided the gate counts. Thanks to Andreas Schilling for many helpful discussions.

18 References

[BuiT75] **Phong Bui-Tuong,** “Illumination for Computer-Generated Pictures”, CACM, Vol. 18, No. 6, June 1975, pages 311-317

[GKHK86] **S. Gottwald, H. Küstner, M. Hellwich and H. Kästner (Edts.),** “Handbuch der Mathematik”, Buch und Zeit Verlagsgesellschaft, D-5000 Köln, 1986, pages 44-45

[Hoff82] **Rolf Hoffmann,** “Rechenwerke und Mikroprogrammierung”, Oldenbourg Verlag, D-8000 München, 1982, pages 85-96

[Knit93] **Günter Knittel,** “VERVE - Voxel Engine for Real-time Visualization and Examination”, presented at the Eurographics Conference 93, Barcelona, September 6-10, 1993